北京郵電大學 Beijing University of Posts and Telecommunications

Queen Mary University of London

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Undergraduate Project Report
2021/22

Improving Security of Web Applications Based on Mainstream Technology

Name:	Song Linxuan
School:	International School
Class:	2018215119
QMUL Student No.:	190018720
BUPT Student No.:	2018213147
Programme:	Internet of Things Engineering
Supervisor:	Marisol García Valls Yan Sun

Date: 28-04-2022

Table of Contents

Abstract

This project mainly studies mainstream web application technologies and their related security flaws. Also, a web application is developed in this project to help the user detect the vulnerabilities and get recommendations. A simple Internet of Things (IoT) system is simulated to make the application more practical. This report first introduces the technologies studied and used and the vulnerabilities studied and focused in this project. Then, the web application is introduced together with how it is designed and implemented, what vulnerabilities is supported by it, how the vulnerabilities can be tested and how the users and visitors can use it. Also, the performance data of the application with the methods and APIs used to get these data, and metrics related to the security check report are introduced in this report.

Key Words: Web Security, Web Application, Web Application Vulnerabilities, Web Application Development, Internet of Things.

摘要

该项目致力于研究主流 web 技术及其相关安全漏洞。此外，该项目还开发了相应的 web 应用以帮助用户测试安全漏洞并向用户提供相关修复建议。本项目还模拟了一个简单的物联网系统以使应用更加接近实际生活场景。本篇报告首先介绍了该项目所学习研究的 web 应用技术和与之相关的一些重点关注的安全漏洞。之后，本文介绍了该项目所开发的 web 应用的设计和实现原理、所支持的安全漏洞种类、测试方式以及用户和游客应如何使用此应用。接下来，本文介绍了该应用的性能测试数据以及测试所用的方法和 API 接口，以及安全漏洞测试报告中数据生成的指标和方式。

关键词：网络安全，web 应用，web 开发，web 安全漏洞，物联网

Chapter 1: Introduction

1.1 Context and Motivation of the Project

Security is one of the most important things for web applications, especially with the rapid development and evolution in the field of information technology and computer science. Today, more and more companies and organizations start to migrate their services to web applications and as a result, it becomes vital important to keep the security of these applications. In fact, many international organisations and standards such as International Organization for Standardization (ISO) 27001: 2013 have listed the requirements to implement and improve the security settings for web applications 41[1]. Also, the Open Web Application Security Project (OWASP) has been publishing the top 10 most critical security risks regularly for the companies and developers to raise the awareness of web application security and try to reduce the damage caused by the vulnerabilities in the applications. Based on these standards and studies, this project aims to study the state of art web technologies and their related security flaws and develop a web application to help the users learn more about the vulnerabilities in their application, find them and offer solutions and recommendations for the users. Furthermore, we will also study the recommended programming patterns and try to help the users to repair the unsafe configurations and minimize the risks.

1.2 Objectives of the Project

To finish the project, 4 main objectives will be achieved:

First is to learn the state of art web programming technologies and their most common configuration issues regarding security, which includes client-side and server-side technologies. The client-side technologies to be studied and analysed are HTML5, Cascading Style Sheets (CSS) and JavaScript while for the server side, the study of Python Flask will be the majority of this project, with lesser content on Node.js.

Second is to discover main flaws of current web application deployments regarding security. To finish this, some selected vulnerabilities based on authoritative publications and rankings will be studied. As mentioned in last paragraph, these vulnerabilities are mainly related to the server-side applications coded for the Flask platform.

Third is to design and develop the application. In this step, a simple Internet of Things (IoT) system will be designed and be used as the baseline to exemplify the vulnerabilities. Also, some third-party codes and APIs will be studied, analysed and reformed to help with the check of

vulnerabilities.

Fourth is to analyse the performance and behaviour of the application, which will be evaluated by the runtime of particular modules and functions. Some JavaScript performance APIs and Python functions will be used here for measurements.

1.3 Methodology

Since this project focuses on software systems development, it is required to follow a proper methodology that follows some of the basic principles to ensure correct design and development. Moreover, the project requires to perform exploratory work on vulnerability search and testing for a particular software platform, which makes it convenient to produce fast prototypes to test some particular functionalities and learn from their outcomes as to take the proper follow up path.

In this context, this section briefly explains the software engineering life cycle that has been followed. Software engineering is the process of applying engineering techniques to the development process of software. These techniques involve a number of phases, being the commonly identified across software engineering methodologies[2]: requirements definition, design, development / implementation, testing, integration, and maintenance.

1.3.1 Agile Software Development

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. It focuses more on the code rather than the analysis or design and is based on an iterative approach to software development. Also, it intends to deliver working software quickly and evolve this quickly to meet changing requirements.

Agile includes a set of practices during software development period, which are iteration, efficient communication, short feedback and quality focus. The iteration, also known as sprint, is to break the whole project into small pieces and each of them should be finished within a short period. An iteration may not have too many functional updates, but to have a release with the least number of bugs. This gives the product sufficient room for failure in the early stage and can ensure the program be finished as on time as possible.

Besides, efficient communication and short feedback are also important during the development. This requires the developers to review how the process went towards the goal in the last period and discuss whether they need to adapt the plans regularly. It can be implemented via different

kinds of meeting, usually sprint meeting and longer meeting. A sprint meeting is an event in scrum that defined what to achieve during the next period of work and how the work can be achieved. In this project, these requirements are realized by meetings with my supervisor. Throughout the project, sprint meetings with the supervisors are held weekly (usually on each Tuesday) to discuss the progress in the past week and what to finish for the next coming week (See Project Supervision Log in Appendix). Sometimes the meeting cycle also goes shorter or longer owing to the workload of corresponding period. We also had some longer meetings when the main objective mentioned in section 1.2 was finished to summarize the project progress and discuss the plan and steps for the upcoming objectives.

1.4 Structure of the Report

The body of this report includes 5 chapters, which are Introduction, Background, Design and Implementation, Results and Discussion and Conclusion and Further Work.

Chapter 1 introduces the motivation and objectives of the project and the methodologies used during the process period. It also has a brief introduction to the content of the whole report.

Chapter 2 is the background of this project. It contains the technologies used to develop the web application for this project and some basic web application vulnerability knowledge used in this project.

Chapter 3 is the design and implementation of the project. It describes how the web application is realized and deployed with a simple IoT system and how it looks like.

Chapter 4 is the results and discussion of the project. This chapter mainly contains the performance and evaluation to the web application.

Chapter 5 is conclusion and further work. It summarizes the whole report and what we have achieved in this project. Besides, it also points out some shortcomings of current applications and the work can be done in the future to overcome them.

Chapter 2: Background

This chapter mainly talks about the background technologies that used in this project, which include the web application and security technologies. Various technologies are used to develop the server and client side of the application and some professional organisations are also introduced to study the security issues related to them.

2.1 Web Application

Traditionally, a web application usually has a C/S structure, which means that there is a client for the user to operate and send requests and a server to process the requests [3]. In early times, a client-side program should be installed in advance by the user to serve as the User Interface (UI) and pre-process the requests from the client. The advantage of this structure is that it can reduce the processing load on the server side. However, if the developer wants to upgrade the application, the client-side program should also be upgraded, which decreases the efficiency. Also, the client-side program should be adapted to different operating systems and computers, which makes the development hard and costly.

With the development of information technology, web applications now can be developed by standard web documents like HyperText Markup Language (HTML) and JavaScript that are supported by nearly all the browsers in the world. With the help of network protocols like HTTP and web services, the client-side program can be loaded automatically by the browser every time when the user accesses the application. The most common examples for modern web applications include online shopping, online forums and blogs.

To develop a web application, different technologies for the client, server side and even the database of the application are required. In this project, the client side is developed by HTML, Cascading Style Sheets (CSS) and JavaScript while for the server side, Node.js and Python Flask are needed. Also, the template engine Jinja2 embedded in Flask is also used to transmit data between the server and client side of the application, which uses curly brackets on the client side to put the data and statements (in the form of {{ parameter }}, {% statement %} and {# comment #}).For the database of the application, data exchange format like JSON and NoSQL database MongoDB are used to store the data.

2.2 Basic Terminology

It is important to understand the differences between the basic but similar terminologies in the field of web security, which are threat, vulnerability and risk. They will be introduced in the

following 3 sections in order of potentiality of danger.

2.2.1 Threat

Basically, threat means any danger possible that can exploit a vulnerability and obtain, damage or destroy an asset intentionally or accidentally. Here "asset" doesn't mean resources in the field of finance, but the information like database, code and software in the information technology or computer science area. Threat is something that may negatively affect the security of the system (program, project, etc.) and exists all the time regardless of whether it will succeed or not in the end. It does not necessarily have to be some errors or bugs that really exists in the system (program, project, etc.) and as a result, we can only try to minimize rather than totally avoid the bad effects caused by it.

The Internet Engineering Task Force (IETF) gives quite a clear description of threat: first, a threat is a potential for security violation. It can be caused by everything that may cause harm, such as an entity, event or action. Second, it is any circumstance or event that may cause negative effect to a system through vulnerabilities like unauthorized access and denial of service. A threat can be intentional or accidental [4]. Just as the words imply, the first condition happens when there is a real individual or organization to execute the attack. This can be avoided by taking actions like improving security policy and fixing bugs. For the second condition, it is usually caused by some objective reasons such as human error, device failure and even some Force Majeure like earthquake. The accidental threat cannot be avoided, but we can still try to reduce the damage caused by it via some means like backup.

2.2.2 Vulnerability

Vulnerability is the weakness or flaw in a system that reduces the security of the system. It is a weakness in the system and currently lack of our protection and is often misused with threat that we mentioned above [4]. The key difference between them is that vulnerability is something that does exist in the system while threat probably does. A vulnerability can exist in any part of the system (for example, hardware, software, network and server) and can be exploited by external force like an attacker, a script and some tools.

A number of organizations and standards have given similar definitions to vulnerability, such as ISO 27005, IETF RFC 4949 and the European Union Agency for Cybersecurity (ENISA). The common sense among all these definitions is that a vulnerability is some shortcomings inside the system and can be exploited by threats, attacks and some forces to violate the security policy and do harm to the assets in the system. A terminal can have one or more vulnerabilities

that can be exploited, which can be caused in different layers and by different reasons. For example, in the software area, the vulnerabilities can be caused by insecure design or insufficient testing, and for network layer, it can be caused by unauthorized connection.

For an application system, the more complex and larger the system is, the higher possibility that it may contains vulnerabilities because larger system can leave more possible bugs and flaws that are vulnerable and can be taken advantage of by the attackers. Also, the use of some external libraries and repetition of same codes can increase the risk level if the attacker has found ways or tools to exploit the vulnerabilities inside them. What's more, for IoT applications, the connection of different devices is also a concern because each node, port and service can be exposed to danger, which also cause a number of vulnerabilities. Besides, other reasons like poor password management, flaws in the Operating System or browser, no filter to user input and lack of maintenance can also lead to more vulnerabilities in the application system.

2.2.3 Risk

This is another item that often misused with the 2 concepts above. A risk is the potential that a threat can exploit the vulnerability to damage the assets in the system. We can regard it as the injection or result of the 2 items above. We know that threat may exist or not, so does vulnerability, which means that if there are threats but no vulnerabilities, the whole system may in a state of low risk and similarly, if there are no threat but some vulnerabilities, it may also lead to low risk.

ISO/IEC 27005 provides a definition for risk in Information Technology area 41[5]. According to the definition, IT risk is the potential that a threat may exploit the vulnerabilities of assets in the organization and do harm to it. It can be measured by the combination of the probability of occurrence of an event and its consequence and can be described simply by the formula (1), where i is the number of threats, p is the probability that the threat may happen and d is the damage caused by corresponding threat.

$$\sum_{i=1}^{n} p_i \times d_i \tag{1}$$

Figure 1 published by OWASP [6] shows the relationship among threat, vulnerability, risk and attack. As we talked above, there are some vulnerabilities exist in the system which is described as "Weakness". The attackers can take use of some threat agents like scripts and tools to attack these "weakness" thus to take control of some security policies and as a result, part of or the whole system suffers some negative impact. The probability this process may happen is

described as "risk".

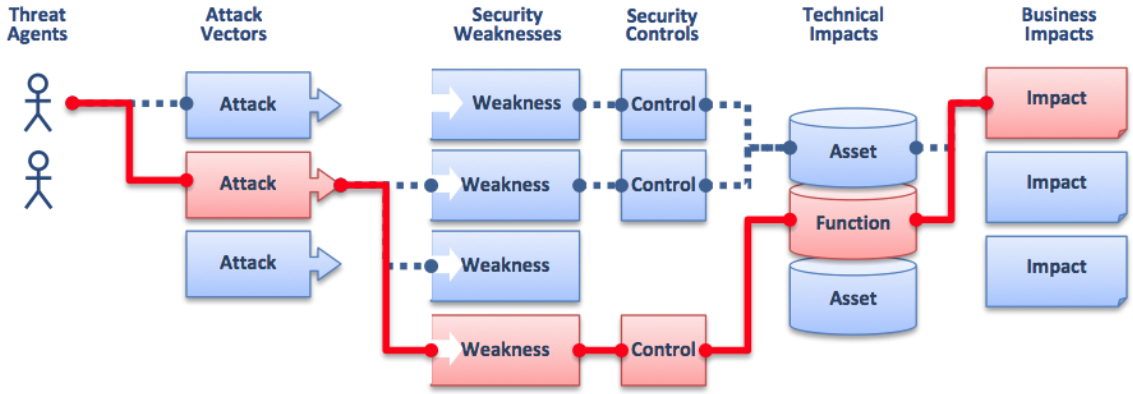


Figure 1 OWASP: Relationship Between Threat Agent and Business Impact

2.3 Security Issues

Security vulnerabilities have existed since the first day that the Internet was invented and develop with the technology. As a result, it is impossible to learn all the vulnerabilities in the world and an official ranking for the most concerned vulnerabilities is needed. This project mainly refers to Open Web Application Security Project (OWASP) and Common Weakness Enumerations (CWEs) for the most common vulnerabilities to study.

