|  |  |
| --- | --- |
| Name: | Ronit Gupta |
| Lab User ID: | 23SEK3324\_U12  528646462716 |
| Date: | 10/01/2024 |
| Application Name: | Wrong secrets  https://github.com/OWASP/wrongsecrets.git |

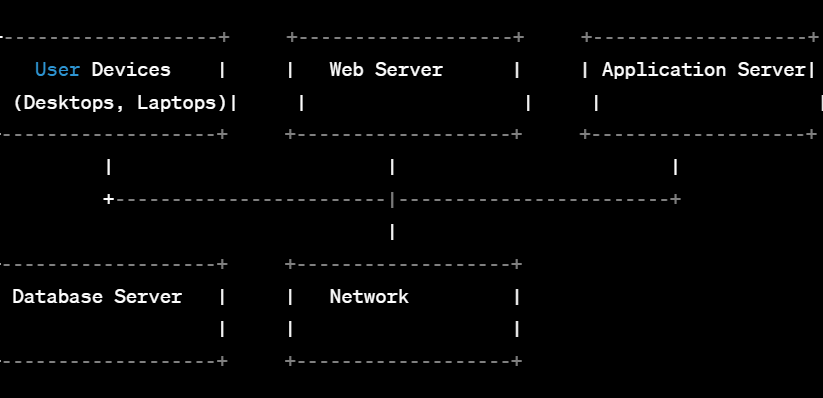
**Follow the below guidelines:**





System Architecture:

(Understand the system and document the physical and logical architecture of the system, use the shapes and icons to capture the system architecture)

**Physical Architecture:**

**Logical Architecture:  
  
  
**

Define system’s normal behavior:

(Define the steady state of the system is defined, thereby defining some measurable outputs which can indicate the system’s normal behavior)

OWASP WrongSecrets is the first Secrets Management-focused vulnerable/p0wnable app! It can be used as a stand-alone game, as part of security trainings, awareness demos, as a test environment for secret detection tools, and bad practice detection tooling. It even has a supporting CTF platform to play the game in a larger group.

WrongSecrets is based on Java, Docker, Terraform, and a bit of scripting fun. It contains more than 40 exercises with various wrongly stored or misconfigured secrets - which you need to find. Finding these secrets will

Help you to look for secrets being misconfigured at your own environment, or target environments for bug bounties.

Help you to re-evaluate your own secrets management practices as well.

In this state, the application behaves in a manner typical of a web application that is not actively exploited or under attack.

**User Interaction:**  
Users can access the application through a web interface using standard web browsers. The user interface is designed to facilitate user interaction, data input, and navigation.

**Functionalities:**The application provides specific functionalities relevant to its purpose, which may include user authentication, data processing, and other features depending on the application's intended use case.

**Data Processing:**  
During steady state, the application processes data in a normal and expected manner. It handles inputs, performs necessary computations, and produces appropriate outputs without the influence of security exploits.

Hypothesis:

(During an experiment, we need a hypothesis for comparing to a stable control group, and the same applies here too. If there is a reasonable expectation for a particular action according to which we will change the steady state of a system, then the first thing to do is to fix the system so that we accommodate for the action that will potentially have that effect on the system. For eg: "If one of our database servers fails, our service will automatically switch to a backup server, and users will not experience any downtime or data loss.")



**Hypothesis**: The WrongSecret application contains well-known vulnerabilities such as SQL injection, cross-site scripting (XSS), or insecure direct object references (IDOR).

**Rationale**: As an intentionally vulnerable application, WrongSecret is expected to include common vulnerabilities for educational and testing purposes.

**Hypothesis**: Specific variants or evasion techniques of known vulnerabilities may exist in WrongSecret that are not covered by standard security tools.

**Rationale**: Despite being aware of common vulnerabilities, variations or evasions might not be fully addressed in testing scenarios.

**Known**

Things we are aware of but don’t understand.

Things we are aware of and understand.

**Hypothesis**: Users might discover undocumented features in WrongSecret that could potentially lead to security vulnerabilities.

**Rationale**: Documentation may not cover every aspect of the application, and hidden functionalities could introduce security risks.

