SECURITY ASSESSMENT

Deekshith A

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 (20)

Transitioning to Secure SDLC

1. Requirements Analysis

1. Conduct user interviews to gather functional requirements.
2. Write a requirements document for task management features.

Security Enhancements:

- Define security requirements (e.g., role-based access, data retention policies)
- Perform privacy impact assessment (PIA)
- Identify compliance needs (e.g., GDPR, HIPAA)
- Document abuse/misuse cases
- Include security-specific acceptance criteria

Transitioning to Secure SDLC

2. Design

1. Create a high-level architecture diagram for the application.
2. Design the database schema for tasks.

Security Enhancements:

- Perform threat modeling (e.g., STRIDE, DFDs)
- Define trust boundaries
- Choose secure design patterns
- Plan for input validation and output encoding
- Design for least privilege
- Plan for encryption (data in transit/at rest)
- Conduct security design review

Transitioning to Secure SDLC

3. Development

1. Code the user interface using HTML and CSS.
2. Implement interactive elements using JavaScript.
3. Set up a Flask application to handle API requests.
4. Implement CRUD operations for tasks.

Security Enhancements:

- Use secure coding guidelines (e.g., OWASP Top 10)
- Implement input validation and output encoding
- Use parameterized queries to prevent SQL Injection
- Secure API endpoints with proper auth/authz
- Avoid hardcoded secrets – use secret management tools
- Use code linters and pre-commit hooks
- Implement logging and monitoring hooks

Transitioning to Secure SDLC

4. Testing

1. Write and execute functional test cases.
2. Conduct browser compatibility testing.

Security Enhancements:

- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Security regression testing
- Fuzz testing and input validation tests
- Dependency scanning for vulnerable packages
- Manual code review or peer review with a security focus
- Test authentication, session management, and access control

Transitioning to Secure SDLC

5. Deployment

1. Deploy the application to Heroku.
2. .Perform smoke testing on the deployed application.

Security Enhancements:

- Secure deployment pipeline (CI/CD with security checks)
- Disable debug mode and remove test data
- Use HTTPS and secure headers
- Restrict admin panel access
- Apply runtime configuration hardening
- Use secrets management (not .env in repo)
- Validate third-party integrations and APIs

Transitioning to Secure SDLC

6. Maintenance

1. Monitor application logs and fix reported issues.
2. Gather user feedback for future feature additions.

Security Enhancements:

- Regularly patch dependencies and libraries
- Monitor for security events and alerts
- Conduct periodic vulnerability scans
- Perform incident response drills
- Audit user access and permissions
- Plan for secure decommissioning of features or services
- Continuously update the threat model

Advocating for Secure SDLC

Why Secure SDLC Matters in Modern Development

1. Proactive Threat Mitigation

Security is built-in from the start, reducing vulnerabilities early—critical for safeguarding wallets, transactions, and sensitive user data in the crypto domain.

2. Faster Response to Emerging Threats

Iterative development allows quick updates and patching, essential for adapting to fast-evolving crypto security threats and zero-day vulnerabilities.

3. Improved Compliance and Audit Readiness

Continuous security checkpoints help meet regulatory and crypto industry compliance standards like GDPR, PCI-DSS, or blockchain-specific frameworks.

4. Cost-Efficient Security Integration

Detecting and fixing issues early is far cheaper than patching post-deployment, protecting both budget and brand trust in a high-stakes crypto environment.

5. Enhanced Cross-Team Collaboration and Accountability

Embedding security in every phase fosters shared responsibility, empowering developers, QA, and DevOps to uphold security without slowing innovation.

Section 2:
Vulnerabilities and Remediation (35)

First Line of Defence : **Secure Authentication**

Vulnerabilities and remediation

Rate Limiting
Description
A login system that doesn't enforce rate limiting allows unlimited login attempts, enabling brute force or credential stuffing attacks.
Risk
Attackers can automate password guessing or test large numbers of leaked credentials, potentially gaining unauthorized access to user accounts.
Remediation
Implement rate limiting using tools like a Web Application Firewall (WAF), set thresholds per IP or user, introduce CAPTCHA after several failed attempts, and temporarily lock accounts after repeated failures.