OWASP is short for Open Web Application Security Project, and is an online community that free and open resources like articles, tools and technologies in the field of web application security. OWASP Top 10 identifies the top 10 most critical web application security risks for companies and developers and is updated regularly. The latest version of OWASP Top 10 is published in 2021 and Figure 2 [7] shows what it contains and the difference between it and previous version (published in 2017) and below are some descriptions for all of them.

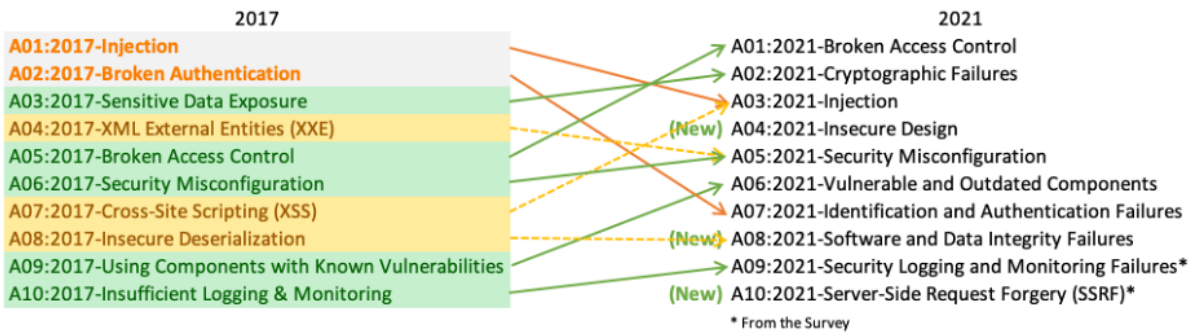


Figure 2 OWASP Top 10 in 2021

2.3.1 Broken Access Control

Access control policy can force the users to act only inside the permitted area of the developer, which can protect the application from unauthorized access, data exposure and some other damages. Broken Access Controls means that some vulnerabilities can be detected in the access control policy from the aspects like the users are given unnecessarily high permissions to some sensitive parts of the application, the access control is bypassed by modifying requests, the verification code is leaked and the access control is misconfigured for some APIs.

This can be prevented by denying public resources to access the application and to implement the completed access control policy for the whole system. It is also important to record all the vulnerabilities encountered and fix them in future versions.

2.3.2 Cryptographic Failures

This is also known as Sensitive Data Exposure according to Figure 2. It may happen when the sensitive data like password is not protected or encrypted properly. For example, the database uses deprecated cryptographic function to encrypt all the passwords can be easily attacked by SQL Injection. To minimize the risk of this vulnerability, the first thing developers should do is to classify all the data processed in the application and determine the secure level for them. For example, the password and business secret need higher level of security than the user's nickname. For all the data need higher level of security, things like how data are transmitted, what cryptographic algorithms to use and how to certificate and validate the receiver need to be considered.

2.3.3 Injection

This is the most studied vulnerability in this project, which includes some classic ones like Cross-Site Scripting (XSS) and SQL Injection. It can happen when the application does not validate or filter the data from the user, the malicious data is directly used by the application. The damage to the application or system depends on the position of injection. For example, if the hostile data injected to the database usually have longer lasting damage to the application than those injected only to the web page.

To prevent this kind of vulnerabilities, it is strongly recommended to separate the data from the commands and validate all the input data from the user to filter those malicious contents out. Also, the developers should choose safer APIs and SQL controls to limit the unauthorized operations and avoid disclosure of sensitive data. Source code review is also a good method to

detect this vulnerability.

2.3.4 Insecure Design

This is not the source of all other vulnerabilities because there is a difference between insecure design and insecure implementation. A secure design can still have implementation defects that can be exploited by the vulnerabilities, while an insecure design cannot be fixed by a perfect implementation. To prevent this, the developer should use a secure development lifecycle to evaluate and design the security and privacy-related controls, limit the resource consumed by user or service and use threat modelling for critical authentication.

2.3.5 Security Misconfiguration

This usually happens when the application has some insecure configurations, such as enabling or installing unnecessary features, default account and password remain unchanged and the latest security functions are not enabled or some out dated components are used. This can be prevented by removing all the unnecessary functions, documents and samples and establishing automatic process to deploy the configurations and environment.

2.3.6 Vulnerable and Outdated Components

Just as the name implies, this kind of vulnerability happens when the components in the application is deprecated for reasons like updating, insecure or not supported. And the recommendations are also quite easy: removing the out-of-date components and replace them by the latest ones from the official sources, and monitoring the unmaintained libraries and other components in case there are discovered security issue.

2.3.7 Identification and Authentication Failures

This usually happens with brutal force attack and cryptographic failures mentioned in section 2.3.2. The attackers can use scripts or programs to “try” the valid username and password automatically, which is also known as brute force. In this condition, the database with weak secure authentications can be intruded, especially for those which still have test accounts such as “admin”, “abc123”. Besides, the attackers can also take advantage of leaked session ID to pretend to be a normal user and make damage to the application system. An effective way to avoid this is to enhance the authentication with the help of verification code, secure questions and face recognition to avoid brute force. Also, improving the complexity of passwords and removing simple test accounts are also required to avoid this kind of vulnerability.

2.3.8 Software and Data Integrity Failures

The software and data integrity are related to the basic infrastructure of the code, and the vulnerability may happen when the application depends on untrusted libraries, modules or other resources. The attackers can upload their updates of untrusted resources and distribute it easily to all the users when they update their libraries. A solution to this vulnerability is to verify all the external tools and components used in the application are from expected and trusted resource by using digital signatures or similar mechanisms and ensure the raw data will never be sent to untrusted clients without integrity check.

2.3.9 Security Logging and Monitoring Failures

The security log and monitor aim to detect the active attacks but sometimes the logging can be insufficient. This can happen when auditable events like failed logins are not logged, warnings and errors are not clear enough and the logs lack of backup. And to prevent this, the developer should ensure that all the access control and failures can be recorded with sufficient user data, the logged data are encrypted in case of injection and establish an error report mechanism.

2.3.10 Server-Side Request Forgery.

This vulnerability occurs when the application is fetching a remote resource without validating the URL submitted by the user. As a result, the attacker can send a crafted request to unexpected links even if the application is protected by firewalls or VPN. The developer can again filter and validate all the data submitted by the user, set a series of URL schemas and avoid sending raw responses to the client to prevent this kind of flaws.

2.4 Vulnerabilities Focused

This section introduces the vulnerabilities focused in this project, which includes Cross-Site Scripting (XSS), Server-Side Template Injection (SSTI) and NoSQL Injection.

2.4.1 Cross-Site Scripting (XSS)

This is a type of vulnerability that belongs to Injection, and the most focused vulnerability in this project. It mainly happens when malicious contents like scripts is submitted by the user or attacker and injected to the application. According to the place of injection, there are 3 kinds of XSS, which are the reflected, stored and DOM-Based XSS. Only the first 2 can happen to the server side, and the 3rd one only make changes to the web page, thus will not be discussed. For the stored one, the malicious content can be injected to the server of the application by

inputting scripts in somewhere that can store your input, such as the User Name box when you register a website. The malicious scripts can be stored in the database and every time the user opens the website, the malicious content will be executed.

For the reflected one, the attacker can add the malicious content to the URL and send the fake link to the users via e-mails, messages or some websites. When the user opens the link and submits the specially crafted contents, the malicious script can steal the input and call particular interface to attack the server.

2.4.2 Server-Side Template Injection

This is also a vulnerability belongs to Injection. The way it happens may similar to reflected XSS but it only attacks the server side. In Python Flask, the Jinja2 template engine allows the developer to transmit data between Python and HTML codes, which is called the template. Also, Python Flask offers a function called *render_template_string()* that allows the developer to write HTML codes directly in Python files in a string. Thus, this vulnerability is about injecting malicious contents to the string parameter that contains HTML codes. The attacker can easily get some sensitive data like SESSION_KEY by injecting particular scripts to the original URL.

2.4.3 NoSQL Injection

As mentioned in section 2.1, this project uses NoSQL MongoDB as the database to store the data. This kind of database does not like traditional SQL that requires relational tables and fixed format for the data, instead, it provides less restrictions and consistency checks and offer better performance and scaling benefits. However, it can still be attacked by procedural language, which may have worse damage than traditional SQL Injection vulnerabilities.

Take MongoDB for example, the NoSQL Injection vulnerability is mainly caused by the operator *$where* that can execute JavaScript codes. This operator is mainly used by the developer as a filter to select particular attributes of data. However, this is not recommended because the attacker can inject malicious script codes to the query statements and manipulate the data in the database.

Chapter 3: Design and Implementation

This chapter describes how the web application is designed and implemented. A simple IoT system is taken as reference to test some particular aspects of vulnerabilities in Python Flask. It is a simple deployment inspired in a demotic setting for remote control of the temperature of a house by monitoring and controlling the involved systems' devices.

3.1 Basic Requirements

Basically, the application should be able to check the vulnerabilities for the user. To make it more practical, a simple IoT system that contains a series of devices and their communications will be introduced. Once the user logs in to the application, a page that contains all the devices should be shown and the user can choose any one of them to test the vulnerabilities it may have. After the test, the user will receive a report that contains the vulnerabilities detected, the security level of the application and recommendations to the user. These recommendations can also be accessed via the main page that contains all the vulnerabilities supported by our application, which can be visited by all the users log in to the application including those use the visitor mode. For those who does not have an account, the application should also offer register service to them and for the visitors, the main page mentioned above is the only page they can access. This page should contain all the vulnerabilities supported by this application together with some examples and solutions to them.

3.2 Overview of the Application

Once the application is started, the welcome page will be shown and to use this application, the user first need to log in his own account. Figure 3 shows how the application mainly works. For those who does not have an account, they can try to access the application with the visitor mode to have a look at all the vulnerabilities supported by this application, or register a new account. If the new account is registered successfully, the information entered by the user will be stored to the MongoDB database and the user will be guided back to the welcome page for to log in this application with the new account. Once the user manages to log in, a page that contains a device list in the user’s house will be shown for the next step.

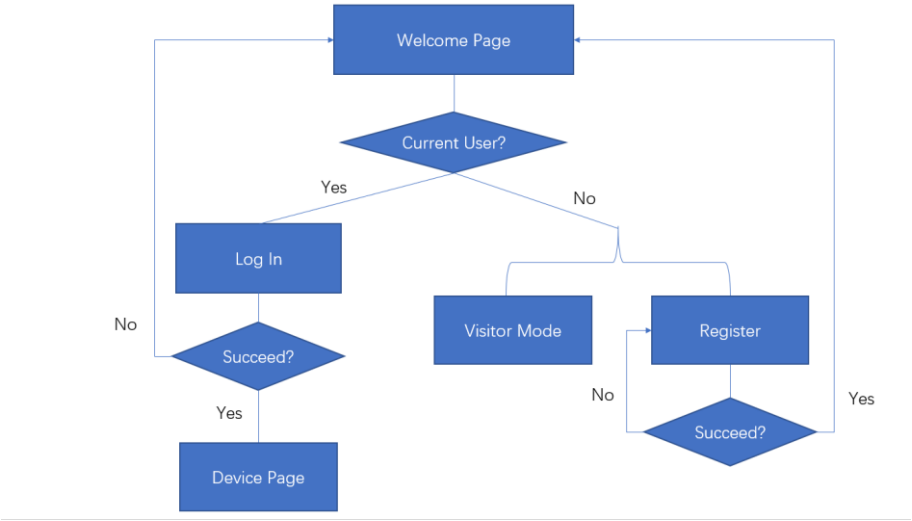


Figure 3 Work Flow for the Welcome Page

The device page will show all the devices and their basic information in the user's house, and the user can choose any of them for the detailed information. Just as shown in Figure 4. Once the user chooses a device to have a look, he will be guided to another page that contains the detailed information of the device. Also, in this page some check boxes that contains all the vulnerabilities can be checked in this application are offered to the user. The user can choose one or more vulnerabilities to have a check and finally a simple report that contains the vulnerabilities detected, security data and recommendations will be shown to the user.

