SECURITY EXERCISE

Assessment

TABLE OF CONTENTS

TABLE OF CONTENTS.................................. 2
Summary 3
problem statement.. 4
Description... 4
Outcome 4
approach.. 5
Security observation 8
Source Code Review.................................... 10
Dynamic Testing... 14
Tooling.. 14
Automation 16

SUMMARY

During the assessment I mainly targeted below aspects of security

- **Source Code review** to identify common security vulnerabilities which includes but not limited to Session Fixation, Security Misconfiguration
- **Composition Analysis** to identify vulnerable components used in the service
- **Security Unit Test cases** to verify certain scenario during the build [not implemented]
- **Dynamic Testing** to verify certain source code findings and service behavior to abuse scenarios

I articulated my approach in below sections and shared details of the each stage.

1. Problem Statement
2. Approach
3. Security Observations
4. Prevention/Automation

PROBLEM STATEMENT

DESCRIPTION

Build a basic python application with flask mysql which has basic functionalities like

1. Sign up
2. Login
3. Add wish
4. Delete wish
5. Update wish

And containerize the application using docker/docker-compose

OUTCOME

- Working python application running inside containers
- Updated code which doesn't change the functionalities but correct the assessed security flaws.
- Production deployment scenarios

APPROACH

Source Code Review

- Data Validation
- AuthZ/AuthN
- Malicious Patterns
- Session Management
- Cryptography
- Business and Design
- Data Management

Dynamic Testing

- AuthN/AuthZ
- Input Validation
- Business Logic
- Testing Weak cryptography
- Configuration & Deployment Management

Design Review

- Design follows organizations policies
- Deployment & Infra consideration
- Asset Classification
- Security Architecture
- Security Principles

Containerize the app

- Docker security best practices
- Clair
- Sysdig-falco
- Docker-bench
- bandit

Source Code Review

Securing an application begins at various stages in SDLC usually starts with an idea which I followed, before approaching dynamic testing/endpoints I started analyzing the source code and understand what exactly is the application

- **Business logic & design review:**
    - **Code – Structure**

```
├── flas_app/
│   ├── app/
│   │   ├── templates/
│   │   │   ├── addwish.html
│   │   │   ├── index.html
│   │   │   ├── signup.html
│   │   │   ├── error.html
│   │   │   ├── signin.html
│   │   │   ├── userhome.html
│   │   ├── static/
│   │   │   ├── css/
│   │   │   ├── js/
│   │   ├── app.py
│   │   ├── Dockerfile
│   │   ├── config.py
│   │   ├── requirements.txt
│   ├── db/
│   │   ├── init.sql
│   ├── docker-compose.yml
│   └── Readme.MD
```

OWASP ASVS:

1. Authentication Requirements – Since the service exposes login endpoint
    a. Should work on POST with secure communication i.e. HTTPS
    b. Input Validation
    c. Password Policies
    d. SQLi*
    e. Verify that APIs are protected with Auth mechanism
2. Session Management
    a. Session should not be exposed in URL parameters or error messages
    b. Session token is random
    c. New Session token is created after authentication
    d. Session Management
    e. Securing session cookies
3. Access Control
    a. Anti-CSRF since state change or user creation is involved
    b. Protection against brute force
4. Validation
    a. Parameter Pollution
    b. Input validation using whitelist
5. Error
    a. Sensitive information is not logged
    b. Exception handling is not printing stack trace
6. API security
    a. Security Headers
    b. Cookies

c. JSON schema validation

Security in RESTAPIS

1. HTTPS endpoints
2. Access Control
3. JWT
4. API Keys
5. Restrict HTTP Methods
6. Input Validations
7. Validate content Types
8. Management API
9. Error Handling, Auditing
10. Security Headers
11. Sensitive Information in HTTP request

SECURITY OBSERVATION

Risk Assessment: Below risk based risk matrix is used to calculate the risk, inherited risk of the application depends on the context, whether the application is exposed to public, no of users etc.

