CYBERSECURITY REPORT

(New Features Report)

1.0 Introduction

This report gives a thorough analysis of the additional features and components that were recently added to the existing website. The scope of this report is to comprehend the features and components, potential security threats, risk assessment, mitigation solutions, and estimated prevention and mitigation costs. Added to this report is a code script of a basic malware-detection program.

2.0 Overview of New Feature

The key added feature in this project is the option for students to submit additional files to be evaluated by their tutors. This feature consists of multiple components, including newly redesigned submission tabs, a diff-display, version control, and new upload buttons.

2.1 Implementation of New Feature with APIs (Application Programming Interface)

In order to apply this new feature and components, the implementation of specific types of API is required. One of the main APIs involved is the Git API.

Git API

Use of the Git API enables version control and diff-display functionality. This permits interaction between the system and Git repositories, branches, and commits. Thus, enabling efficacious file comparisons and exhibiting the differences between various versions.

Apart from the Git API, there are a few other APIs involved. One of them is associated with the file upload feature, which allows the system to receive and process the additional uploaded files. Another API is related to the backend server which permits communication between the frontend and backend systems, storage of necessary information in the database, and interaction with the Git API.

3.0 Security

The incorporation of additional features could impact the security of the existing website. There are certain security concerns that should be considered.

3.1 Data Breach

The exposing of student data such as submitted files, grades, and personal information, is a result of weak security measures that could cause a breach. Strong encryption techniques, secure data storage and frequent security audits are particularly essential to prevent such breaches.

3.2 Unauthorised Access

A lack of authorisation and authentication tools might enable unauthorised access to the system, thereby jeopardising sensitive data such as confidential student information, submitted files and backend infrastructure. It is important to implement secure authentication practices, including multi-factor authentication.

3.3 Injection Attacks

As a result of APIs related to the new feature, the system is vulnerable to injection attacks. Attackers could take advantage of this feature and upload malicious files, compromise the system or execute arbitrary code, putting the system at a huge risk. These risks can be mitigated by incorporating not only a strict file type validation but also file size limits and secure coding practices.

3.4 Malicious Uploads

Students may unintentionally submit files containing malware or viruses if sufficient security checks are not carried out. This risk can be mitigated by using an antivirus software, file-scanning procedures and educating students on relevant information to practice safe file uploading.

3.5 Insecure Data Storage

In order to prevent unauthorised access and data leakage, stored data much be encrypted, access controls should be enforced, and secure storage mechanisms must be implemented.

3.6 API Vulnerabilities

The APIs might have vulnerabilities that could be taken advantage of by attackers. Insecure data transfer, weak authentication, and a lack of input validation are common vulnerabilities within the APIs themselves. Frequent security testing and secure coding practices are crucial to limit these potential risks.

4.0 Risk Assessment

An in-depth analysis should be conducted to assess the risks that come with the new features and components. The table below demonstrates the risk assessment on the vulnerabilities and threats surrounding the additional feature and components.

THREATS	LIKELIHOOD	IMPACT	RISK LEVEL
Data Breach	Medium	High	High
Unauthorised Access	Medium	High	High
Injection Attacks	Low	Medium	Medium
Malicious Uploads	Medium	Medium	Medium
Insecure Data Storage	Low	Medium	Low-Medium
API Vulnerabilities	Medium	Medium	Medium

4.1 Impacts

Data Breach: Significant impact.

Reputational damage, loss of trust from users (students) and regulatory penalties.

Unauthorised Access: Severe impact.

Potential data manipulation, legal consequences and leaked sensitive student information.

Injection Attacks: Moderate impact.

Data corruption, system compromise and service disruption.

Malicious Uploads: Moderate impact.

Potential system disruption, compromised data integrity, and loss of data.

Insecure Data Storage: Low-Moderate impact.

Unauthorised access, data exposure, potential data loss, reputational damage and legal consequences.

API Vulnerabilities: Moderate impact.

Data manipulation, unauthorised access and attacks towards the system.

5.0 Mitigation

In order to minimise the risks distinguished above, it is recommended that the following strategies be adopted:

5.1 Data Encryption and Secure Storage

- Ensure that sensitive data is encrypted at rest and in transit to prevent unauthorised access.

- Securely store submitted files and student data using encrypted databases and file systems.

- Maintain regular data backups and rest restoration processes to minimise data loss risks.

5.2 Regular Security Checks

- Identify and address potential security weaknesses through regular penetration testing and vulnerability assessments.

- Review and audit code for secure coding practices.

- Monitor security threats and vulnerabilities related to the APIs and apply patches and updates as soon as possible.

5.3 Secure Authentication and Authorisation:

- Implement multi-factor authentication mechanisms to ensure strong authentication.

- Implement and maintain strong password policies that include password complexity requirements and regular password changes.

- Implement strong password policies, such as requiring a complex password and changing it regularly.

- Ensure that only authorised users can access sensitive functions and data by applying role-based access controls.