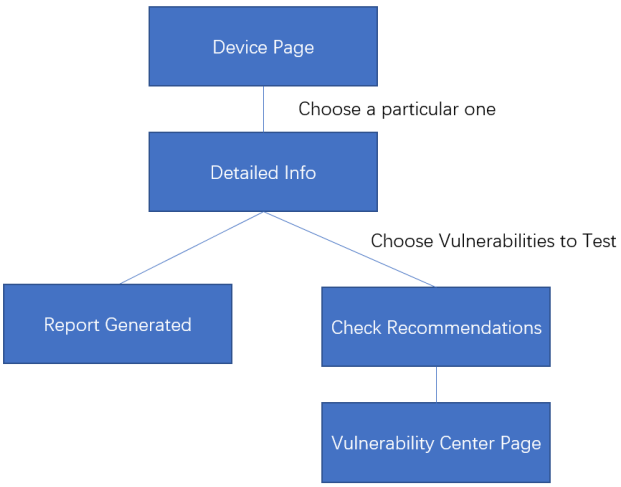


Figure 4 Work Flow for the Device Page

As for the simple IoT system, imagine a smart house with air conditioners that can adjust the settings according to the environment temperature. In this situation, there are 2 different

temperature monitors which are connected to 2 air conditioners in different rooms respectively. The monitor and air conditioner in the same room can be paired via Wi-Fi or Bluetooth network. There is also a router that can generate Wi-Fi signal and a server (the owner's computer) to process requests and send instructions. JSON and MongoDB are used to store the basic settings of the devices and the user's information.

Below is some detailed information for the devices in the system:

- The router "router_01": located in the dining hall and generates the network "wifi_01".
- Air Conditioner "AC_01": located in the bedroom and can be connected to the network "wifi_01". It can also generate the Bluetooth network "blue_01" and use it to receive data from the monitor.
- Air Conditioner "AC_02": located in the living room and can only be connected to the network "wifi_01". Commands can be sent via the network.
- The sensor "temperature_monitor_01": connected with "AC_01" via the network "blue_01", and can send temperature data to it.
- The sensor "temperature_monitor_02": connected with "AC_02" via the network "wifi_01", and can send temperature data to it.
- The server "computer": connected to both networks and can monitor the whole application system.

All the information of these IoT devices are stored in a JSON file named *devices.json,* and will imported to the server side of application for process when needed.

3.3 Application Implementation

3.3.1 Model of an IoT Device

The implementation of a certain device is quite simple, which is defined by the Flask route *@app.route('/<username>/dev'),* where *<username>* is the name of current user and can vary between different users, and the statement *name = f'{username}'* can be used to extract the value of the parameter to *name* for future use, where *f* means that the variable inside the followed curly brackets is a formatted string literals. After this, the JSON file that contains all the device data will be imported by the code in Code 1. The type of the variable *data* is 'dict', which means that the file has been imported successfully and stored in the format of python dictionary. All the related data will be sent to the client side with the help of Jinja2 template engine embedded

in Flask and then shown to the user.

```
with open("devices.json", "r", encoding="utf-8") as f:
    data = json.loads(f.read())
```

Code 1 Import the JSON File

For the client side, the data is received and processed with the combination of HTML and Jinja2 statements shown in Code 2. This code is part of an HTML table and uses *for-loop* to fill in different lines of data. The data transmitted from the server side is a python dictionary named *devices* and the *for-loop* can travel the whole dictionary for all the data we want. For the last column of each device, a link that contains the username and the name of current device can guide the user to the page that contains detailed information of particular device. Figure 5 shows how the page looks like finally.

```
{% for device in devices %}
<tr>
    <td>{{ device.category }}</td>
    <td>{{ device.name }}</td>
    <td>{{ device.model }}</td>
    <td>{{ device.position }}</td>
    <td>
        <a href="http://127.0.0.1:5001/{{ user }}/dev/{{ device.name }}">VIEW</a>
    </td>
</tr>
{% endfor %}
```

Code 2 How the Client-Side Processes Data

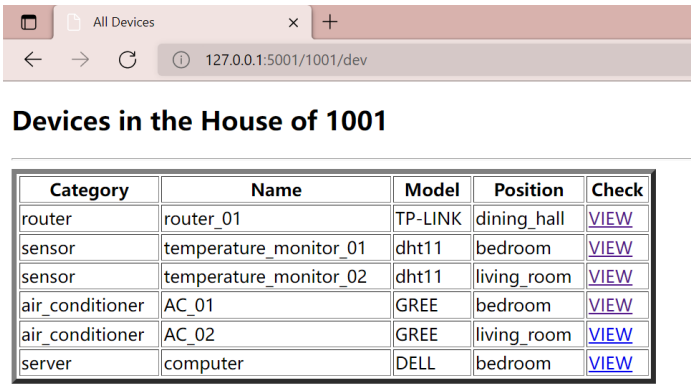


Figure 5 Page for All Devices

The implementation of the page that contains detailed information of a particular device is similar to the one contains all devices, which includes a table that lists all the details with the help of HTML and Jinja2. Besides, this page also offers some check boxes that describes the vulnerabilities supported by the application for the user to choose. The implementation for the vulnerability test will be talked about in section 3.4, and the presentation of this page is shown in Figure 6.

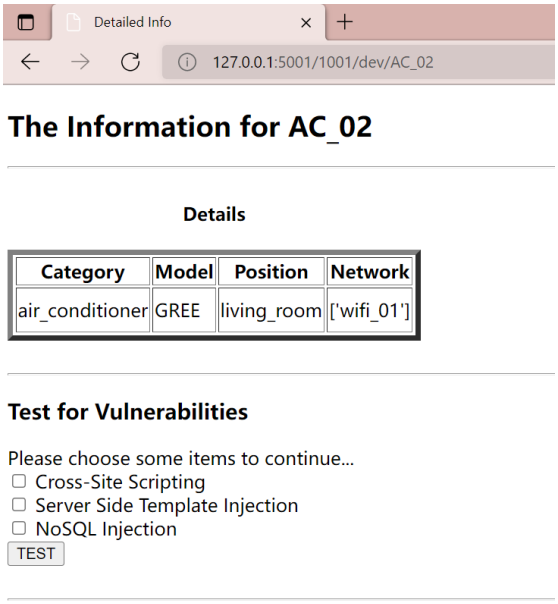


Figure 6 Page for Certain Device

3.3.2 How the Application is Invoked

As mentioned in section 3.2, all the users will be shown the welcome page at the start of the application. When the user tries to log in, the input username will be firstly looked for in the MongoDB database by using the function *find_one()*. If it cannot be found, an error message

will be shown to the user while if found, the corresponding password will be checked. Only when both the 2 inputs are correct will the user be guided to the device page mentioned in section 3.3.1. The error message is generated in the server side via *flash()* function and received by *{{get_flashed_messages()[0]}}* on the client side. Figure 7shows an example when the error message is triggered. The register page is also in similar format, in which there are input boxes for the user to enter the username and password and a mechanism to avoid the same username registered. Once the user is registered successfully, he will be guided back to the welcome page and can log in with the new account. For the visitors, the application offers a special account for them which can only access the page that contains all the vulnerabilities supported by this application, together with their examples and solutions.

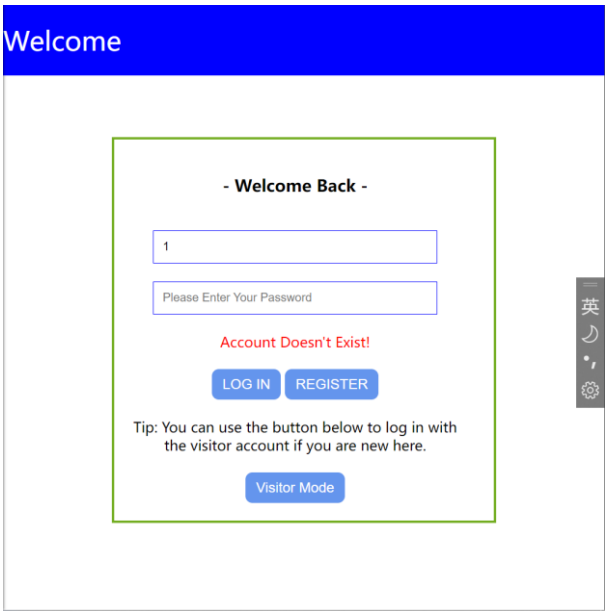


Figure 7 Login Error

3.3.3 How the Server Stores Information About the Readings of Sensors

It is important to know the user's critical operations for the security of the application. In this project, it is realized by the logger function of python. At the start of each router and function, a statement like *app.logger.info("The user " + user + " checks the details of " + device_id)* is added to keep record of the user's operations. Also, the function *logging.FileHandler('test.log', encoding='UTF-8')* is used to store all the logs to the particular file (*test.log* here). Figure 8 shows a screenshot of how the log file looks like.

```
2022-04-27 23:22:27,606 - INFO - app.py - r01 - 184 - The user 1001 checks the details of AC_01
2022-04-27 23:26:59,617 - INFO - app.py - welcome - 60 - Login Page Loaded
2022-04-27 23:26:59,620 - INFO - app.py - welcome - 102 - test
2022-04-27 23:27:03,032 - INFO - app.py - welcome - 60 - Login Page Loaded
2022-04-27 23:27:03,057 - INFO - app.py - welcome - 80 - Main Page Open Start
2022-04-27 23:27:04,488 - INFO - app.py - r01 - 185 - The user 1001 checks the details of temperature_monitor_02
2022-04-27 23:27:16,342 - INFO - app.py - r01 - 185 - The user 1001 checks the details of temperature_monitor_02
```

Figure 8 Log File

3.4 Security Checks

As mentioned in section 3.3.1, each device can be tested for the vulnerabilities supported by this application. By clicking TEST button in Figure 9, the chosen item will be sent to the server and processed separately.

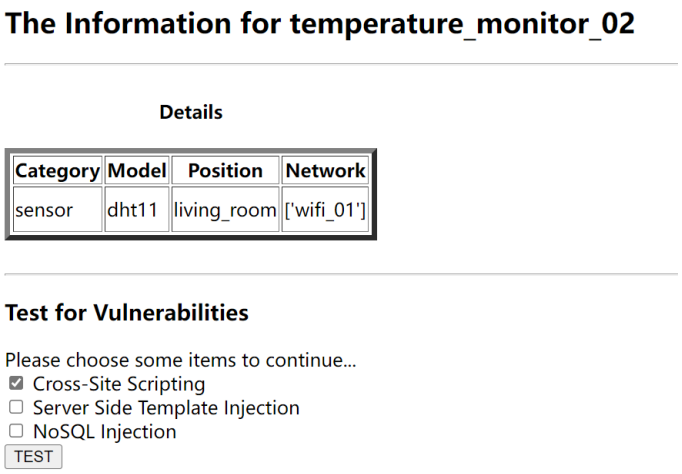


Figure 9 Device Details Page

As for how to test the vulnerabilities, currently most open-source test methods are based on adding payloads to the URL and try to send requests to the server. The test for vulnerabilities in this application is also based on this idea. A class named *TEST()* is created to process the test operations. 2 parameters are needed to call this class, which are the URL and the vulnerability to test. Every time a test is triggered, a text file named by current time will be created to store the test report by the function *open(file_name, "a", encoding="utf-8")* , where *file_name* is the name of the file and *a* means the file is opened for writing.

After this, the function *check(url, vul_to_check)* will be called to start the check of the vulnerability. This function will first find corresponding payloads according to the name of vulnerability to check and import them to a python list named *payload* line by line. Then the URL will be processed and crafted with the malicious payloads, just as shown in Code 3, where

url_test(new_url, 'get') in the last line is a function to send GET request to the server by using *requests.get(url, timeout=10)*. The result can be determined by the status code of the request, where 200 means the request is sent successfully and will be regarded as corresponding payload "works".

```
# start to check
# first split the url by ? to get the route
domain = url.split("?")[0]
for _payload in payload:
    # combine the route with the payload to test
    new_url = domain + _payload
    # test with GET requests
    result = self.url_test(new_url, 'get')
```

Code 3 How to Craft the Original URL

For the POST requests, things are different. The payload will be directly sent to the server as the "input data", and the judge for each condition remains the same as GET request, which is by the statue code of the request. At the end of each test, the result will be stored to the file created at the beginning of the whole test and a security level will be determined according to the results.

3.5 Vulnerabilities

3.5.1 Cross-Site Scripting (XSS)

XSS is one of the most classic vulnerabilities belongs to Injection. In this condition, the attacker can inject the malicious content to different parts of the application via trusted or untrusted methods. The malicious contents are mainly written in JavaScript, and it may also include HTML, Python or other languages or codes. With the help of these contents, the attacker can obtain the sensitive data on the web page, the server and even the database of the application. Based on where the malicious content is injected, XSS can be separated to 3 categories, which are Reflected, Stored and DOM-Based XSS. Since the 3rd one is caused by the change in the

browser environment of the client, thus not a kind of server-side vulnerability and will not be discussed here.

The Reflected XSS is mainly exploited by adding malicious codes and scripts to the URL of the application. It is called “Reflected” because the bad contents will not be stored to the server or the database but only be injected to the HTML or JavaScript codes of the web page to make some changes to the original page, where the attack codes are “reflected”. For example, most searching engines like Google and Baidu add the input content from the user directly after the original URL with some individualized contents based on the application itself or the browser. For example, if we search the word “XSS” on Google, the returned URL will be *https://www.google.com/search?q=XSS&some_other_contents*. For example, Figure 10 shows what happens when the user submits the script code *<script>alert('XSS Warning');</script>*. As a result, for some applications with poor or even no security configurations, the attacker can input and submit their malicious codes and get the data they want. The attacker can also encode their contents with Unicode, ASCII and base 64 to make it as innocent as possible, send it to the victims via e-mail and get the sensitive data as much as they want.