	Impact		
Likelihood	Low	**->Moderate<-**	High
Low	Note	Low	Moderate
->Moderate<-	Low	**->Moderate<-**	High
High	Moderate	High	Critical

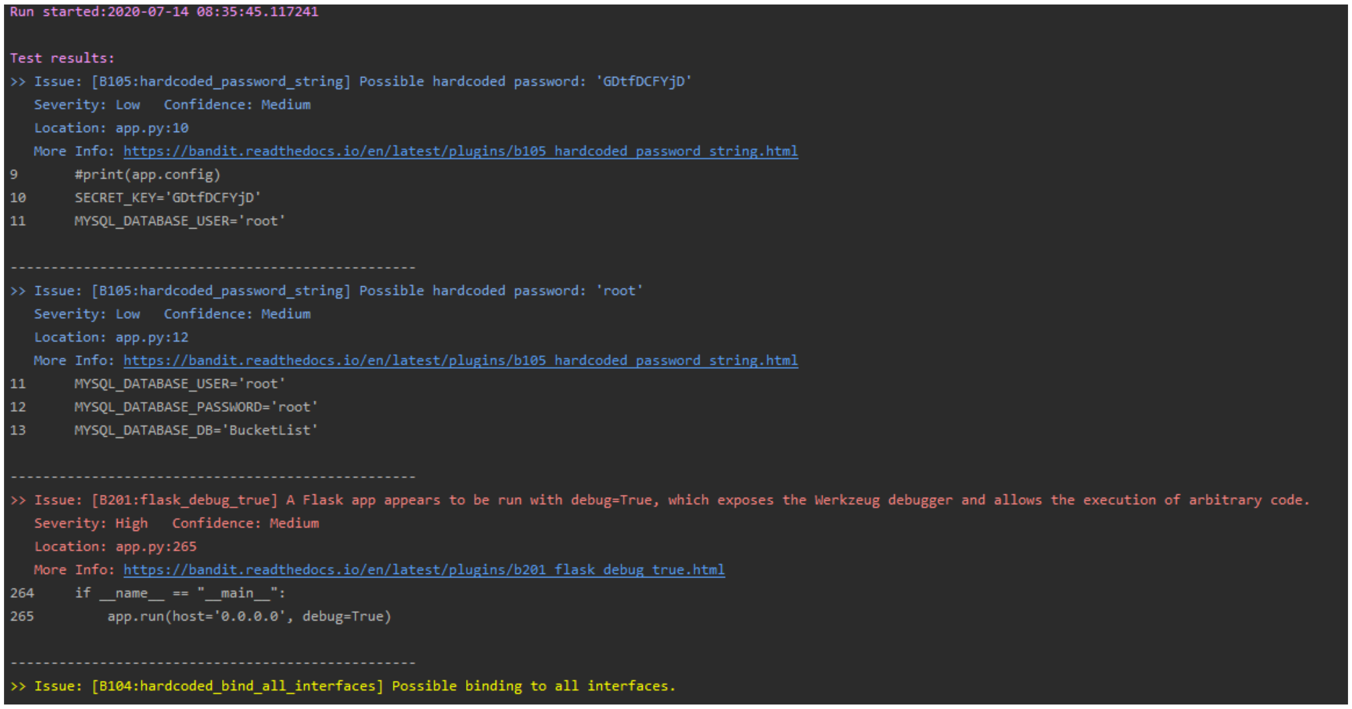
SECURITY OBSERVATIONS

Security Exercise — May 4, 2020

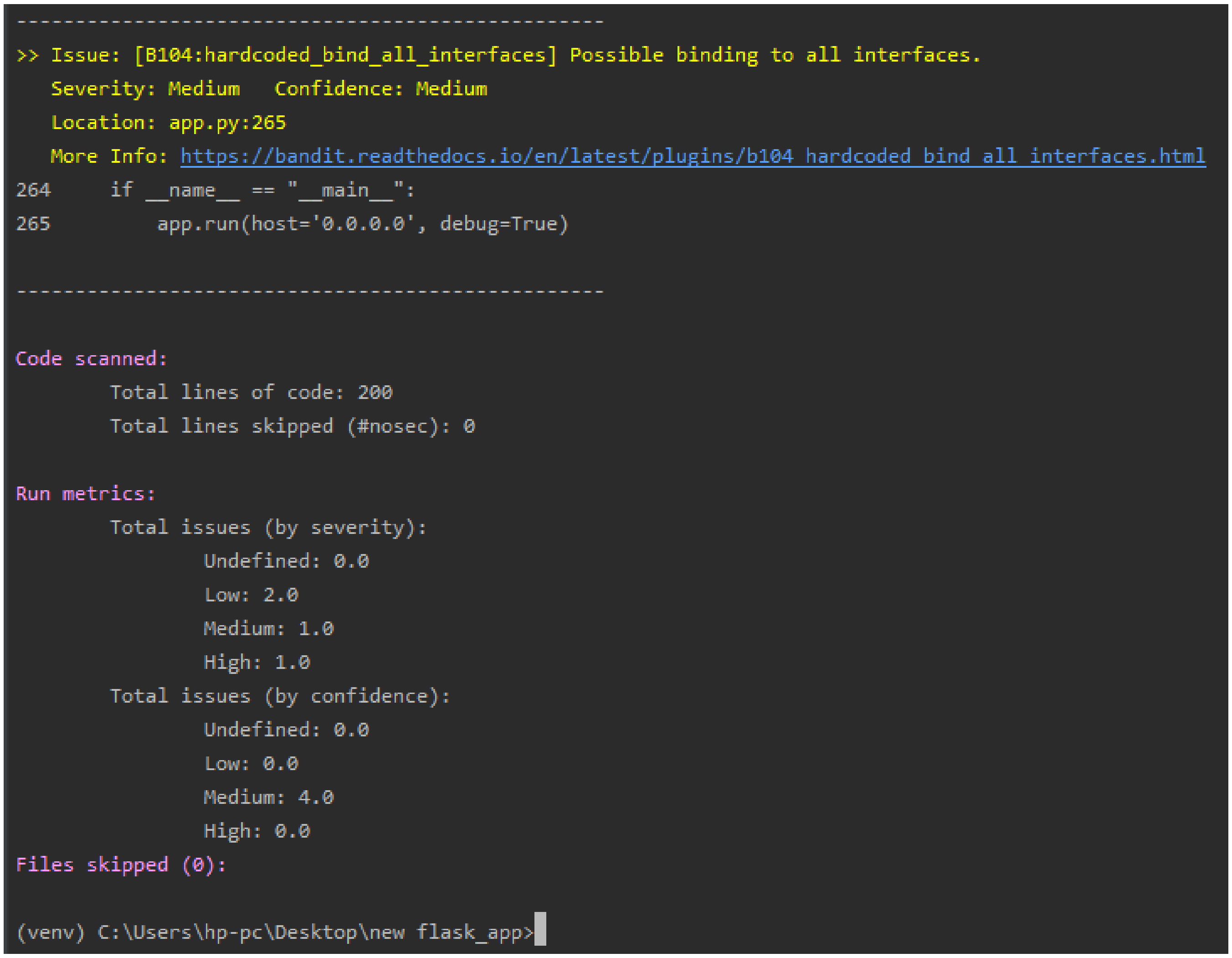
DESCRIPTION	OWASP CATEGORY	AFFTECTED PATH	RISK	OBSERVATION, IMPACT & REMEDIATION
Session Fixation Attack	Session Management	localhost/login	CRITICAL	**Observation**: Session ID from anonymous user is passed during login **Impact**: an attacker can forge a sessionID and send it to the victim, after the successful login attacker can impersonate as the authenticated user **Remediation**: Invalidate any existing session identifiers prior to authorizing new sessions. Implement JWT to maintain session in REST framework
Missing Input Validation	Input Validation	localhost/login localhost/singup localhost/addwish	MEDIUM	**Observation**: No input validation is implemented for user input fields **Impact**: Lack of input validation may lead to command injection, XSS vulnerabilities **Remediation**: Apply input validation checks for every user input especially follow white list approach
Vulnerable dependencies	Dependency Management	Source Code	MEDIUM	**Observation**: It was observed that many of the dependency used were outdated and vulnerable = **Impact**: Packages with knows vulnerabilities may introduce additional security vulnerabilities if exploited **Remediation**: Upgrade the vulnerable dependency with latest version
Missing Security Headers	Security Misconfiguration	Source Code/Dynamic testing localhost/	HIGH	**Observation**: It was observed that Security headers were not configured **Impact**: If Security headers are not set in response, an attacker can exploit underlying vulnerabilities such as XSS **Remediation**: Implement Security Headers/CSP either at gateway level or code
Lack of rate limiting	Broken Authentication	localhost/	HIGH	**Observation**: It was observed that application did not have API rate limiting implemented **Impact**: An attacker can leverage this vulnerability to brute force the authentication mechanism or perform Denial of Service attack