5.4 Secure File Uploads

- Ensure that file types and sizes are validated, and size limits are enforced so that malicious or oversized files cannot be uploaded.

- Detect and prevent files containing malware or viruses by using antivirus scanning mechanisms.

- Prevent code execution vulnerabilities and mitigate the risk of injection attacks by sanitising user input.

5.5 API Security

- In order to address known vulnerabilities, APIs should be regularly updated and patched.

- Prevent common API security issues by implementing secure coding practices, such as input validation and output encoding.

- Monitor API traffic with API gateways or firewalls, detecting malicious requests, and blocking them.

6.0 Estimated Costs

There are several factors that will determine the cost of prevention and mitigation measures, including the organisation's size, resources and the chosen security solutions. Some of the possible costs include:

- Infrastructural investments in. secure systems, including encrypted storage and secure servers.

- The installation of security tools such as firewalls and antivirus scanners.

- Conducting regular vulnerability or incident assessments and responding to incidents with the help of cybersecurity professionals.

- Security protocols, password policies and secure coding practices and trainings for staffs.

- Legal fees and regulatory penalties related to potential data breaches.

As a matter of fact, it is important to note that implementing preventive measures can result in substantially lower costs than resolving security incidents. It is important to take proactive security measures as a result of the potential financial and reputational consequences of a data breach or cyberattack.

7.0 Conclusion

In addition to providing additional flexibility to students and tutors, the new feature also increases the potential for security vulnerabilities and threats. By conducting a thorough risk assessment and implementing mitigation strategies, the new feature, components and APIs can be reduced in terms of risk. An effective cybersecurity strategy includes securing authentication and authorisation, encrypting data and storing it securely, uploading files securely, implementing API security and conducting regular security assessments.

As a final point, it is important to note that cybersecurity is an ongoing process. Frequent monitoring, updates and testing are crucial to ensure that security measures continue to function effectively as well as to address new and emerging threats. A company's reputation can be safeguarded, and important information can be protected by prioritising cybersecurity.

8.0 Code Script (Malware-detection program)

Attached below is a basic code script of a malware-detection program that can be worked with to ensure that the additional files uploaded by students do not contain any suspicious or malicious content. The code conducts a few checks to ensure the security of the files. The first check is checking if a file is an executable file with the ".exe" extension. If that is not the case, the code checks if the file is an executable file disguised as a different file type. If this does not occur to be the case, the code checks if the file contains or is very similar to any viruses that are recognised in a specific online database. Finally, the code checks the text in the document to ensure that there are no suspicious patterns that could be malicious.

Code:

```
# To work with file paths

from pathlib import Path



# To import hashlib

import hashlib



# This is to work with APIs (e.g. VirusTotal and Google Malicious URL databases)

import requests



# To work with PDF files, e.g. extract text

import PyPDF2



# To find patterns in text

import re



path = "Downloads/some_submission.py"
```

```
is_malicious = do_all_checks(submission_path=path)



def do_all_checks(submission_path):
    """
    Step 1:Check if the file itself could be malicious
        Step 1a: Check if the path has a suspicious extension (e.g. '.exe')
        Step 1b: Check if the file is executable
        Step 1c: Check if the file is a known virus (compare with a Virus Database API)

    Step 2: Check if the content of the text has malicious code, or links (URLs)
        Step 2a: Extract text from a PDF file
        Step 2b: Check if the text has malicious code
        Step 2c: Extract all the URLs in the PDF File
        Step 2d: Chekc if the urls are known to be malicious (compare with a Malicious URL Database)

    """
    # Step 1:Check if the file itself could be malicious
    step_1a = is_suspicious_file_extension(submission_path)
    step_1b = is_file_executable(submission_path)
    step_1c = is_file_a_known_virus(submission_path)

    # Assumes submission is a PDF
    # Step 2a
    text = extract_text_from_pdf(submission_path)
    # Step 2b:
    step_2b = find_malicious_code_in_text(text)
```

```
    if step_1a or step_1b or step_1c or step_2b:
        is_malicious = True
    else:
        is_malicious = False

    return is_malicious


def is_suspicious_file_extension(submission_path):
    """Checks if the path has a suspicious extension"""
    suspicious_extensions = [
        ".bat",
        ".com",
        ".vbs",
        ".vbe",
        ".js",
        ".jse",
        ".scr",
        ".hta",
        ".reg",
        ".lnk",
        ".exe",
        ".dll",
        ".cmd",
    ]
    extension = Path(submission_path).suffix
```

```
    if extension.lower() in suspicious_extensions:
        return True
    else:
        return False


def is_file_executable(submission_path):
    """Opens the file in binary mode and reads the first two bytes.
    The logic checks if the first two bytes are MZ. The first two bytes of an executable
    file are always MZ.
    """
    with open(submission_path, "rb") as file:
        # Read the first 2 bytes of the file
        first_two_bytes = file.read(2)

    # Check if the first 2 bytes correspond executable files

    if first_two_bytes == b"MZ":
        return True
    else:
        return False


def is_file_a_known_virus(submission_path):
    """Checks if the file's signature is in a well known database (VirusTotal.com).
    Checks if the file's hash is known to be malicious by querying VirusTotal.
    """
```

```
    # Get the file's hash.
    with open(submission_path, "rb") as f:
        file_hash = hashlib.sha256(f.read()).hexdigest()

    virus_db_url_query = (
        "https://www.virustotal.com/vtapi/v2/file/report?resource={}".format(file_hash)
    )
    # Check if the file's hash is known to be malicious.
    response = requests.get(virus_db_url_query)
    if response.status_code == 200:
        for result in response.json()["scans"]:
            if result["result"] == "malicious":
                return True

    return False


def extract_text_from_pdf(submission_path):
    """Extact Text from a pdf file"""
    with open(submission_path, "rb") as file:
        pdf = PyPDF2.PdfFileReader(file)
        text = ""
        for page_num in range(pdf.getNumPages()):
            page = pdf.getPage(page_num)
            text += page.extractText()
    return text
```

```
def find_malicious_code_in_text(text):
    # Define suspicious patterns
    python_patterns = [
        r"eval\(",  # eval function
        r"exec\(",  # exec function
        r"os\.system\(",  # os.system function
        r"subprocess\.",  # subprocess module
        r"import\s+os",  # os module import
        r"import\s+pickle",  # pickle module import
    ]

    javascript_patterns = [
        r"<script>",  # Opening script tag
        r"document\.cookie",  # Accessing cookies
        r"Function\(",  # Function constructor
        r"execScript\(",  # execScript function
        r"setTimeout\(",  # setTimeout function
        r"document\.write",  # Writing to the HTML document
    ]

    sql_patterns = [
        r"(\s+|;)(drop|truncate|delete|update|insert)\s+table",  # SQL destructive actions
        r"(\s+|;)exec(\s|\()",  # SQL EXEC command
        r"(\s+|;)xp_cmdshell",  # SQL Server xp_cmdshell function
        r"(\s+|;)sleep\s*\(",  # SLEEP function for time-based blind SQL injection
    ]
```

```
all_patterns = python_patterns + javascript_patterns + sql_patterns



for pattern in all_patterns:

    if re.search(pattern, text):

        return True

return False
```