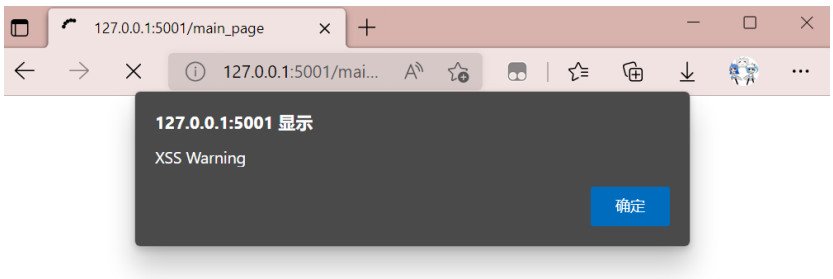


Figure 10 XSS Test

For the Stored XSS, it is similar to the reflected one but the malicious content will be stored to the server or application and executed every time particular user starts the application. This happens more to the conditions where the user is required to submit some characteristic contents such as the e-mail address, name and telephone number. These contents will be permanently stored to the database of the application and as a result, if the attacker submits some malicious content, he can execute the attack any time to get sensitive data and do harm to the application.

Similar things can also happen to the *eval()* function in Node.js. This function can also take the user input without any filtering and as a result, the attacker can exploit this vulnerability by passing malicious scripts to the server.

The most effective way to prevent XSS is filtering the input, since the malicious scripts

usually have some common characteristics such as special symbols like <, & and ‘, and some key words like *script.* As a result, all the symbols from the user input should be escaped based on some trusted libraries or APIs, such as the *escape()* function offered by Python Flask. Also, it is important to turn off HTTP TRACE of the server because the attacker can easily get the cookie via JavaScript. Besides, verification code and banning unauthorized scripts are also useful ways to avoid automatic script attacks.

Based on these ideas, this application uses the payloads in Code 4 to test the XSS vulnerability in the application. These payloads can be added to the URL for GET request tests and submitted directly to the server for POST request tests. Note that these are some sample payloads and full contents can be accessed in the file *xss.txt*.

```
// pop a prompt box that writes "1"
</script>"><script>prompt(1)</script>
// change some lower case letters to the capital to test if the script works
</ScRiPt>"><ScRiPt>prompt(1)</ScRiPt>
// an HTML tag that contains pop-up prompt
"><h1 onclick=prompt(1)>Clickme</h1>
// a link says "click me" that can lead to pup-up prompt
"><a href=javascript:prompt(1)>Clickme</a>
// %28 represents "(" while %29 represents ")", thus the JS command is "confirm(1)",
which means this is also a link that can lead to pop-up malicious script
"><a href="javascript:confirm%28 1%29">Clickme</a>
// the ciphertext encrypted by base64, whose plain text is "<svg/onload=alert(2)>"
"><a href="data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+">click</a>
// a text area that can be focused automatically and execute scripts
"><textarea autofocus onfocus=prompt(1)>
// a script coded by unicode, whose plain text is "confirm("1")"
"><a/href=javascript&colon;co\u006efir\u006d&#40;&quot;1&quot;&#41;>clickme</a>
// a script coded by unicode, whose plain text is "confirm`1`"
"><script>co\u006efir\u006d`1`</script>
```

Code 4 XSS Payload Examples

3.5.2 Server-Side Template Injection (SSTI)

Jinja2 is a template engine developed by the same author as Python Flask and embedded in it,

which allow developers to transfer individualized parameters and writing python commands directly in HTML codes via curly brackets ({{parameter}}, {% command %} and {# comment #}) instead of developing a huge amount of HTML files for each user and node. For example, the code * Welcome, {{ user.uid }} * shows how to generate welcome message according to different user ID.

Also, Python Flask offers a function called *render_template_string()*, which allows the deceloper to render HTML template codes directly from the server as a string variable, without importing HTML files from the template folder. Code 5 shows a complete example of the combination of Jinja2 and this method. As the code shows, the server has a default string variable named *name_s*, whose value is *Alice*. Also, the server can receive a GET request with the parameter *name* and stores it to the original variable *name_s*. The HTML template can receive the variable *name_s* from the server and store it as *name_c*, which will be finally shown to the user. In this case, since the content transmitted to HTML template is a variable (*name_c* in the example) instead of in the form of *%s*, HTML will escape all the characters automatically by default, which means that the page will show all the plain texts from the GET request directly instead of being affected or injected like XSS. Figure 11 shows the result of direct access to this URL: Since no GET request is sent to the server, the HTML shows the original value of *name_s* in the server directly and if we send some GET requests with the normal content *TOM* or even the JavaScript code *<script>alert(1)</script>* to the server, the requests will be received and the contents are shown by the browser, just as shown in Figure 12.

```
def hello_ssti():
    # set a variable in the server
    name_s = 'Alice'
    # receive a GET request
    if request.args.get('name'):
            name_s = request.args.get('name')
     # HTML template
     template = '<h2>Welcome back, {{name_c}}</h2>'
     return render_template_string(template, name_c=name_s)
```

Code 5 Combination of Jinja2 and Python Flask

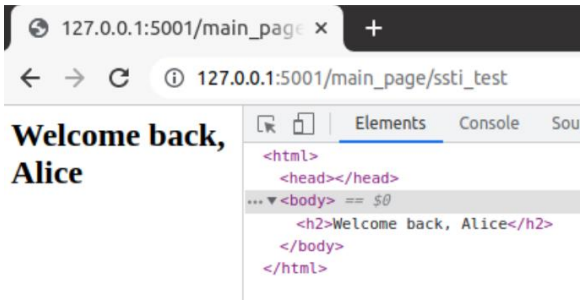


Figure 11 Result of Code 5

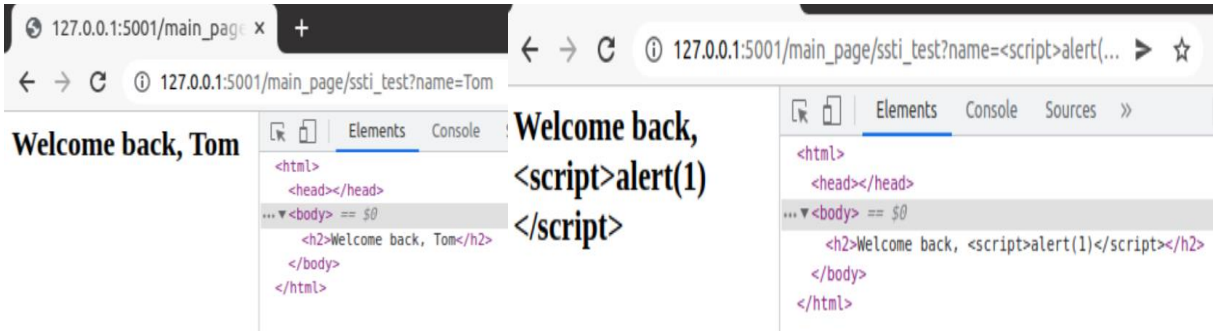


Figure 12 GET Request Test

However, if the template sets the content to show in the form of *%s*, i.e., use a string instead of Jinja2 template, things will be quite different. In this condition, HTML will not escape the contents in a string variable, instead, the whole *%s* variable will be regarded as an inherent statement in the HTML template. Thus, some malicious content can be sent to the server to cause XSS, SSTI and even Remote Code Execution (RCE). To simulate this condition, the only step is to change the variable *template* in Code 5 to *template = '<h2>Welcome back, %s </h2>' % name_s* and in this condition, the JavaScript statement in Figure 12 will be injected to the HTML and executed, which causes XSS, shown in Figure 13.

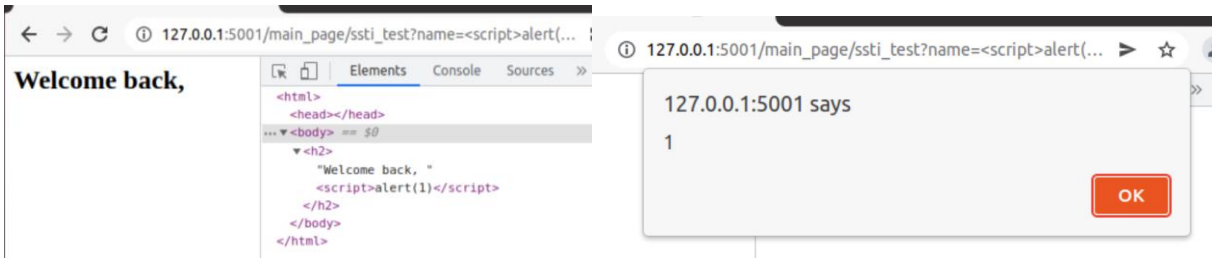


Figure 13 JavaScript Test

Since the content in the GET request will be regarded as normal HTML statement in this condition, Jinja2 can be used again to transfer malicious contents. This can be simply tested by sending the request *name={{1%2b3}}*, where *%2b* is the URL encoded format of the symbol "+". Figure 14 shows the result of this request, in which the contents in the Jinja2 template is successfully injected to the *template* variable on the server side and rendered on the client side.

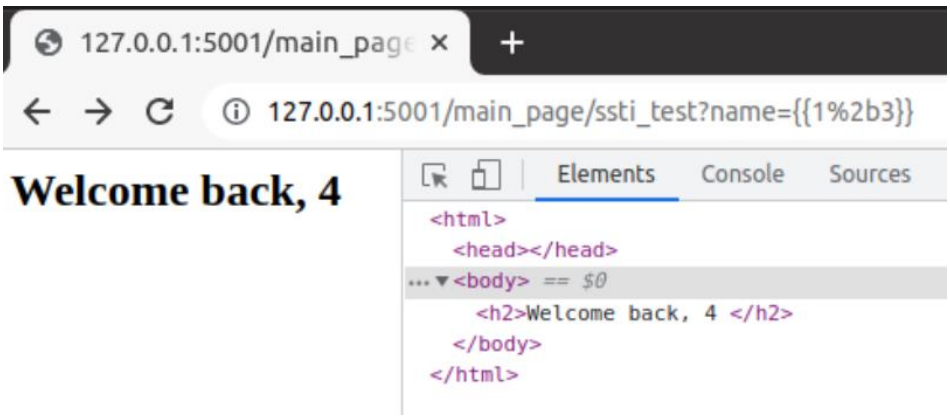


Figure 14 Simple SSTI Test

To read sensitive files, the attacker can sends the GET request that contains the statement *{{ get_flashed_messages.__globals__.__builtins__.open("/etc/passwd").read() }}*, where *get_flashed_messages* is a Python function usually used in HTML files with Jinja2 templates to show messages from the server, *__globals__* is another Python function that can get all the readable modules, keys and variables in the same module as current function, *__builtins__* is a python module that can call all the built-in Python objects directly (such as the functions *print()* to print something in the terminal and *open()* in this example), while *open()* is a classic Python function used to open files, and *read()* is used to read all the texts from it. After this, all the content in the file */etc/passwd* is injected to the server-side HTML template and shown on the screen, just as shown in Figure 15. This file is a database in Linux system that stores the information for all the user accounts on the system, which is owned by the root, can be modified by the root or the users with *sudo* privileges and read by all the users. Each line represents a single user in this system and is divided to 7 parts for different data. Take the first data line in Figure 15 for example, the first data is the username (*root*), then comes the password. For the old versions of Linux systems, the password is shown as encrypted text, which can be decrypted by the attackers easily but for recent versions of the system as shown in the figure, all the password is denoted as "x" and the encrypted text is stored in another file named */etc/shadow*. The next 2 items are UID and GID, which are the identifier for the user and the group the user

belongs to. The 5th one is the full name of the user and the next one is the home dictionary for the user. The last one is the absolute path to the user's login shell, which is started when the user logs into the system.

Though the password is invisible here, the attacker can still use other commands such as */home/username/.bash_history* to check the passwords leaked in the history operations and then do harm to the system.

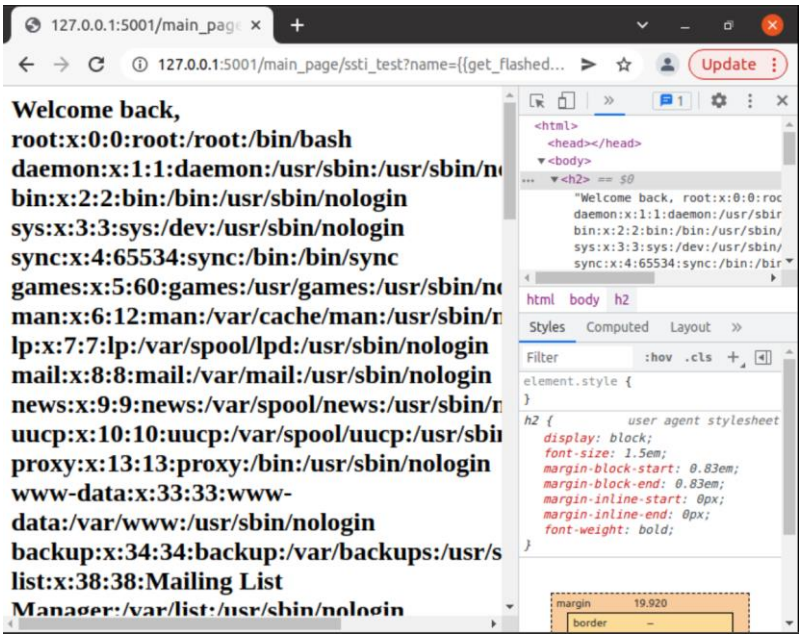


Figure 15 Result of Reading Sensitive File

Furthermore, if the attacker can also get more secret configurations of the server by injecting some other sensitive commands. The simplest example is using the payload "?name={{config.items()}}" to get many global attributions in current Python Flask environment, just as Figure 16 shows. With the help of these attributes like "SECRET_KEY", the attackers can craft fake sessions and use scripts to pretend to be a normal user and make further damage to the application system.