				Remediation: Implement rate limiting either at gateway layer or within the service
Lack of CSRF protection	CSRF	Localhost/	**CRITICAL**	**Observation**: It was observed that anti-CSRF token is not implemented **Impact**: A successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth **Remediation**: Referrer Header, CSRF token
Lack of strong Password policies	Broke Authentication	Signup	**HIGH**	**Observation**: It was observed that the strong password policies is not enforced **Impact**: If strong password policy is not enforced, an attacker can brute force weak password easily **Remediation**: Enforce strong password policies
SQL Injection	Injection	login	**HIGH**	**Observation**: It was observed that the application is not using parameterized statements instead stored procedures are used. **Impact**: Stored procedures may not stop sql injection attacks **Remediation**: Use prepared statements
Sensitive information disclosure – Hardcoded password	Security misconfigurati on	App.py	**INFO**	**Observation:** It was observed that password is hardcoded in the application **Impact:** Password should never be shared in a response **Remediation:** Remove password field from response

SOURCE CODE REVIEW



```
Run started:2020-07-14 08:35:45.117241

Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'GDtfDCFYjD'
   Severity: Low   Confidence: Medium
   Location: app.py:10
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b105_hardcoded_password_string.html
9       #print(app.config)
10      SECRET_KEY='GDtfDCFYjD'
11      MYSQL_DATABASE_USER='root'

--------------------------------------------------
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'root'
   Severity: Low   Confidence: Medium
   Location: app.py:12
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b105_hardcoded_password_string.html
11      MYSQL_DATABASE_USER='root'
12      MYSQL_DATABASE_PASSWORD='root'
13      MYSQL_DATABASE_DB='BucketList'

--------------------------------------------------
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
   Severity: High   Confidence: Medium
   Location: app.py:265
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b201_flask_debug_true.html
264     if __name__ == "__main__":
265         app.run(host='0.0.0.0', debug=True)

--------------------------------------------------
>> Issue: [B104:hardcoded_bind_all_interfaces] Possible binding to all interfaces.
```



```
--------------------------------------------------
>> Issue: [B104:hardcoded_bind_all_interfaces] Possible binding to all interfaces.
   Severity: Medium   Confidence: Medium
   Location: app.py:265
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b104_hardcoded_bind_all_interfaces.html
264     if __name__ == "__main__":
265         app.run(host='0.0.0.0', debug=True)

--------------------------------------------------

Code scanned:
        Total lines of code: 200
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0.0
                Low: 2.0
                Medium: 1.0
                High: 1.0
        Total issues (by confidence):
                Undefined: 0.0
                Low: 0.0
                Medium: 4.0
                High: 0.0
Files skipped (0):

(venv) C:\Users\hp-pc\Desktop\new flask_app>
```

2. No Input Validation

Input validation is one of the most effective technical controls for application security. It can mitigate numerous vulnerabilities including cross-site scripting, various forms of injection, and some buffer overflows. Input validation is more than checking form field values.

All data from users needs to be considered untrusted. Remember one of the top rules of secure coding is “Don’t trust user input”. Always validate user data with the full knowledge of what your application is trying to accomplish.

