4. PROJECT DESIGN

4.1 Overview of Nessus

Nessus is a popular vulnerability scanner used to detect potential security flaws in systems. It is frequently utilized for network vulnerability assessments, offering real-time scanning to uncover vulnerabilities, misconfigurations, missing patches, and other security weaknesses. In the context of a project like "Leveraging Real-Time Security Intelligence for Enhanced Defense," Nessus can provide valuable insights that empower security teams to enhance their defenses. Here's how Nessus can be leveraged in such a project:

1. Real-Time Vulnerability Detection: Nessus continuously monitors for known vulnerabilities, outdated software, configuration issues, and security gaps. Integrating it into your security infrastructure ensures that vulnerabilities are quickly identified and updates are immediately provided, allowing security teams to act swiftly and mitigate potential risks.

2. Comprehensive Scanning Across Systems: Nessus can scan a variety of systems, such as servers, workstations, firewalls, and network devices. It comes with built-in plugins that detect common vulnerabilities, misconfigurations, and compliance violations. This feature enables ongoing monitoring of your network and helps pinpoint any security holes that need to be addressed.

3. Risk Prioritization: Nessus doesn’t only identify vulnerabilities, it also categorizes them by their severity (e.g., critical, high, medium, low). This classification is essential for real-time security intelligence, as it helps prioritize which vulnerabilities pose the greatest risk to your organization. This allows you to focus resources on addressing the most severe vulnerabilities first.

4. Integration with SIEM Systems: When integrated with Security Information and Event Management (SIEM) systems, Nessus vulnerability data can be correlated with other security events, such as suspicious activity or intrusion attempts. This integration provides deeper insights, facilitating a comprehensive and proactive defense strategy that can quickly detect and address emerging threats.

5. Automated Patch Management: Nessus can identify missing patches and outdated software, which can become easy targets for attackers. By integrating Nessus with patch management tools, you can automate the patching process to ensure that systems are updated promptly and reduce exposure to security risks.

6. Compliance Checks and Reporting: Nessus can help organizations assess whether they are compliant with industry standards and regulations such as PCI DSS, HIPAA,

and GDPR. In real time, Nessus evaluates whether systems meet the required compliance criteria and generates reports to guide remediation efforts and strengthen security posture.

7. Ongoing Monitoring for Proactive Defense: Continuous monitoring is a core element of real-time security intelligence. Nessus can be used to maintain a steady flow of vulnerability scans, ensuring that as new threats or vulnerabilities are discovered, systems are promptly re-scanned and mitigated.

8. Automated Scanning and Action: Nessus supports automated vulnerability scans based on schedules or event triggers, reducing the need for manual oversight. This automation is key for real-time intelligence systems, which rely on rapid responses to emerging threats. Nessus can be configured to automatically alert personnel, initiate patching, or take other corrective actions when vulnerabilities are detected.

9. Integration with Threat Intelligence: Nessus can integrate with external threat intelligence feeds to enrich its vulnerability scanning capabilities. By incorporating real-time threat intelligence, Nessus can identify vulnerabilities that are actively being exploited, enabling you to take immediate action and defend against these evolving threats.

Example Workflow for the Project

1. Data Collection: Nessus continuously scans and collects data on vulnerabilities across systems in real time.

2. Vulnerability Prioritization: Nessus assigns a severity score to each detected vulnerability, enabling the system to prioritize which issues require urgent attention.

3. Integration with SIEM/Threat Intelligence: Vulnerability data is pushed to a SIEM system or directly linked with threat intelligence feeds, facilitating correlation with other security events and giving a broader view of the threat landscape.

4. Patch Management and Remediation: Nessus flags missing patches and outdated software. Automated patch management tools then address these issues, reducing system vulnerabilities.

5. Feedback Loop: As new vulnerabilities are discovered or data updates become available, the system continuously updates Nessus, ensuring that new risks are swiftly identified and mitigated.

By incorporating Nessus into your "Leveraging Real-Time Security Intelligence for Enhanced Defense" project, you can ensure that vulnerabilities are identified in real time and provide your security teams with the insights needed to enhance your organization's defense mechanisms and minimize the risk of security breaches.