9.0 References

Australian Government 2021, *Protect Your Business from Cyber Threats*, Australia Business, Accessed 10 May 2023.

https://business.gov.au/online/cyber-security/protect-your-business-from-cyber-threats.

Federal Trade Commission (2019), *Data Breach Response: A Guide for Business*, Federal Trade Commission, Accessed 09 May 2023.

https://www.ftc.gov/business-guidance/resources/data-breach-response-guide-business.

IBM Security Network Intrusion Prevention System (2013), *Injection Attacks*, IBM, Accessed 17 May 2023. https://www.ibm.com/docs/zh/snips/4.6.0?topic=categories-injection-attacks.

IBM (2022), *How Risk Assessment Scores Are Calculated*, IBM, Accessed 17 May 2023. https://www.ibm.com/docs/en/elms/elm/6.0.6?topic=risk-how-assessment-scores-are-calculated.

JP Morgan (2022), *12 Tips for Mitigating Cyber Risk*, JP Morgan Chase, Accessed 08 May 2023. https://www.jpmorgan.com/commercial-banking/insights/12-tips-for-mitigating-cyber-risk.

Kapil, B (2023), *Introduction to Github Api's*, Login Radius, Accessed 10 May 2023. https://www.loginradius.com/blog/engineering/github-api/.

Nyakundi, H (2023), *5 Ways to Improve Your Web Application and API Security*, The New Stack, Accessed 17 May 2023.

https://thenewstack.io/5-ways-to-improve-your-web-application-and-api-security.

Sotnikov, I (2018), *How to Perform IT Risk Assessment*, Netwrix, Accessed 11 May 2023. https://blog.netwrix.com/2018/01/16/how-to-perform-it-risk-assessment/.

Sukianto, A (2022), *10 Ways to Reduce Cybersecurity Risk for Your Organization*, Upguard, Accessed 14 May 2023.

https://www.upguard.com/blog/reduce-cybersecurity-risk.

Tamara, J (2022) , *Website Maintenance Cost In 2023 And Hiring A Professional Vs Self-Maintenance*, Hostinger Tutorials, Accessed 12 May 2023, https://www.hostinger.com/tutorials/website-maintenance-cost#:~:text=SSL%20certificate%20costs%20may%20vary.

Tunggal, AT (2019), *How to Perform an IT Cyber Security Risk Assessment: Step-By-Step Guide*, Up-guard, Accessed 11 May 2023.

https://www.upguard.com/blog/cyber-security-risk-assessment.

Virtu (2023), *6 Strategies for Cybersecurity Risk Mitigation*, Virtu IT Solutions, Accessed 07 May 2023. https://virtu.net/6-strategies-for-cybersecurity-risk-mitigation/.