Input validation regex for post user endpoint

3. Session Management Implementation

1. Session ID name and fingerprinting - The name used by the session ID should not be extremely descriptive nor offer unnecessary details about the purpose and meaning of the ID.

   It is recommended to change the default session ID name of the web development framework to a generic name, such as `id`.

2. Session ID Length - The session ID must be long enough to prevent brute force attacks, where an attacker can go through the whole range of ID values and verify the existence of valid sessions.

   The session ID length must be at least `128 bits (16 bytes)`

3. Session Entropy -The session ID value must provide at least `64 bits` of entropy (if a good PRNG is used, this value is estimated to be half the length of the session ID).

4. Session ID content value - The session ID content (or value) must be meaningless to prevent information disclosure attacks, where an attacker is able to decode the contents of the ID and extract details of the user, the session, or the inner workings of the web application

4. Session Management Implementation

1. Secure Attribute - The `Secure` cookie attribute instructs web browsers to only send the cookie through an encrypted HTTPS (SSL/TLS) connection

2. HttpOnly Attribute -The `HttpOnly` cookie attribute instructs web browsers not to allow scripts (e.g. JavaScript or VBscript) an ability to access the cookies via the DOM document.cookie object. This session ID protection is mandatory to prevent session ID stealing through XSS attacks.

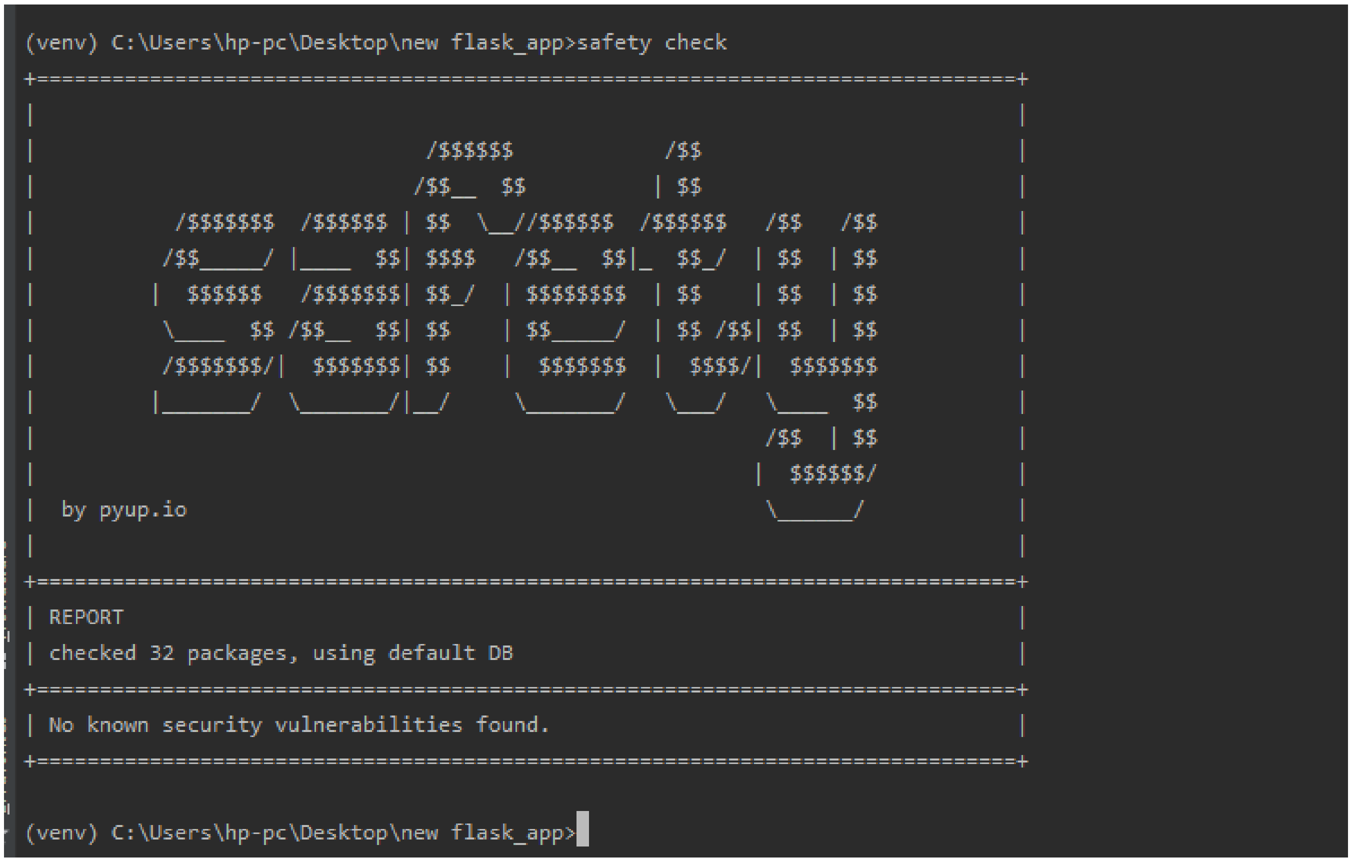
3. Same Site – SameSite allows a server define a cookie attribute making it impossible to the browser send this cookie along with cross-site requests