Vulnerabilities and remediation

Remote Code Execution (RCE)
Description
If user input is executed as code due to insecure backend logic (e.g., passing input directly to system functions), attackers can run arbitrary commands on the server.
Risk
This can lead to complete server takeover, including data theft, unauthorized fund transfers, or service disruption — a catastrophic event for any platform, especially in cryptocurrency
Remediation
Avoid dynamic code execution altogether. Validate and sanitize all input rigorously, use secure libraries or APIs instead of eval() or exec(), and run applications with the least privileges necessary.

Vulnerabilities and remediation

Cross-Site Scripting (XSS)
Description
XSS occurs when an application includes untrusted user input in web pages without proper validation or encoding, allowing malicious scripts to run in users' browsers.
Risk
Attackers can hijack sessions, steal wallet data or credentials, perform unauthorized transactions, or phish users via the trusted interface.
Remediation
Encode output before rendering it in the browser (especially in HTML, JS, or attributes), sanitize input using well-tested libraries, and implement a strong Content Security Policy (CSP) to block unauthorized script execution.

Threat Matrix

Pathway (Vulnerability)	Impact Level	Likelihood Level
Rate Limiting	Medium–High	High
Remote Code Execution (RCE)	High–Critical	Medium
Cross-Site Scripting (XSS)	Medium–High	High

Note : Likelihood is the chance that a vulnerability will be discovered and exploited.

Impact → / Likelihood ↓	Critical	Medium	High
High		Rate Limiting XSS	Rate Limiting XSS
Medium	RCE	RCE	
Low			

Section 3:
Container Security (20)

Container scan Report

Container Image: vulnerables/cve-2014-6271

Tool Used: Trivy

Scan Date: 12/06/2025

Scan Scope: Vulnerabilities,Secrets,Misconfigurations

Security Levels: LOW, MEDIUM, HIGH, CRITICAL

Findings:

Vulnerabilities : 254
Secrets : 1
Misconfigurations : 0

Trivy scan Results

Target	Type	Vulnerabilities	Secrets
vulnerables/cve-2014-6271:latest (debian 7.11)	debian	254	-
/etc/ssl/private/ssl-cert-snakeoil.key	text	-	1

	CVE-2017-12133					glibc: Use-after-free read access in clntudp_call in sunrpc https://avd.aquasec.com/nvd/cve-2017-12133
	CVE-2013-2207	LOW				(pt_chown): Improper pseudotty ownership and permissions changes when granting access to the... https://avd.aquasec.com/nvd/cve-2013-2207
	CVE-2015-5180					glibc: DNS resolver NULL pointer dereference with crafted record type https://avd.aquasec.com/nvd/cve-2015-5180
	CVE-2017-15670					glibc: Buffer overflow in glob with GLOB_TILDE https://avd.aquasec.com/nvd/cve-2017-15670
	CVE-2017-15671					glibc: Memory leak in glob with GLOB_TILDE https://avd.aquasec.com/nvd/cve-2017-15671
	CVE-2017-15804					glibc: Buffer overflow during unescaping of user names with the ~ operator... https://avd.aquasec.com/nvd/cve-2017-15804
ncurses-base	CVE-2017-10684	CRITICAL		5.9-10		ncurses: Stack-based buffer overflow in fmt_entry function in dump_entry.c https://avd.aquasec.com/nvd/cve-2017-10684
	CVE-2017-10685					ncurses: Stack-based buffer overflow caused by format string vulnerability in fmt_entry function... https://avd.aquasec.com/nvd/cve-2017-10685
	CVE-2017-11112	HIGH				ncurses: Illegal address access in append_acs function https://avd.aquasec.com/nvd/cve-2017-11112
	CVE-2017-11113					ncurses: Null pointer dereference vulnerability in _nc_parse_entry function https://avd.aquasec.com/nvd/cve-2017-11113
	CVE-2017-13728		will_not_fix			ncurses: Infinite loop in the next_char function https://avd.aquasec.com/nvd/cve-2017-13728
	CVE-2017-16879					ncurses: Stack-based buffer overflow in the _nc_write_entry function https://avd.aquasec.com/nvd/cve-2017-16879
	CVE-2017-13729	MEDIUM				ncurses: Illegal address access in the _nc_save_str function https://avd.aquasec.com/nvd/cve-2017-13729
	CVE-2017-13730					ncurses: Illegal address access in the function _nc_read_entry_source() https://avd.aquasec.com/nvd/cve-2017-13730
	CVE-2017-13731					ncurses: Illegal address access in the function postprocess_termcap()