**Hypothesis**: There could be undiscovered vulnerabilities in WrongSecret that go beyond the typical scope of security testing.

**Rationale**: Unforeseen interactions or code paths may lead to security issues not explicitly tested for.

**Unknown**

**Unknown**

**Known**

Things we are neither aware of nor understand.

Things we understand but are not aware of.

**Vulnerability analysis using trivy:**

root@ip-172-31-0-232:~# trivy repo https://github.com/OWASP/wrongsecrets

-----

pom.xml (pom)

Total: 6 (UNKNOWN: 0, LOW: 1, MEDIUM: 2, HIGH: 1, CRITICAL: 2)

A screenshot of a computer program

Description automatically generated

**Vulnerability analysis using snyk:**

A screenshot of a computer

Description automatically generated

**Some critical vulnerability analysis:**

**1. Use of a Broken or Risky Cryptographic Algorithm:** The use of a broken or risky cryptographic algorithm poses a significant security risk to applications and systems. Cryptographic algorithms are fundamental in ensuring the confidentiality, integrity, and authenticity of sensitive information. When a cryptographic algorithm is deemed broken or risky, it means that vulnerabilities or weaknesses have been identified that could be exploited by attackers, undermining the security guarantees provided by cryptography.

**Mitigation Techniques:**

\* Stay Informed:  
Regularly monitor security advisories, cryptographic community announcements, and industry best practices to stay informed about the security status of cryptographic algorithms.

\* Algorithm Agility:  
Design systems with algorithm agility in mind, allowing easy replacement of cryptographic algorithms as security requirements evolve.

\* Deprecate Weak Algorithms:  
Explicitly deprecate and disallow the use of known weak or broken cryptographic algorithms in system configurations and policies.

**2. Use of Password Hash With Insufficient Computational Effort:** The use of password hashes with insufficient computational effort, often referred to as "weak hashing," makes it easier for attackers to conduct brute-force attacks or use precomputed tables (rainbow tables) to crack hashed passwords. Inadequate computational effort means that the hashing algorithm is too fast or not resistant enough to withstand various types of attacks.

**Mitigation Techniques:**

\* Use Strong Hashing Algorithms:  
Choose cryptographic hash functions designed for password hashing, such as bcrypt, Argon2, or scrypt. These algorithms are intentionally slow and computationally intensive, making them resistant to brute-force and rainbow table attacks.

\* Stay Informed About Best Practices:  
Stay informed about current best practices for password hashing, as cryptographic recommendations may evolve over time. Follow industry standards and guidelines provided by organizations like NIST.

\* Monitor for Anomalies:  
Implement monitoring mechanisms to detect unusual patterns of login attempts or other activities that may indicate a brute-force attack.

**3. Use of Hardcoded Credentials:** The use of hardcoded credentials refers to the practice of embedding usernames, passwords, or other authentication credentials directly into the source code or configuration files of an application. This insecure practice poses significant security risks and can lead to unauthorized access if the credentials are discovered by attackers.

**Mitigation Techniques:**

\* Environment Variables:  
Utilize environment variables or configuration files external to the source code to store sensitive information, allowing for easier updates and rotations without modifying the code.

\* Token-Based Authentication:  
Consider using token-based authentication mechanisms, such as OAuth tokens, instead of hardcoded usernames and passwords.

\* Least Privilege Principle:  
Assign the least privilege necessary for applications and services. Avoid using overly permissive credentials, even if they are hardcoded.

**4. SQL Injection (SQLi):** SQL Injection occurs when an attacker injects malicious SQL code into user inputs, leading to unauthorized access or manipulation of the database.

**Mitigation Techniques:**

\* Use parameterized queries or prepared statements to sanitize input data.

\* Implement proper input validation and parameterization to prevent direct user input in SQL queries.

\* Employ web application firewalls (WAFs) to detect and block SQL injection attempts.

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

Steps:

1. Install docker using the command sudo apt install docker.io.

2. Run:  
docker run -p 8080:8080 jeroenwillemsen/wrongsecrets:latest-no-vault

3. . You can then access the web application at http://127.0.0.1:8080

**Output snapshot**

Application was live on 13.233.160.187:8080