   The main goal is mitigate the risk of cross-origin information leakage, and provides some protection against cross-site request forgery attacks.

5. Expire and Max-Age Attribute- session management mechanisms based on cookies can make use of two types of cookies, non-persistent (or session) cookies, and persistent cookies. If a cookie presents the `Max-Age` (that has preference over `Expires`) or `Expires` attributes, it will be considered a persistent cookie and will be stored on disk by the web browser based until the expiration time.

6. Session Expiration - In order to minimize the time period an attacker can launch attacks over active sessions and hijack them, it is mandatory to set expiration timeouts for every session, establishing the amount of time a session will remain active

Vulnerable Dependencies – Integreated safety check in build to check for vulnerable packages and frameworks



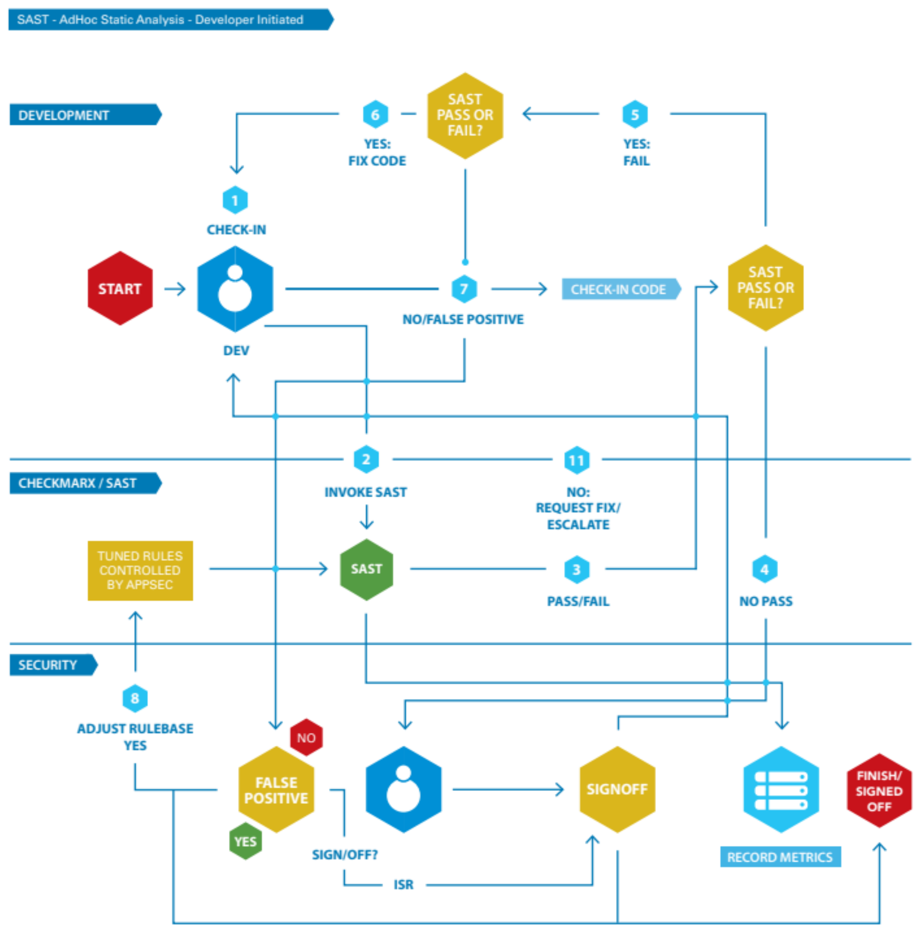
```
(venv) C:\Users\hp-pc\Desktop\new flask_app>safety check
+==============================================================================+
|                                                                              |
|                               /$$$$$$            /$$                         |
|                              /$$__  $$          | $$                         |
|           /$$$$$$$  /$$$$$$ | $$  \__//$$$$$$  /$$$$$$   /$$   /$$           |
|          /$$_____/ |____  $$| $$$$   /$$__  $$|_  $$_/  | $$  | $$           |
|         |  $$$$$$   /$$$$$$$| $$_/  | $$$$$$$$  | $$    | $$  | $$           |
|          \____  $$ /$$__  $$| $$    | $$_____/  | $$ /$$| $$  | $$           |
|          /$$$$$$$/|  $$$$$$$| $$    |  $$$$$$$  |  $$$$/|  $$$$$$$           |
|         |_______/  \_______/|__/     \_______/   \___/   \____  $$           |
|                                                          /$$  | $$           |
|                                                         |  $$$$$$/           |
|  by pyup.io                                             \______/            |
|                                                                              |
+==============================================================================+
| REPORT                                                                       |
| checked 32 packages, using default DB                                        |
+==============================================================================+
| No known security vulnerabilities found.                                     |
+==============================================================================+

(venv) C:\Users\hp-pc\Desktop\new flask_app>
```

