**Logo, company name

Description automatically generated**

Template based on the Centers for Medicare & Medicaid Services, Information Security & Privacy Management’s Assessment

**Security Assessment Report**

Version N.0

January 1, 1900

Table of Contents

[1. Summary 3](#_Toc127779445)

[1. Assessment Scope 3](#_Toc127779446)

[2. Summary of Findings 3](#_Toc127779447)

[3. Summary of Recommendations 4](#_Toc127779448)

[2. Goals, Findings, and Recommendations 4](#_Toc127779449)

[1. Assessment Goals 4](#_Toc127779450)

[2. Detailed Findings 5](#_Toc127779451)

[3. Recommendations 5](#_Toc127779452)

[3. Methodology for the Security Control Assessment 5](#_Toc127779453)

[4. Figures and Code 7](#_Toc127779454)

[4.1.1 Process flow of System (this one just describes the process for requesting) 7](#_Toc127779455)

[4.1.2 Other figure of code 7](#_Toc127779456)

[5. Works Cited 7](#_Toc127779457)

# Summary

Executive Summary Here: Describe the overall goal, method, and major findings/recommendations here. (it’s the TLDR)

## Assessment Scope

What tools, platforms, OSes, Browsers, and software (including your own) was tested or used in testing?

Tools: Visual Studio Code, PyCharm, Jupyter Notebook

Platforms: Windows 10, Ubuntu 20.04

Operating Systems: Windows 10, Ubuntu 20.04

Browsers: Google Chrome, Mozilla Firefox, Safari

Software: Python 3.8, TensorFlow, Scikit-learn, NumPy, Pandas, Flask, SQLAlchemy, MySQL

## Summary of Findings

Chart, bar chart

Description automatically generated

Diagram

Description automatically generated

Figure 1. Findings by Risk Level

Explain above and link to full table of explanation of top risks like Figure 3.

Figure 2. SWOT

Explain which issues were used from above SWOT (which are addressed in this assessment).

A big problem we had and still have for the project is database security to protect against SQL injections.

## Summary of Recommendations

# Goals, Findings, and Recommendations

## Assessment Goals

The purpose of this assessment was to do the following:

* Ensure that the system was in compliance with regulations you had to deal with or any other requirements (to include the assignments themselves).
* Determine if the application was securely maintained.
* Etc.

## Detailed Findings

Ensure each vulnerability is thoroughly explained, specific risks to the continued operations are identified, and the impact of each Threat or Weakness is analyzed as a business case. Ensure these are linked to Table 1 when describing the Risk Value. This is not the fixes – it’s the description of the problems found. The fixes go in the next section (for ease of lookup using TOC) - build this off your checklist, SWOT, and risk assessments.

Injection attacks: Injection attacks occur when an attacker injects malicious code into a system through user inputs. This can occur when the input is not properly validated, allowing the attacker to run commands on the system. In a sports handicapping AI system, this vulnerability can be exploited to inject false data or predictions into the system, compromising its accuracy and reliability. (“Injection Attack Types”)

Cross-site scripting (XSS): XSS attacks occur when an attacker injects malicious code into a website or application, which then executes on the user's browser. This vulnerability can be exploited to steal user data, such as login credentials or personal information. In my project system, an XSS attack could be used to gain unauthorized access to the system, steal data, or tamper with predictions. (“Cross Site Scripting (XSS)”)

Cross-site request forgery (CSRF): CSRF attacks occur when an attacker tricks a user into making an unintended action on a website or application. This vulnerability can be exploited to perform actions on the user's behalf, such as transferring funds or making purchases. In a sports handicapping AI system, a CSRF attack could be used to alter predictions, input false data, and some other malicious things.

Man-in-the-middle (MitM) attacks: MitM attacks occur when an attacker intercepts communications between two parties, allowing them to eavesdrop on the conversation, modify data, or inject malicious code. This vulnerability can be exploited to steal user data, gain access to the system, or tamper with predictions in a sports handicapping AI system.

Password attacks: Password attacks occur when an attacker tries to guess or crack a user's passwords. This vulnerability can be exploited to gain unauthorized access to the system, steal users data, or alter predictions in a my system.

Insufficient authentication and authorization: Insufficient authentication and authorization occurs when a system does not properly verify user identities or permissions. This vulnerability can be exploited to gain unauthorized access to the system, steal data, or manipulate predictions in a sports handicapping AI system.

Insecure storage: Insecure storage occurs when sensitive data is not properly secured, such as by using weak encryption or storing data in plain text. This vulnerability can be exploited to steal sensitive data, such as user credentials or predictions, in a sports handicapping AI system.

Lack of input validation: Lack of input validation occurs when a system does not properly check and sanitize user inputs, leaving the system vulnerable to injection attacks and other vulnerabilities. This vulnerability can be exploited to inject false data, steal sensitive data, or alter predictions in a sports handicapping AI system.

## Recommendations

Here’s where your fixes go (ensure you reference Table 2 for your ease of fix evaluation and explain why it matches that category).