Though SSTI can cause huge damage to the application and system, the way to avoid it is quite easy, which are to stop using *%s* to transmit input content and reduce the use *render_template_string()* to render HTML templates directly in Python files. If not possible, use Jinja2 templates to transmit the parameters to the HTML template instead of the strings. The best way to prevent this is to write HTML files separately instead of writing strings directly inside a Python file.

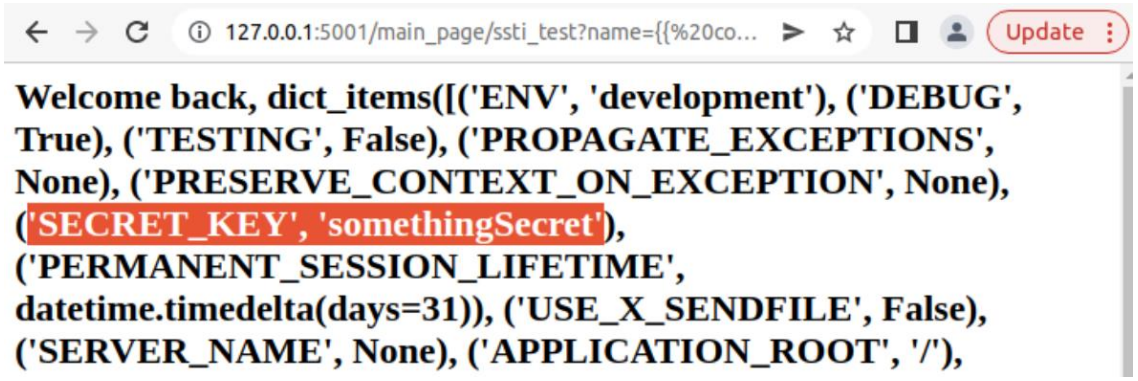


Figure 16 SSTI Configure Test

3.5.3 NoSQL Injection

Unlike traditional database like MySQL, NoSQL database allows the user to store the data with fewer restrictions on format and relations. This application uses a classic NoSQL database MongoDB as the database to store the user data and test for these vulnerabilities. Normally, the MongoDB API excepts the query statements in the form of BSON. However, according to the official documents [8], JavaScript can also be accepted by the API, which leaves room for arbitrary script input and execution.

$where* is a typical operator used in MongoDB as a simple filter, which is also the place the injection may happen. Also, since JavaScript is supported, more codes and functions can also be used for some complex queries. For example, the command *db.user.find({$where: function(){return obj.uid > 1001}}) can be used to find the detailed information for all the user whose *uid* is greater than 1001, just as shown in Figure 17.

```
MongoDB Enterprise > db.user.find({$where: function(){return obj.uid > 1001}})
{ "_id" : ObjectId("6221ab9cd4583b7f5b8ecc22"), "uid" : "1002", "psw" : "fba9d88164f3e2d9109ee770223212a0" }
{ "_id" : ObjectId("6221abd1d4583b7f5b8ecc23"), "uid" : "1003", "psw" : "aa68c75c4a77c87f97fb686b2f068676" }
{ "_id" : ObjectId("6221abe0d4583b7f5b8ecc24"), "uid" : "1004", "psw" : "fed33392d3a48aa149a87a38b875ba4a" }
{ "_id" : ObjectId("62242fabf7c8621411c97c62"), "uid" : "1005", "psw" : "2387337ba1e0b0249ba90f55b2ba2521" }
{ "_id" : ObjectId("622ef33702dcc81a75f45b1b"), "uid" : "1006", "psw" : "9246444d94f081e3549803b928260f56" }
{ "_id" : ObjectId("622ef34902dcc81a75f45b1c"), "uid" : "1007", "psw" : "d7322ed717dedf1eb4e6e52a37ea7bcd" }
{ "_id" : ObjectId("622ef34e02dcc81a75f45b1d"), "uid" : "1008", "psw" : "1587965fb4d4b5afe8428a4a024feb0d" }
{ "_id" : ObjectId("622ef35402dcc81a75f45b1e"), "uid" : "1009", "psw" : "31b3b31a1c2f8a370206f111127c0dbd" }
{ "_id" : ObjectId("622ef35a02dcc81a75f45b1f"), "uid" : "1010", "psw" : "1e48c4420b7073bc11916c6c1de226bb" }
{ "_id" : ObjectId("625a8b66a39642ba0dbbee05"), "uid" : "1020", "psw" : "65cc2c8205a05d7379fa3a6386f710e1" }
```

Figure 17 Example for *$where* in MongoDB

However, the attacker can also take advantage of the JavaScript queries to inject some malicious contents to the database or get the whole data list from it. For example, the developer can seek for particular user in the collection called *user* via the command *db.user.find({ $where: "this.uid == 1001" })*. In normal condition, the database will only return the data for the particular user whose *uid* is 1001. If an attacker manages to inject the statement "*; return ‘’ == ‘’* " behind it, i.e., to make the original query become

db.user.find({$where: "this.uid ===1001; return '' == '' "}), all the data in the *user* collection will be returned since all the data satisfy the condition *null == null* and as a result, the attacker can get the whole list of users in the database, just as shown in Figure 18, where the first query is the normal one, while the second is the one after injection.

An effective way to prevent NoSQL Injection is still filter the input from the user. Since most of this happens with JavaScript codes, similar escape methods as XSS can be used. Also, sanitization libraries can also be used to process the user input before it is submitted to the server or database. Finally, since Python offers its own API *pymongo* that can operate the MongoDB database with Python codes directly, it is strongly recommended not to use *$where* and JavaScript statements for query.

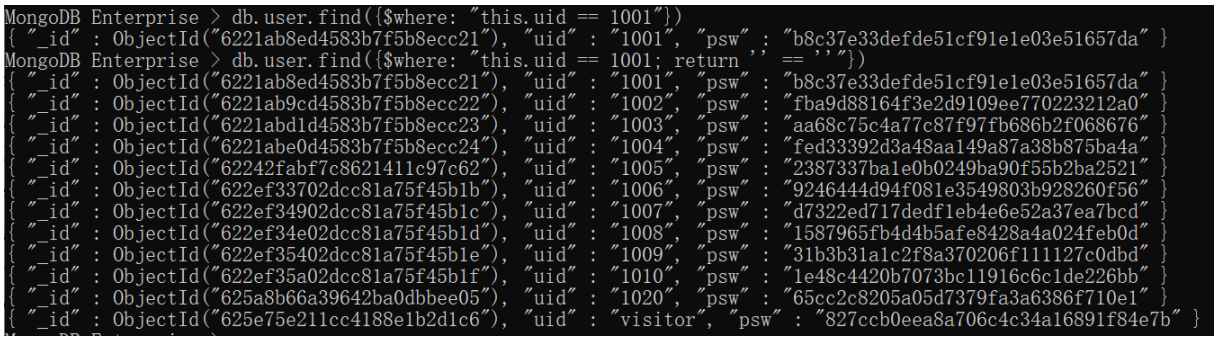


Figure 18 NoSQL Injection Test

Chapter 4: Results and Discussion

This section describes some experiments performed on the implemented system to extract information about the performance of the system. Some external APIs are used in order to test the performance on the client and server side of the application. Also, a metric is introduced to evaluate the vulnerability checking results of the application.

4.1 Performance

It is important to analyse the temporal cost of the web application at the different steps, which are:

- Client invocation of server URL
- Server runs a function in response to the client invocation
- Client receives responses from the server

4.1.1 Client-Side Method

Traditionally, *Date.getTime()* in JavaScript is used to set several time slots in the client side of the application and the time cost for a certain block can be determined by calculating the difference between the slots. This is quite useful but hard to calculate the time cost inside some event or process such as the onload event.

To solve this, W3C offers an API called Navigation Timing [9]. It is defined by High Resolution Timing API and can provide useful information to measure the actual performance during each period of a web site, which is much more accurate and reliable than other functions, such as *Date.getTime()* mentioned above.

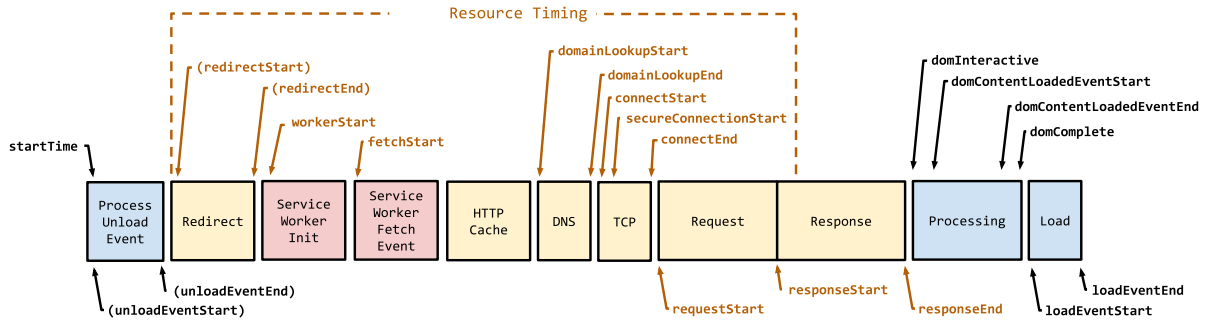


Figure 19 Processing Model of the Navigation Timing API

Figure 19 shows the structure of the processing model of the Navigation Timing API, which is also the loading process of a web page. When the browser starts to unload the previous document, *startTime* will be triggered to start the timing period for this page. For example, currently the user is browsing the website *google.com*, then he types another URL (e.g. *youtube.com*) in the address bar of the browser and presses the "Enter" key on the keyboard. At this time, the browser executes following operations: first to unload current document (*google.com*) and second, request for the next document (*youtube.com*). The value of *startTime* is the time that the browser starts the first step. If the previous document is empty, the value of *startTime* will be equal to *fetchStart*, which is the time before the browser sends any request. Also, for some particular browsers, *navigationStart* instead of *startTime*. Also, If the previous and requested documents are of the same domain, then *unloadEventStart* and *unloadEventEnd* will represent the start and end time that the browser unloads the previous document respectively. Otherwise, both of them will be 0.

After this, if the web page comes from redirection, the redirecting process is triggered , which will be timed by *redirectStart* and *redirectEnd*. Then the browser checks the cache during the period between *fetchStart* and *domainLookupStart*, and the latter one is also the start time of DNS query, with *domainLookupEnd* denotes the end time. If the browser doesn't carry on this operation (e.g. use cache instead), both of them are equal to the value of *fetchStart*.

Then, *connectStart* and *connectEnd* will be triggered to record the time that TCP starts to establish the connection and succeeds in doing this. Also, if the browser doesn't need TCP connection, then both of them are equal to *domainLookupEnd*. When these operations are finished, the browser starts to send requests (to the server, cache, local resource, etc.), which will be recorded by *requestStart*, and the moments that the browser receives the first and last bytes of response data from the server (or other resources we mentioned above) will be recorded by *responseStart* and *responseEnd* respectively.

Above are the main period will be used for this application. With the help of the *PerformanceNavigationTiming* interface under this API, the time cost in millisecond (ms) during each period can be easily obtained.

4.1.2 Server-Side Methods

For the server side, the Python function *time.perf_counter()* can be used to calculate how long it takes to run a function. This is a new function under the *time* module since Python 3.3. It

returns the value of a performance counter in second, and has the highest resolution to measure a short period of time in Python [10]. It can be called at the start and end point where timing is required and the time elapsed can be also easily got by calculating the difference between the 2 calls.

Besides, since the performance tests should be repeated for hundreds of times to make it statistical, it's impossible to test manually. Thus, an API named Selenium can be used to assist the tests. This is a powerful package that can automate browser interaction from Python programs [11]. Code 6 shows a simple example of how this API can be used and to repeat the test, a while-loop or for-loop can be added before starting the driver.

```
# set the driver for browser (Microsoft Edge)
s = Service(r"D:\Edge Driver\msedgedriver.exe")
# start the driver for Microsoft Edge Browser
driver = webdriver.Edge(service=s)
# load the web page
driver.get('http://127.0.0.1:5001/')
# find the input box by the id attribute of HTML code and send something
id_input = driver.find_element(By.ID, 'uid')
id_input.send_keys('1001')
# find the button and click
button = driver.find_element(By.ID, 'log')
button.click()
# close the browser
driver.close()
# end the process
driver.quit()
```

Code 6 Example for Selenium

4.1.3 Client Invocation Performance Test

The performance data can be calculated by the difference between *loadEventStart* and *fetchStart* that mentioned in section 4.1.1. And to get the data of time when using Selenium to execute the browser automatically, the code *invoke_time = driver.find_element(By.ID, 'invocation')* in the Python file can be used to get the data by finding the item whose value of *id* attribute in corresponding HTML file is *invocation* for further actions (e.g. store to the

database for analyze). Table 1 shows some of the data got from this test, and the line chart with full data is in Figure 20 , from which it can be obtained that the invocation of client is around 120 milliseconds and there is not an obvious pattern inside.