4.2 Proposed Solution

To address the identified vulnerabilities, a structured testing and mitigation approach was implemented. The testing involved manual and automated techniques using tools like Burp Suite, Nessus, Snort, and OWASP ZAP. Each vulnerability was assessed based on its impact, exploitability, and potential mitigation strategies. The primary focus was to identify security flaws, understand their root causes, and apply the best security practices to eliminate or reduce risk. The vulnerabilities were categorized into five key types, each requiring specific countermeasures.

Identified Vulnerabilities & Their Solutions:

1. Cross-Site Scripting (XSS):

   Findings: Input fields allowed JavaScript execution, leading to potential session hijacking and data theft.

   Mitigation:

- Implemented Content Security Policy (CSP) to block unauthorized script execution.
- Used input sanitization to remove harmful script injections.
- Applied server-side validation to filter user input.

2. Security Misconfigurations:

   Findings: Missing security headers, exposed directories, and use of default credentials.

   Mitigation:

   - Configured HTTP security headers (X-Frame-Options, X-Content-Type-Options).
   - Disabled directory listing and removed sensitive debug information.
   - Enforced strong authentication policies to prevent unauthorized access.

3. SQL Injection:

   Findings : application login mechanism is vulnerable to SQL Injection,system failed to validate and sanitize user inputs

Mitigation :

- Use prepared statements (parameterized queries) to prevent SQL Injection.
- Frameworks such as SQLAlchemy (Python) or Hibernate (Java) provide built-in security mechanisms.
- Deploy a WAF to detect and block malicious SQL queries.

4. Intrusion detection using snort :

Findings : Snort IDS was deployed to detect malicious activity, system successfully identified and logged SQL Injection attack.

Mitigation :

- Adjust Snort rules to reduce false positives while maintaining detection accuracy.
- Implement a Security Information and Event Management (SIEM) system to correlate alerts.
- Configure automated blocking mechanisms for detected SQL Injection attempts.

5. Cross- Site Request Forgery:

Findings: The application did not verify whether requests were initiated by an authenticated user, allowing attackers to trick users into executing unauthorized actions.The session cookies were automatically included in cross-origin requests, enabling unauthorized state-changing actions.

Mitigation :

- Implemented CSRF Tokens to validate legitimate user actions and prevent forged requests.
- Configured `SameSite=Strict` and `SameSite=Lax` cookie attributes to block cross-origin request sharing.

- Restricted HTTP methods by ensuring critical actions (e.g., fund transfers) only accept P0ST requests.

Testing and findings :

A series of manual and automated security tests were conducted to identify vulnerabilities in the system. The testing included input validation checks, authentication analysis, API security assessment, and configuration audits. Tools like Burp Suite, OWASP ZAP, and Nessus were used to simulate attacks. The findings revealed five key security flaws, which are detailed below along with their testing methodologies and impact.

1. Cross-Site Scripting (XSS)

- Testing Approach: Injected malicious JavaScript into input fields to check for execution.
- Findings: Detected unfiltered user input, allowing attackers to execute scripts in users' browsers, leading to session hijacking and content manipulation.

2. Security Misconfigurations

- Testing Approach: Scanned for missing security headers, default credentials, and open directories.
- Findings: Found exposed admin panels, missing HTTP security headers, and publicly accessible sensitive files, increasing the risk of unauthorized access.

3. SQL Injection:

- Findings : application login mechanism is vulnerable to SQL Injection,system failed to validate and sanitize user inputs
- Testings : Found vulnerability in login, missing firewall, and login credentials opened and accessed using sql injection

4. Cross Site Request Forgery (CSRF)

- Testing Approach: Attempted to perform unauthorized actions on behalf of an authenticated user by crafting malicious requests from an external website.
- Findings: The application did not validate the origin of requests, allowing unauthorized fund transfers without user interaction.

5. Intrusion detection using snort:

- Findings : The kali vm was open to attacking, the ubuntu machine is for detecting.

- Testings : the vulnerabilities were detected, the risk factor was displayed, the packet transfer information was also given.