Download The complete report here : Trivy_Scan_Report

Report to Fix Major Container Issues

Vulnerability Name	Unpatched Software Version	Patched Software Version
apache2:CVE-2018-1312	2.2.22-13+deb7u12	2.2.22-13+deb7u13
libapr1:CVE-2017-12613	1.4.6-3+deb7u1	1.4.6-3+deb7u2
libprocps0:CVE-2018-1126	1:3.3.3-3	1:3.3.3-3+deb7u1
libssl1.0.0:CVE-2017-3735	1.0.1t-1+deb7u2	1.0.1t-1+deb7u3
openssl: CVE-2017-3735	1.0.1t-1+deb7u2	1.0.1t-1+deb7u3
procps:CVE-2018-1126	1:3.3.3-3	1:3.3.3-3+deb7u1
sensible-utils:CVE-2017-17512	0.0.7	0.0.7+deb7u1

Section 4:
API Security (15)

API Vulnerabilities and remediation

1. Broken Object Level Authorization (BOLA)
Description
This vulnerability occurs when an API fails to properly verify if a user has permission to access a specific resource or data object. Attackers could manipulate object IDs or parameters to access other users' confidential information.
Risk
Since the API will handle sensitive user data, unauthorized access could expose personally identifiable information (PII), behavioral data, or financial details to unauthorized parties, including malicious users or vendors beyond intended scope.
Remediation
Implement strict authorization checks on every endpoint that accesses user data. Use user identity tokens (e.g., OAuth JWTs) to verify ownership or permission for requested resources. Avoid relying on client-supplied IDs without validation; map resource access based on authenticated user context. Perform thorough testing, including authorization fuzzing and penetration tests.

API Vulnerabilities and remediation

2.Lack of Proper Data Encryption (In Transit and At Rest)
Description
If API communication or stored data is not encrypted, confidential user information can be intercepted (man-in-the-middle attacks) or stolen from storage.
Risk
Sensitive user data (such as PII, usage patterns, or contact info) can be exposed during network transmission or through data breaches, leading to privacy violations, regulatory penalties (e.g., GDPR, CCPA), and loss of customer trust.
Remediation
Enforce HTTPS/TLS for all API requests and responses to protect data in transit. Use strong encryption (AES-256 or equivalent) for sensitive data stored in databases or logs. Implement secure key management policies. Avoid logging sensitive data unnecessarily, or mask/redact sensitive fields. Periodically audit encryption configurations and update certificates.

API Vulnerabilities and remediation

3. Excessive Data Exposure
Description
APIs may return more data than necessary for a given request, unintentionally leaking sensitive user attributes or metadata.
Risk
Third-party vendors or attackers could gain access to unnecessary private information, increasing privacy risks and attack surface.
Remediation
Follow the principle of least privilege and data minimization: only share data strictly required for the vendor's function. Design API responses with explicit data schemas controlling which fields are returned. Use API gateways or middleware to filter and sanitize outgoing data. Conduct threat modeling and data flow analysis during design to identify and restrict sensitive fields. Provide data access roles and scopes to control what external parties can query.