Table 1: Part of Client Invocation Performance Data

No.	1	2	3	4	5	6	7	8	9	10
Value (ms)	126	151	96	124	131	120	122	127	88	122

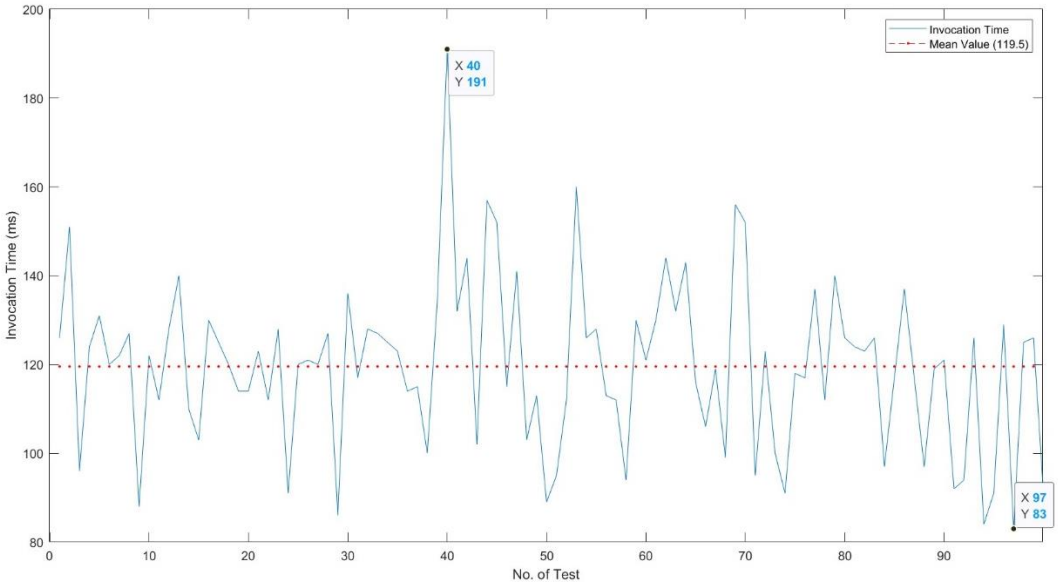


Figure 20 Line Chart for Client Invocation Performance

4.1.4 Server Function Performance Test

The login function is taken as example to calculate the time that the server runs a function in response to the client invocation. The moment that the server receives a POST request is regarded as the start point and the moment before it loads the next page is the end point in this test. By calculating the difference between this time slot, the result of how long it takes to run this function can be obtained. Table 2 shows part of performance data and Figure 215 shows the Line Chart with full data. The time data is much more stable except for the greatest and lowest values, which may be affected by the change of cache space.

Table 2: Part of Data for Server Function Performance

No.	1	2	3	4	5	6	7	8	9	10
Value (ms)	2.22	2.03	1.91	1.96	1.96	2.18	1.78	2.21	1.93	2.00

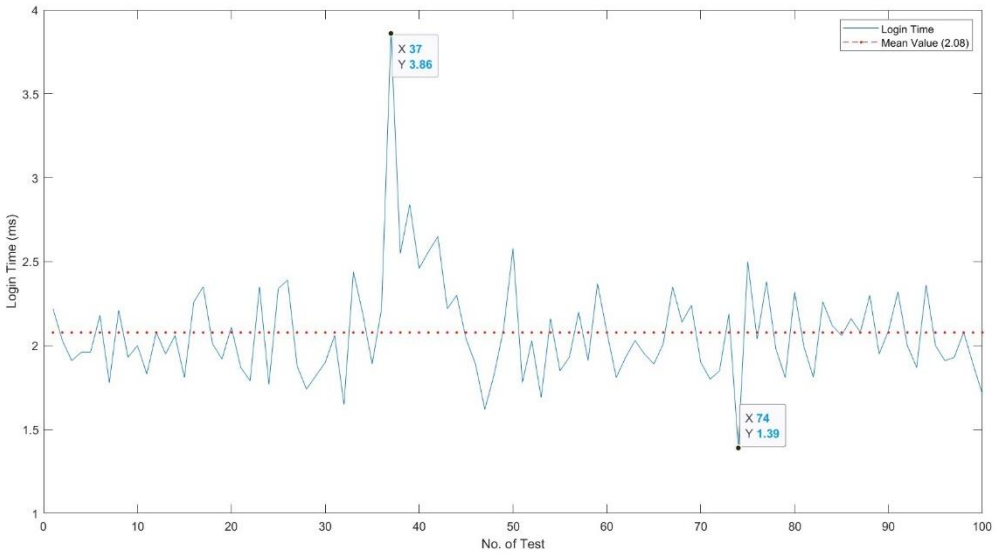


Figure 21 Line Chart for Server Function Performance

4.1.5 Client Response Performance Test

The JavaScript method mentioned above can be used again to get the time that the server returns response to the client by calculating the difference between *responseStart* and *responseEnd*. Similarly, part of data got from this test are in Table 3, and line chart with full data are in Figure 22.

Table 3: Part of Data for Client Reives Response

No.	1	2	3	4	5	6	7	8	9	10
Value (ms)	2	2	2	2	2	2	1	1	2	2

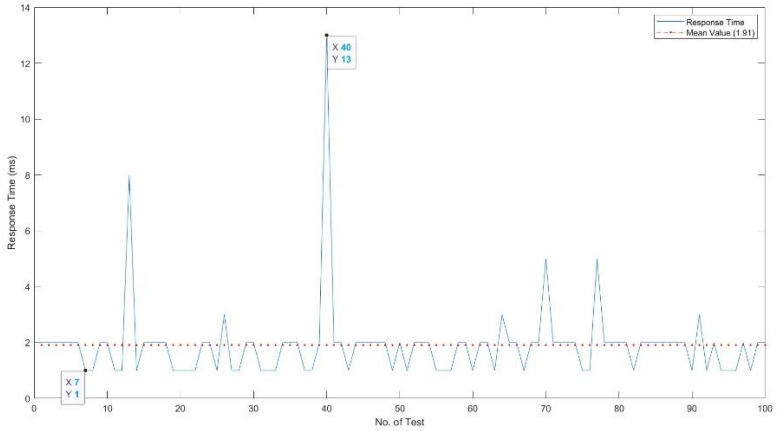


Figure 22 Line Chart for Data for Client Reives Response

4.1.6 Overall Data

Figure 23 and Figure 24 show the combination of data in client invocation and response performance test. We can see that the invocation time of the client mainly varies with the response time, since both of them are related to the performance of the server. It can be inferred that the performance of client invocation and response is related to the connection between the client and server and the cache storage of the server. The performance of response is also more stable than invocation since when the client starts to receive response, the connection between it and the server is already established and stable. Other factors may influence the performance may include the invocation time of the browser, the cache of the test program and the performance fluctuations of the test machine.

Figure 23 Combination of Client Invocation and Response Performance 1

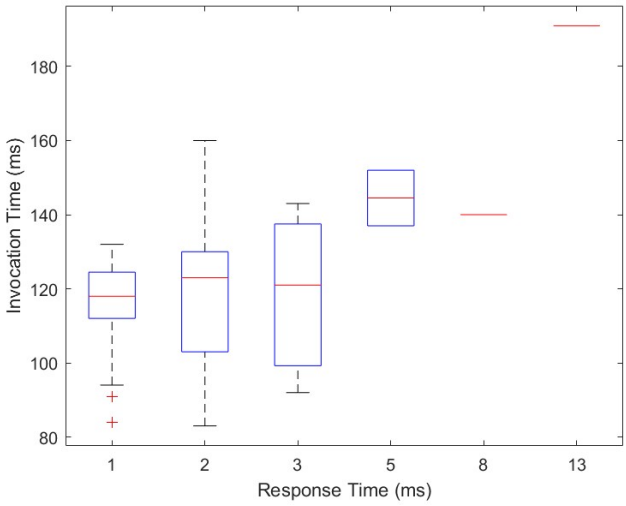


Figure 24 Combination of Client Invocation and Response Performance 2

4.2 Metrics to Evaluate Vulnerability Checks

It is important to offer the user a data result instead of putting numbers of code result in front of the user. Thus, some metrics are introduced to evaluate the result of vulnerability checks.

An IoT application can be divided into a 4-layer architecture, which are the object sensing layer to collect physical signals from the real word, the data exchange layer to transmit the data, the information integration layer to process the data and the application service layer to provide services for the users. The devices in the simple IoT system of this application can be classified to different layer as shown in table.

Table 4: IoT Devices in This Application and Their Layers

Device	Layer
router_01	2 - Data Exchange Layer
temperature_monitor_01	1 - Object Sensing Layer
temperature_monitor_02	1 - Object Sensing Layer
AC_01	4 - Application Service Layer
AC_02	4 - Application Service Layer
computer	3 – Data Exchange Layer

$$L = n \sum_{i=1}^{i_{max}} \mathrm{v}_i \times p_i \quad (2)$$

Also, Equation 2 can be used to calculate the risk level for each device, where:

- v represents vulnerability level. The ranking for different vulnerabilities is referred to OWASP Top 10 2021 as mentioned in section 2.3. The range of v is [1, 10], with the smaller number represents lower level and larger number means higher level.
- p is the percentage of failed tests. Since this application uses payloads in the database to test the vulnerabilities, thus some payloads can be passed and others may lead to failure. The number of failed tests can be counted and the ratio of failure can be regarded as the possibility that particular vulnerability may occur to corresponding device. It's obvious that the range of p is [0, 1].
- n is the layer of corresponding device, and it's obvious that the range of n is [1, 4].
- i represents the number of vulnerabilities tested, with the minimum value 1 and maximum value 3 in this application.
- L stands for the risk level calculated by all the parameters above. The range of L is [0, 120].

Since the decimal fractions can be incomprehensible for the users, a metric as concise as possible should be presented to the user. The risk level can be classified into 5 intervals to indicate the security rank R, with the interval [0, 24) means R = 5 to [96, 120] for R = 1. Figure 25 shows a report generated by this application. In this test, the user tested XSS for the device *temperature_monitor_01* that belongs to layer 1, and passed half of all the payloads which means the value of p is 0.5. More details can be seen in the figure.

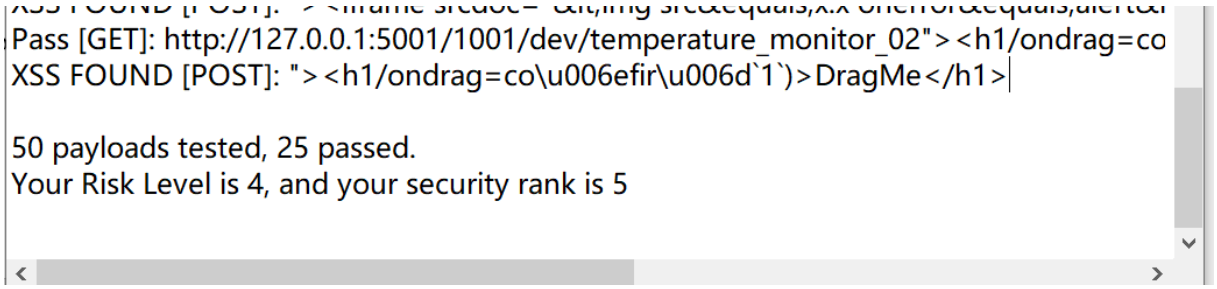


Figure 25 An Example Test Result

Chapter 5: Conclusion and Further Work

5.1 Conclusion of the Project

This project studied state of art web application technologies including HTML5, CSS, JavaScript, Python Flask, Node.js and MongoDB, and the security vulnerabilities related to them. The vulnerabilities studied are based OWASP Top 10 published in 2021, and the Injection vulnerability is most focused and studied in this project, and Cross-Site Scripting (XSS), Server-Side Template Injection (SSTI) and NoSQL Injection are the 3 main injection vulnerabilities focused in this project.

A web application is developed to offer vulnerability checking service for the user. The application offers different level of services for different kinds of user (registered user and the visitor). The visitors can only have access to the page that describes all the vulnerabilities supported by this project, while the formal user can access all the services. To make the application more practical, a simple Internet of Things system related to intelligent home that contains 2 temperature monitors, 2 air conditioners, 1 router and 1 computer as the server is established and simulated by different Python Flask routes. The formal user can have access to the information page of these devices, test for the vulnerabilities inside them. A report that contains the test details, risk level and security rank will be generated after the test. Also, the user can look for recommendations in the page that describes all the vulnerabilities supported by this application.

5.2 Further Work

First, currently this project only focuses on some particular vulnerabilities that belong to 1 category, other kinds of vulnerabilities are ignored for a long time after learnt about at the beginning stage of the project. More categories of vulnerabilities can be studied in the future to make this project more perfect and also, can be supported by the application of the project.

Also, the application of this project can be improved a lot. The most obvious part is that the user interface is too simple for an application and the number of vulnerabilities supported by it, as mentioned, can also be improved. Besides, this application only uses payloads to test the vulnerabilities, more methods can external tools can be added in the future to improve the accuracy and efficiency of testing. Finally, the improved application can be connected to the public network and offer service for all people around the world.

References

[1] M. Agreindra Helmiawan, E. Firmansyah, I. Fadil, Y. Sofivan, F. Mahardika and A. Guntara, "Analysis of Web Security Using Open Web Application Security Project 10," 2020 8th International Conference on Cyber and IT Service Management (CITSM), 2020, pp. 1-5, doi: 10.1109/CITSM50537.2020.9268856.

[2] Pressman, R. & Maxim, B. (2015). *Software Engineering: A Practitioner's Approach.* New York: Mc Graw Hill.