* For authentication we can use captcha and other manual authentication processes.
* Developers should follow secure coding practices to limit chances for a hack
* Make sure to keep SQL databases as secure as possible
* Sensitive data, such as user credentials and predictions, should be properly encrypted and hashed to prevent unauthorized access and theft.
* Regular software updates should be performed to ensure that the system is protected against known vulnerabilities

# Methodology for the Security Control Assessment

**3.1.1 Risk Level Assessment (delete this text: you don’t have to change 3.1.1)**

Each Business Risk has been assigned a Risk Level value of High, Moderate, or Low. The rating is, in actuality, an assessment of the priority with which each Business Risk will be viewed. The definitions in Table 1 apply to risk level assessment values (based on probability and severity of risk). While Table 2 describes the estimation values used for a risk’s “ease-of-fix”.

Table - Risk Values

| Rating | Definition of Risk Rating |
| --- | --- |
| High Risk | Exploitation of the technical or procedural vulnerability will cause substantial harm to the business processes. Significant political, financial, and legal damage is likely to result |
| Moderate Risk | Exploitation of the technical or procedural vulnerability will significantly impact the confidentiality, integrity and/or availability of the system, or data. Exploitation of the vulnerability may cause moderate financial loss or public embarrassment to organization. |
| Low Risk | Exploitation of the technical or procedural vulnerability will cause minimal impact to operations. The confidentiality, integrity and availability of sensitive information are not at risk of compromise. Exploitation of the vulnerability may cause slight financial loss or public embarrassment |
| Informational | An “Informational” finding, is a risk that has been identified during this assessment which is reassigned to another Major Application (MA) or General Support System (GSS). As these already exist or are handled by a different department, the informational finding will simply be noted as it is not the responsibility of this group to create a Corrective Action Plan. |
| Observations | An observation risk will need to be “watched” as it may arise as a result of various changes raising it to a higher risk category. However, until and unless the change happens it remains a low risk. |

Table - Ease of Fix Definitions

| Rating | Definition of Risk Rating |
| --- | --- |
| Easy | The corrective action(s) can be completed quickly with minimal resources, and without causing disruption to the system or data |
| Moderately Difficult | Remediation efforts will likely cause a noticeable service disruption   * A vendor patch or major configuration change may be required to close the vulnerability * An upgrade to a different version of the software may be required to address the impact severity * The system may require a reconfiguration to mitigate the threat exposure * Corrective action may require construction or significant alterations to the manner in which business is undertaken |
| Very Difficult | The high risk of substantial service disruption makes it impractical to complete the corrective action for mission critical systems without careful scheduling   * An obscure, hard-to-find vendor patch may be required to close the vulnerability * Significant, time-consuming configuration changes may be required to address the threat exposure or impact severity * Corrective action requires major construction or redesign of an entire business process |
| No Known Fix | No known solution to the problem currently exists. The Risk may require the Business Owner to:   * Discontinue use of the software or protocol * Isolate the information system within the enterprise, thereby eliminating reliance on the system   In some cases, the vulnerability is due to a design-level flaw that cannot be resolved through the application of vendor patches or the reconfiguration of the system. If the system is critical and must be used to support on-going business functions, no less than quarterly monitoring shall be conducted by the Business Owner, and reviewed by IS Management, to validate that security incidents have not occurred |

**3.1.2 Tests and Analyses**

* This was completed using

Planning: The first step is to plan the penetration testing process. This involves identifying the scope of the testing, the objectives, and the potential risks that need to be assessed.

Reconnaissance: This step involves gathering information about the system, such as network topology, system architecture, and application framework. This can be done using various tools such as port scanners, network mappers, and web crawlers.

Vulnerability Scanning: This step involves scanning the system for known vulnerabilities using automated tools such as Nessus, OpenVAS, or Qualys.

Exploitation: This step involves attempting to exploit the vulnerabilities found during the vulnerability scanning phase. This can be done manually or using automated tools such as Metasploit.

Post-exploitation: Once access to the system has been gained, this step involves identifying additional vulnerabilities, escalating privileges, and gathering sensitive information.

Reporting: The final step involves compiling a detailed report of the findings, including the vulnerabilities discovered, their severity, and recommendations for remediation.

**3.1.3 Tools**

This was completed using Burp Suite: A web application testing tool that can be used to identify and exploit web application vulnerabilities.

Metasploit: A penetration testing tool that can be used to exploit vulnerabilities in a system. https://tryhackme.com/room/metasploitintro

Nessus: A vulnerability scanning tool that can be used to identify known vulnerabilities in a system.