DYNAMIC TESTING

1. **Session Fixation –**
2. **Cross Site Scripting**
3. **Remote code execution**

TOOLING

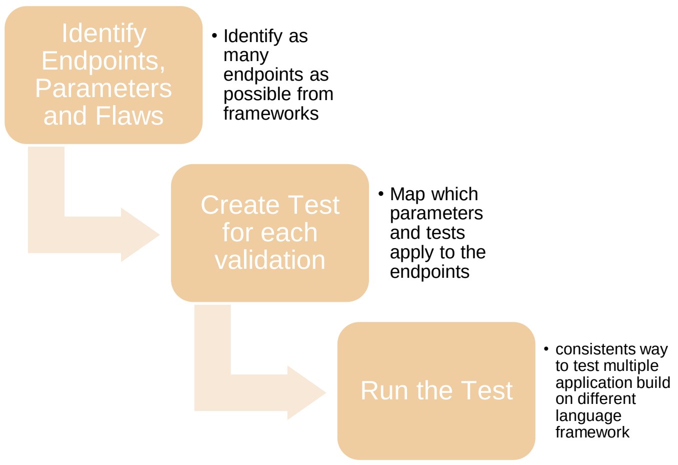
- **Source Code Review –**
    - bandit



- **Composition Analysis –**

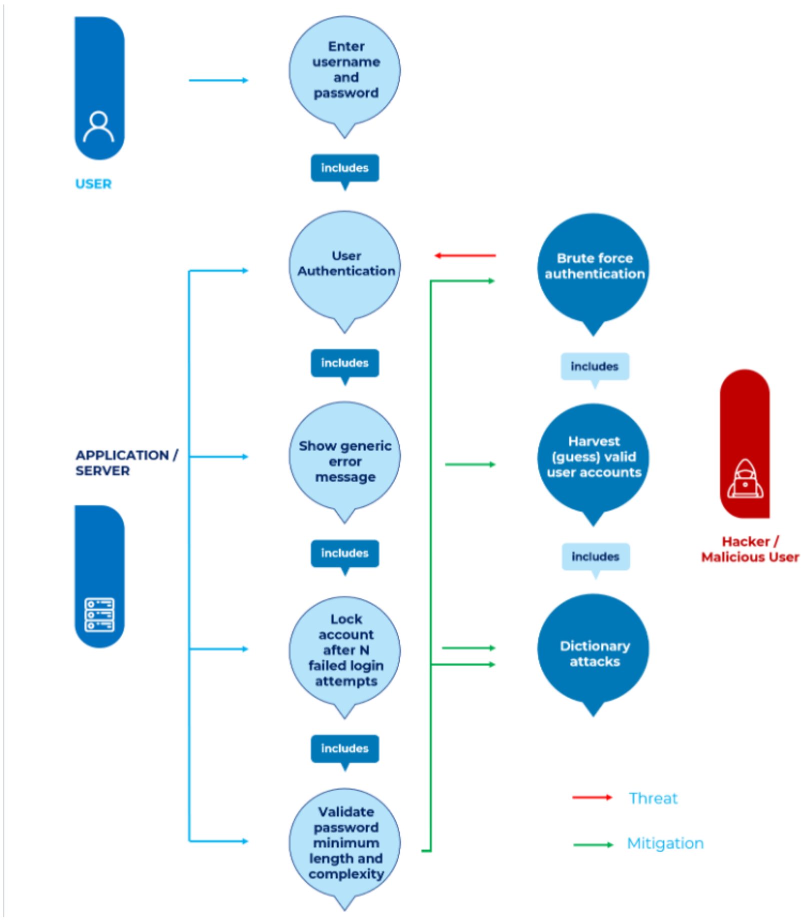
- ○ Dependency checker(Build) /safety check

- **Security Unit Test Cases**


Identify Endpoints, Parameters and Flaws
• Identify as many endpoints as possible from frameworks
Create Test for each validation
• Map which parameters and tests apply to the endpoints
Run the Test
• consistents way to test multiple application build on different language framework

- Security Test Requirements Through Use and Misuse Cases
- Security Testing in the Test Workflow
- Security Testing in the Coding Phase:
- Unit Tests:
  Identity, authentication & access control
  Input validation & encoding
  Encryption
  User and session management
  Error and exception handling
  Auditing and logging

- **Functional Testers Security Tests**


USER
Enter username and password
includes
User Authentication
includes
Show generic error message
includes
Lock account after N failed login attempts
includes
Validate password minimum length and complexity
APPLICATION / SERVER
Brute force authentication
includes
Harvest (guess) valid user accounts
includes
Dictionary attacks
Hacker / Malicious User
Threat
Mitigation