[3] *Web Application.* https://en.wikipedia.org/wiki/Web_application

[4] Shirey, R. W. (2007). *Internet security glossary, version 2*. RFC 4949, 126(7-8), 304-334.

[5] International Organization for Standardization. *ISO/IEC 27005:2018*

[6] Smithline N. *Relationship Between Threat Agent and Business Impact.* From OWASP. http://www.owasp.org/index.php/File:2010-T10-ArchitectureDiagram.png, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=12312894

[7] Open Web Application Security Project (2021). *OWASP Top Ten*. https://owasp.org/www-project-top-ten/

[8] *MongoDB Official Documents.* https://www.mongodb.com/docs/

[9] Mozilla. *Navigation Timing API*. https://developer.mozilla.org/en-US/docs/Web/API/Navigation_timing_API

[10] Python 3.9.10 Documentation - The Python Standard Library. time - Time access and conversions. https://docs.python.org/3.9/library/time.html

[11] Selenium Client Driver. Selenium 4.1.0 documentation. https://www.selenium.dev/selenium/docs/api/py/index.html

Acknowledgement

Thanks to my supervisor who offers me the chance to work on this project and continued to guide me and help me out when I was in difficulties.

Thanks to my roommates and friends who always stands with me however the situation goes.

How Time Flies.

How Things Changes.

How can the 4-year undergraduate life go so fast.

Finally, thanks to all the lecturers, professors, teaching assistants everyone in Beijing University of Posts and Telecommunications and Queen Mary University of London that have taught me for the whole 4 years.

Appendix

北京邮电大学本科毕业设计（论文）任务书

Project Specification Form

Part 1 – Supervisor

论文题目 **Project Title**	Improving security of web applications based on mainstream technology		
题目分类 **Scope**	Software Development	Implementation	Software
主要内容 **Project description**	Today, large amounts of web applications with little or no security configurations at all are accessible through the Internet. Many of these were academic exercises that required to use some container technology or document-based data bases from a purely functional perspective; this led to poor specialized configurations to ensure security. Many of these applications were later abandoned by their programmers or simply kept as initially deployed and they are still public and fully accessible. On the one hand, the project focuses on the stydy and analysis of selected state of the art web technologies and some of their related security flaws. As web programming technologies evolve, new flaws appear; and also appear the recommended programming patterns to overcome these flaws. Overall, cibersecurity in web systems is a dynamic and evolving process that requires much effort in continuous analysis of systems, of web programming tools, and systems prototyping. For this purpose, an initial basic set up of a web server will be put in place to analyze selected flaws on the field and selected common security missconfigurations. Then, a set of recommendations for their configuration and public set up will be designed and programmed.		
关键词 **Keywords**	Web application, web programming, web server, cibersecurity, web services, JavaScript, HTML5, CSS3, MySQL, MongoDB		
主要任务 **Main tasks**	**1** Recall state of the art web programming technologies and their most common configuration issues regarding security		
	2 Discover main flaws of current web application deployments regarding security, mainly from academic projects		
	3 Initial web application design and deployment to analyse its configuration failures regarding security		
	4 Elaborate set of security recommendations and configuration and program these repairs to the initial setting		
主要成果 **Measurable outcomes**	**1** Quality of the state of the art elaboration, of the elaboration of current security failures, and of the security repairs to be programmed		
	2 Programming of the initial and improved applications		
	3 Comparison and evaluation of the improvements with reference papers		

北京邮电大学 本科毕业设计（论文）任务书

Project Specification Form

Part 2 - Student

学院 School	International School	**专业 Programme**	**Internet of Things Engineering**		
姓 Family name	Song	**名 First Name**	Linxuan		
BUPT 学号 BUPT number	2018213147	**QM 学号 QM number**	190018720	**班级 Class**	2018215119
论文题目 Project Title	Improving Security of Web Applications Based on Mainstream Technology				
论文概述 Project outline **Write about 500-800 words** **Please refer to Project Student Handbook section 3.2**	Nowadays, with the development of network communications, more and more industries have started their online business based on web applications. At the same time, the number of people learning web development is also increasing. As a result, a large number of imperfect web applications used for academic exercises are linked to the Internet, which are lack of maintenance and may be abandoned by the developer shortly. These web applications with security flaws can usually be accessed freely on the Internet, which can be easily attacked by the hackers. The web application usually has C/S structure, which means that there's a client (usually a browser) that sends requests for the user to operate and a server that receives and processes the requests to provide services to the client. The common examples include online shopping, online forum and blog. These operations are served by web services, which provides an interface between the client and the server. It usually contains the Simple Object Access Protocol (SOAP) to interact between the client and the server, the Web Services Definition Language (WSDL) that describes the operations in XML format. The web services are so widely used that any tiny flaw can be taken advantage of by the attackers. Thus, security becomes vital important for a web application. According to the Open Web Application Security Project (OWASP), we are facing various web application security risks every year and the categories and the menace of which also changes every year. From last year's OWASP Top 10 project, we can see that the top risks for 2021 includes not only traditional categories like Injection and Cryptographic Failures but also some new ones such as Insecure Design and Server-Side Request Forgery. Therefore, we are going to carry out this project in order to study the state of art web technologies, the security flaws inside them and the recommended programming patterns to overcome these vulnerabilities. Through this project, we will develop a web application to help the users to find the security flaws and offer recommendations to them. Furthermore, we will help the users to repair the initial settings and configurations to minimize the security risks. To reach this target, we will have 4 main tasks: First, we need to learn the state of art web application programming technologies and their most common security configuration issues. These technologies include HTML5, CSS, JavaScript, Python Flask, Node.js, MongoDB and JSON. As we know, a web application contains client and				

	server. The client will be developed by HTML5, while the server can be programmed by Python and JavaScript. Since both Python and JavaScript can be used to develop the server application separately, we can develop 2 versions of application and find the flaws respectively. Second, we need to discover the main security flaws of current web application. All the technologies above can cause security flaws, and what we are going to do is to find these vulnerabilities, compare them and present them to the user. The third and fourth tasks are to develop the lite and advanced versions of application. For the first step, we will design and program the basic structure of the application, test it and add some MongoDB data storage to the server. Then we can develop a more complicated one to link it to the Internet and fulfil it with all the functions we talked above. **References** [1] *OWASP Top Ten*, https://owasp.org/www-project-top-ten/ . [2] Web Application, https://en.wikipedia.org/wiki/Web_application . [3] Web Services, https://en.wikipedia.org/wiki/Web_service . [4] M. Vieira, N. Antunes and H. Madeira, "Using web security scanners to detect vulnerabilities in web services," *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, 2009, pp. 566-571, doi: 10.1109/DSN.2009.5270294.
道德规范 **Ethics**	Please confirm that you have discussed ethical issues with your Supervisor using the ethics checklist (Project Handbook Appendix 1). [YES]

	Summary of ethical issues: (put N/A if not applicable) N/A
中期目标 **Mid-term target.** **It must be tangible outcomes, E.g. software, hardware or simulation.** **It will be assessed at the mid-term oral.**	1) Finish task 2: The investigation of security flaws in MongoDB data storage and for the combination of HTML5 clients and Python Flask servers or Node.js servers respectively. 2) Finish task 3.2, 3.3 and 3.4: The sample version of the web application, including the client and the server side, in which we can analyse the security flaws in user's application. 3) Finish task 3.5: The test for the sample application. 4) Finish task 3.6: Add MongoDB data storage to the server.

Work Plan (Gantt Chart)

Fill in the sub-tasks and insert a letter X in the cells to show the extent of each task

	Nov 1-15	**Nov 16-30**	**Dec 1-15**	**Dec 16-31**	**Jan 1-15**	**Jan 16-31**	**Feb 1-15**	**Feb 16-28**	**Mar 1-15**	**Mar 16-31**	**Apr 1-15**	**Apr 16-30**
Task 1 [Recall state of the art web programming technologies and their most common configuration issues regarding security]												
Study the language required for web application, including HTML5, Node.js, Python Flask, MongoDB and JSON	X	X	X	X	X	X	X					

Study web services			X	X	X	X	X					
Study cybersecurity issues related to these technologies			X	X	X	X	X					
Survey the web for simple example web applications that use these technologies					X	X	X					
Task 2 [Discover main flaws of current web application deployments regarding security, mainly from academic projects]												
Investigate security flaws for the combination of HTML5 clients and Python Flask servers			X	X	X	X	X	X				
Investigate security flaws for the combination of HTML5 clients and Node.js servers			X	X	X	X	X	X				
Study the flaws in MongoDB data storage					X	X	X	X				
Discover and compare the vulnerabilities in the technologies above			X	X	X	X	X	X				
Task 3 [Initial web application design and deployment to analyse its configuration failures regarding security]												
Design the structure of the sample application		X	X									
Program the client side of the application with HTML5		X	X	X	X	X	X	X				
Program the server side of the application with Python Flask			X	X	X	X	X	X				
Program the server side of the application with Node.js			X	X	X	X	X	X				
Test the sample application						X	X	X				
Add MongoDB data storage to the server side of the application					X	X	X	X				
Task 4 [Elaborate set of security recommendations and configuration and program these repairs to the initial setting]												
Modify the sample application to offer recommendations to multiple security flaws in user's application						X	X	X	X	X	X	X
Add the function of modify original code for the user						X	X	X	X	X	X	X
Connect the application to public web services							X	X	X	X	X	X
Test the advanced version of application								X	X	X	X	X

北京邮电大学 本科毕业设计（论文）初期进度报告

Project Early-term Progress Report

学院 School	International School	**专业 Programme**	**Internet of Things Engineering**		
姓 Family name	Song	**名 First Name**	Linxuan		
BUPT 学号 BUPT number	2018213147	**QM 学号 QM number**	190018720	**班级 Class**	2018215119
论文题目 Project Title	Improving Security of Web Applications Based on Mainstream Technology				

已完成工作 Finished work:

1 Summary of material was read or researched

1.1 Technologies

The technologies currently studied and used in the project includes the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript and Python Flask.

1.1.1 HTML

This is the most basic technology used in this project. It is used to display text, image, audio, video and many other kinds of the element in a web browser. These elements are described by HTML tags. An HTML file is usually composed of 2 parts, which are the head and the body. The tags `<html>` ... `</html>` indicate the start and end of the HTML document and wraps all other elements inside them. The head is marked by the tags pair `<head>` ... `</head>`. This contains the title and some other important settings for the HTML document. The body contains all the visible contents of the HTML document. Similar to the head, it is marked by the tags pair `<body>` ... `</body>`.

1.1.2 CSS

CSS is a language to “decorate” the markup language such as HTML. It can set the layout, colour, fonts and many other styles for the elements in the document to provide high accessibility and better experience for the user. There’re 3 places in which we can put CSS in an HTML file.

The first way is to use “style” attribute directly in an HTML tag. However, if we want to change the style for all the tags with the same name, class or id, this may seem to be hard to complete. Thus, we need another solution. We can write all these properties inside a style element with the help of selector and put it at the end of the head.

As for the selector, there’re also many kinds of it and the most used are type, id and class selector. To use them, we first need to declare particular tags and attributes to match and set the styles inside the curly brackets. Furthermore, if we have multiple HTML files to “decorate”, we can also create an external CSS file and import it in the head of the HTML document whenever we need, which is the third way to use CSS.

1.1.3 JavaScript

This is an important technology used in the client side of a web application together with HTML and CSS. It can provide many functions such as to control the elements in a HTML file, response to the browser events and transfer data between the client and the server. Similar to CSS, there're also 3 ways to deploy JavaScript for a HTML file. The first way is to use the tags pair `<script>...</script>` directly in the body, the second is to use it in head and the third is to write a JavaScript file and import it when needed.

1.1.4 Python Flask

This is a micro web framework written in Python. Currently we use this technology to build the server.

1.2 Python Flask Vulnerabilities

There're a lot of vulnerabilities can happen to Python Flask, such as Cross-Site Scripting (XSS) and Server-Side Template Injection (SSTI). In this part we are going to talk about these 2 vulnerabilities and solutions to them.

1.2.1 Cross-Site Scripting (XSS)

a) Introduction

XSS is a security vulnerability that belongs to Injection. There're several types of XSS, here we only talk about those related to Python Flask, which are reflected and stored XSS.

For the reflected XSS, the attacker can craft a special URL which looks innocent and trusted but contains malicious codes. These URLs can be sent to the user via email, SMS and even ads. When the user opens the URL, the server takes the malicious part and sends it back to the browser to execute, and then the attack starts. This attack can also happen when we enter some abnormal texts to the input box. A classic example for it is the search engine without any filter.

For the stored one, it's similar to the reflected XSS. However, this time the malicious content will be stored to the database and every time we open the website, the attack will be executed.

b) Solution

1) Filter

In most conditions, since we don't know what the input is and how the input value is processed, we cannot set a common rule for filtering. However, for some conditions like inputting the phone number and email address, it's quite useful to filter the input by limiting the length or setting some invalid characters.

2) Escape HTML

This is also based on the idea of filter. We can use library to escape some sensitive characters such as &, < and >.

3) HTTP-only Cookie

This allows a web server to set a cookie that is unavailable to client-

side scripts.

4) Verification Code
This can avoid some scripts pretend to be the user to carry on some dangerous operations.