Nmap: A port scanning tool that can be used to identify open ports on a system. nmap -p- <target IP>

Wireshark: A network analysis tool that can be used to capture and analyze network traffic. wireshark -i eth0 https://tryhackme.com/module/wireshark

# Figures and Code

Insert any pictures here (including of major code issues or code that was used as a tool – can just screenshot and add link to github). This section must include at least 4 figures or code portions:

To authennicate using captcha in python:

from random import randint

from PIL import Image, ImageDraw, ImageFont

# Generate a random captcha string

def generate\_captcha\_string():

captcha\_chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

captcha\_length = 6

captcha\_string = ''.join([captcha\_chars[randint(0, len(captcha\_chars)-1)] for \_ in range(captcha\_length)])

return captcha\_string

# Generate a captcha image

def generate\_captcha\_image(captcha\_string):

captcha\_font = ImageFont.truetype('arial.ttf', 36)

captcha\_image = Image.new('RGB', (200, 100), (255, 255, 255))

captcha\_draw = ImageDraw.Draw(captcha\_image)

captcha\_draw.text((10, 40), captcha\_string, font=captcha\_font, fill=(0, 0, 0))

# Add some noise to the image

for \_ in range(100):

captcha\_draw.point((randint(0, 200), randint(0, 100)), fill=(randint(0, 255), randint(0, 255), randint(0, 255)))

return captcha\_image

# Verify the user's input against the captcha string

def verify\_captcha(captcha\_string, user\_input):

return captcha\_string == user\_input

# Example usage

captcha\_string = generate\_captcha\_string()

captcha\_image = generate\_captcha\_image(captcha\_string)

captcha\_image.show()

user\_input = input('Enter the captcha string: ')

if verify\_captcha(captcha\_string, user\_input):

print('Captcha authentication successful!')

else:

print('Captcha authentication failed.')

Securing SQL database in Python sqllite using hashing:

import sqlite3

from hashlib import pbkdf2\_hmac

import os

# function to create a salted hash of a password

def hash\_password(password):

salt = os.urandom(32)

key = pbkdf2\_hmac('sha256', password.encode('utf-8'), salt, 100000)

return salt + key

# function to verify if a password matches its hash

def verify\_password(password, hash):

salt = hash[:32]

key\_to\_check = pbkdf2\_hmac('sha256', password.encode('utf-8'), salt, 100000)

return key\_to\_check == hash[32:]

# connect to the database

conn = sqlite3.connect('mydatabase.db')

# create a table for users

conn.execute('''CREATE TABLE users

(id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT NOT NULL,

password TEXT NOT NULL);''')

# add a user to the table

password = 'my\_password'

hashed\_password = hash\_password(password)

conn.execute("INSERT INTO users (username, password) VALUES (?, ?)", ('my\_username', hashed\_password))

conn.commit()

# verify the user's password

stored\_password = conn.execute("SELECT password FROM users WHERE username = 'my\_username'").fetchone()[0]

print(verify\_password(password, stored\_password)) # prints True if the password is correct, False otherwise

# close the connection to the database

conn.close()

Protect against Man-in-the-middle attack in python:

import socket

import ssl

# create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# wrap the socket with SSL/TLS encryption

context = ssl.create\_default\_context()

s = context.wrap\_socket(s, server\_hostname='MyHandicapper.ai')

# connect to the server

s.connect(('MyHandicapper.ai', 443))

# send data to the server

data = 'Hello, server!'

s.send(data.encode())

# receive data from the server

received\_data = s.recv(1024)

print(received\_data.decode())

# close the connection

s.close()

Make a secure login in python using tkinter

from tkinter import \*

root = Tk()

root.title("Secure Login")

label = Label(root, text="Please enter your login credentials:")

label.pack()

username\_label = Label(root, text="Username:")

username\_label.pack()

username\_entry = Entry(root)

username\_entry.pack()

password\_label = Label(root, text="Password:")

password\_label.pack()

password\_entry = Entry(root, show="\*")

password\_entry.pack()

def validate\_login():

username = username\_entry.get()

password = password\_entry.get()

# Check if the username and password match with saved credentials

if username == "my\_username" and password == "my\_password":

# Login successful, display success message

success\_label = Label(root, text="Login successful!")

success\_label.pack()

else:

# Login failed, display error message

error\_label = Label(root, text="Invalid login credentials.")

error\_label.pack()

# Create a Button widget to submit the login credentials

submit\_button = Button(root, text="Login", command=validate\_login)

submit\_button.pack()

# Run the main loop

root.mainloop()

### Process or Data flow of System (this one just describes the process for requesting), use-cases, security checklist, graphs, etc.

Diagram

Description automatically generated

Describe the process flow here.

### Other figure of code

HERE

# Works Cited

“Cross Site Scripting (XSS).” *OWASP Foundation*, https://owasp.org/www-community/attacks/xss/. Accessed 1 May 2023.

“Injection Attack Types.” *Contrast Security*, [https://www.contrastsecurity.com/glossary/injection-attack-types. Accessed 1 May 2023](https://www.contrastsecurity.com/glossary/injection-attack-types.%20Accessed%201%20May%202023).

“Burp Suite - Application Security Testing Software.” *PortSwigger*, https://portswigger.net/burp. Accessed 1 May 2023.

<https://fgcu.instructure.com/courses/538132/files?preview=49205290>

NetworkSecurityIntro.pptx

Cypher Encryption lecture

https://tryhackme.com/module/wireshark

https://tryhackme.com/room/metasploitintro