Authentication Flow

Security unit test case to check security headers

AUTOMATION

PREVENTION				
Security Exercise				
Description	**Category**	**SAST**	**DAST**	**Automation**
Session Fixation Attack	Session Management	-	✓	Security Unit Test Case
Missing Input Validation	Input Validation	✓		Security Unit Test Case
Vulnerable dependencies	Dependency Management	✓		Dependency Check
Missing Security Headers	Security Misconfiguration		✓	Security Unit Test Case

Lack of rate limiting	Broken Authentication		✓	Security/Performance Test
Portability Flaw: Locale Dependent Comparison	Security Misconfiguration	✓		SAST
Lack of CSRF protection	CSRF	✓	✓	Security Unit Test Case
Lack Username/Password policies	Broke Authentication	✓	✓	Security Unit Test Case
Cookies attributes	Security Misconfiguration	✓	✓	Security Unit Test Case
Sensitive information disclosure – Password in response	Broken Authentication	✓	✓	Security Unit Test Case

Container Security

Description	**Tools**	**Description**
Image scanning	Snyk, Clair, Trivy, Twistlock	Scan docker images for potential security vulnerabilities
Container run time	sysdig-falco, Twistlock	Monitor container for suspicious behavior
Compliance	Docker-bench, snyk, docker-hunt	Check compliance against CIS benchmkarks
Secrets Management	Hashi Vault, secrets, Vaults	Secrets storage and usage to avoid exposure of sensitive information