1.2.2 Server-Side Template Injection (SSTI)

a) Introduction
This is similar to XSS. It happens when the user input is entered to the template directly without being filtered. Jinja2 is a template engine that is widely used in Python Flask. It uses double curly brackets ({{ }}) to wrap the parameter. Python Flask also offers a function called `render_template_string()` that allows us to write HTML codes directly in a string instead of a HTML file. As a result, if we want to pass a parameter in form of `%s`, the attacker can input some malicious commands to manipulate the templates which is fatal to the server and even the whole application.

b) Solution
Since this is mainly caused by the `%s` parameters passed by `render_template_string()`, we can first change them to `render_template()`, which is much safer than the previous one. If it's impossible to change, all the `%s` parameters should be forbidden from passing.

2 Summary of work was done

Currently, we have programmed several webpages and linked them with the Python Flask server. We have also studied some vulnerabilities that happens most in Python Flask, and applied some exploit examples and solutions in the application.

2.1 Login

To use this application, the user first needs to login. Currently we don't have a database so we only set 1 user in the server and use a simple `if` statement to verify. As we can see in the code below, only if both right ID and password are submitted will the user be directed to the main page, otherwise the login page will be reloaded.

```
1.  if uid == '123' and psw == '123':   # set to current user
2.      currentUser['uid'] = uid
3.      currentUser['psw'] = psw
4.      return redirect('/mainPage')
5.
6.  else:   # reload
7.      return redirect('/')
```

After logging in, the user ID will be stored and passed to the client. It will also emerge at the top of the web page.

2.2 Main Page

Currently we have 2 Python Flask Vulnerabilities in this page, including Cross-Site Scripting (XSS) and Server-Side Template Injection (SSTI). Since we have introduced them in section **1.2** above, here we talk about the exploit examples directly.

2.2.1 Reflected XSS

We use a simple input box and a link to test it. As we can see in the code below, the input box can only show what the user entered without any filter. As a result, it is easy to be attacked.

```
if request.method == 'POST':
    input = request.form.get('input')
    return 'Your input is: ' + str(input)
```

We also added a button to test Reflected XSS. When it is clicked some malicious code will appear in the box and if we submit it directly, the attack will be executed.

```
reflectTest.onclick = function(){
   var input = document.getElementById('input');
   input.setAttribute("value", "<script>alert('XSS Warning');<\/script>");
}
```

2.2.2 Stored XSS

Similar to the reflected one, we also use an input box to test and the difference between them is that the stored type can store the malicious code and every time the user opens the web page, it will be executed and do harm to the application.

2.2.3 SSTI

To exploit this attack, we set a page to show the name of the user in our database. If we use the link *http://127.0.0.1:5001/mainPage/sstiTest* directly, the web page will only show the name of the person, as we set in the template.

```
@app.route('/mainPage/sstiTest')
def hello_ssti():
    person = {
        'name': 'John',
        'age': '20',
        'password': 'abc123'
    }
    if request.args.get('name'):
        person['name'] = request.args.get('name')

    template = '<h2>%s</h2>' % person['name']
    return render_template_string(template, person=person)
```

As we mentioned in section **1.2.2**, we only need to change the parameter passed to the template to obtain some sensitive data. For example, the links below can get the password of the person and even config data, which is terrible for an application.

```
http://127.0.0.1:5001/mainPage/sstiTest?name={{person.password}}
http://127.0.0.1:5001/mainPage/sstiTest?name={{config}}
```

3 Problems were faced

We need to keep track of different versions of code so that it can be possible to roll back when a new version goes wrong. Also, I made quite a lot of mistakes when writing JavaScript codes, which made the pace slow.

4 Solutions were found

A repository is established on GitHub so that both me and my supervisor can keep record of the latest code and find problems from it. For the second problem, the only way is to search more and be as familiar as possible with JavaScript.

是否符合进度？On schedule as per GANTT chart?

[YES]

下一步 Next steps:

Continue studying the vulnerabilities related to Python Flask and HTML5.
Extract a general model for the server, vulnerabilities and their risk level.
Start learning Node.js and related vulnerabilities.

北京邮电大学 本科毕业设计（论文）中期进度报告
Project Mid-term Progress Report

学院 School	International School	**专业 Programme**	**Internet of Things Engineering**		
姓 Family name	Song	**名 First Name**	Linxuan		
BUPT 学号 BUPT number	2018213147	**QM 学号 QM number**	190018720	**班级 Class**	2018213147
论文题目 Project Title	Improving Security of Web Applications Based on Mainstream Technology				

是否完成任务书中所定的中期目标？Targets met (as set in the Specification)?
[NO]

已完成工作 Finished work:

1 Project Objective

With the development of network technologies, more and more web applications appear and so does the vulnerabilities. As a result, we are going to wok on this project to study the state of art web technologies and the vulnerabilities related to them, so that we can find solutions to the web applications at risk. To achieve this, we should finish the following 4 tasks:

a) Recall state of the art web programming technologies and their most common configuration issues regarding security. As we know, a web application of C/S structure is composed of client and server. For the client side, we need to study HTML5, CSS3 and JavaScript while for the server, Python Flask, Node.js, MongoDB and JSON are needed.

b) Discover main flaws of current web application deployments regarding security, mainly from academic projects. All the technologies we mentioned above can cause vulnerabilities, thus we need to learn about how they are caused and find them out in the application.

c) Initial web application design and deployment to analyse its configuration failures regarding security. To finish this task, we need to develop the basic version of the application that can test and exploit some vulnerabilities. The structure of the client and server side of the application should be developed using the technologies we mentioned above.

d) Elaborate set of security recommendations and configuration and program these repairs to the initial setting. This is to add more functions to the basic version of application. After we can test the vulnerabilities, we need to find solutions to the user of the application and make it in use.

2 Targets set in the specification of the project

There are 4 mid-term targets in the specification of the project, which are:

a) The investigation of security flaws in MongoDB data storage and for the combination of HTML5 clients and Python Flask servers or Node.js servers respectively.

b) The sample version of the web application, including the client and the server side, in which we can analyse the security flaws in user's application.

c) The test for the sample application.

d) Add MongoDB data storage to the server.

I have finished the first 3 tasks until this report is completed. For the fourth one, I didn't manage to finish it because I chose to use JSON and text files as the database of the basic version of the application, which is quite different from MongoDB. I had no knowledge about these technologies at the beginning thus I underestimated the time they might take. I'm still working to adapt it to my application and this should be finished before the deadline of the presentation video.

3 Work finished

3.1 Technologies studied

Until now, I have studied 3 technologies used in the client side and 2 in the server side of the application. The ones used in the client side are Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. These are the most basic technologies for a web application.

HTML is used to display text, forms, image, audio and many other kinds of the element that you want to show to the user of the application. An HTML file usually includes 2 parts, which are the head to describe some important attributes and settings of the file and the body to contain all the elements we mentioned above by using HTML tags and to set attributes and scripts for them by using CSS and JavaScript.

CSS, as we mentioned above, is to "decorate" the elements in an HTML file. It can set the layout, colour and many other styles and attributes for the elements to make the web page much more pretty and easier for the users to operate. There're 3 ways to use this technology, which are:

a) Use the "style" attribute directly to set styles for a single HTML tag
b) Use selectors in the head part of the HTML file to set styles for all elements that share the same class and even the same kind of tag.
c) Create single CSS file that contains selectors. This means that the same styles can be imported to different HTML files, which can make it easier to manage the styles of the web page as the amount of the code increases.

JavaScript is an important technology in the client side of the application. It can set scripts for all elements and movements in the web page to make the client response to the events and transfer data with the server. JavaScript can be used in similar way to CSS, and the differences between them are that JavaScript uses `<script>...</script>` tag directly in the body and head part instead of set the

attributes or using selectors.

For the server side, the technologies I studies are Python Flask and Node.js. Python Flask is a micro web framework written in Python. This is the main technology we use to build the server of the application. It uses Werkzeug as the Web Server Gateway Interface (WSGI) and Jinja2 as the template engine to implement the operations in the server and interactions between it and the client.

Node.js is a JavaScript runtime environment that can execute JavaScript codes outside of a web browser. It allows the programmers develop JavaScript commands in the console or terminal that can produce dynamic web page content before it's sent to the user's browser.

3.2 Vulnerabilities Studied

3.2.1 Python Flask Vulnerabilities

3.2.1.1 Cross-Site Scripting (XSS)

There're 2 kinds of XSS that can happen to Python Flask, which are the stored and the reflected. For the stored one, the malicious content can be injected to the server of the application by inputting scripts in somewhere that can store your input, such as the User Name box when you register a website. The malicious scripts can be stored in the database and every time the user opens the website, the malicious content will be executed.

For the reflected one, the attacker can add the malicious content to the URL and send the fake link to the users via e-mails, messages or some websites. When the user opens the link and submits the specially crafted contents, the malicious script can steal the input and call particular interface to attack the server.

The solution to XSS is mainly based on filter. We know that the script codes should contain some special characters like "<", ">" and "&", thus we can filter the sensitive characters with the help of some libraries to avoid the scripts. We can also set the input contents. For example, when we are requiring the telephone number from the user, only fixed-length numbers can be entered, thus to avoid complex scripts. Furthermore, we can set verification codes to avoid the scripts taking operations automatically.

3.2.1.2 Server-Side Template Injection (SSTI)

This is similar to XSS, but only happens to the servers. Jinja2 allows us to pass the parameters between the server to HTML codes and the server by using double curly brackets, and Python Flask also offers a function called **render_template_string()** that allows us to write HTML codes directly in a string (**%s**) instead of an external HTML file. Thus, the attacker can inject the malicious scripts to the **%s** parameter, pass it to the server and manipulate it, which can be critically dangerous to even the whole application.

Fortunately, this vulnerability can be easily sloved. The simplest way is

to stop using **render_template_string()** and import HTML files from the template folder. The other way is to ban **%s** parameters in the application.

3.2.1.3 Denial of Service (DoS), Distributed Denial of Service (DDoS)

DoS attack aims to make the web services unavailable to the users instead of taking use of security flaws. DDoS is a typical DoS attack which can generate a huge amount of traffic from many machines to make the server crash.

There're 2 common types of DoS vulnerabilities:

a) High CPU/Memory Consumption. The attacker can send crafted requests that could cause the system to take disproportionate resources to process.
b) Crash. The attacker can send specially crafted value to make the server crash.

A traditional way to avoid DoS is using Firewall. This technology is widely used in the network. However, traditional Firewall cannot afford DDoS, instead, it can easily become the victim of DDoS because the principle of Firewall is high-intensive detection, which can be taken advantage of by DDoS to consume enormous amount of network source and cause the server crash. A solution to this is Anycast. In this condition, the traffic flow caused by DDoS will be distributed to the closest nodes in the network and as a result, not all nodes may be affected.

3.2.1.4 CVE-2021-33026

CVE is short for "Common Vulnerabilities & Exposures". This is a dictionary that contains the vulnerabilities reported all over the world. It gives every vulnerability a unique number, description and solution so that the engineers from all around the world can find corresponding vulnerability easily. The title of this section means that this vulnerability is found in the year 2021 and the number of it is 33026.

The Flask-Caching extension up to and including 1.10.1 for Flask servers relies on Pickle serializer for serialization, which may lead to remote code execution or local privilege escalation. If an attacker gains access to cache storage like filesystem, they can construct a crafted payload, poison the cache, and execute Python code that can be fatal to the server. To avoid this attack, we can update the extension to the newest version or use safer serializer like JSON.

3.2.2 Node.JS Vulnerabilities

3.2.2.1 Remote Code Execution (RCE)

The **eval()** function takes input from input parameter without escaping or filtering the user input. It's a very common and typical example function. The attacker can exploit this vulnerability by passing malicious scripts to the input parameter.

This is similar to XSS we mentioned in section 3.2.1.1, and we can use similar way to avoid this attack, which is filter and escape. By filtering we can block the inputs that contains malicious scripts and by escaping, we can transform the sensitive characters to Unicode or some trusted ones to protect our application.

3.2.2.2 Remote OS Command Execution

This one is similar to RCE that we mentioned above. The key difference between them is that this vulnerability occurs because of unsafe uses of **exe.exec()** which allows application to interact with System/OS commands. As a result, the attacker can inject some malicious codes to this function and do harm to the server. Since this is also a kind of injection, the way to avoid it is also filter and escape.

3.2.2.3 Regular Expression Denial of Service (ReDoS)

This is a kind of Denial of Service (DoS) attack which we mentioned in section 3.2.1.3. The attacker can send large amount of false data, which consumes a huge number of server resource of the application and makes the application unavailable to other users. Thus, we an find 2 ways to avoid this vulnerability: one is to use a Firewall, the other is to improve the settings of the network.

3.2.2.4 Reverse Shell (Exploit Server-Side JavaScript Injection)

The attacker can use some scripts to craft JavaScript codes according to the IP address and local port of the attacker. It can be quite easy to execute when the attacker has direct connection with the node.js application or when the attacker and the victim are in the same network. When the connection is established, the attacker can inject the JavaScript codes to the application and take control of the server.

Unfortunately, currently there's no way that we can totally block this attack, especially for the servers connected to the network. But there're still some ways that we can mitigate the risk caused by it:

a) Limit the exploitation. We can set trusted IP addresses and ports that can access the server to reduce the risk of being attacked via strange addresses. Note that this can only limit the risk because some attacks can even be executed over DNS, which is hard to defend.

b) Remove unnecessary tools. We can remove all the unnecessary tools and interpreters to reduce the amount of reverse shell codes to make the attack much more difficult. This also cannot fully ensure the safety because a determined attacker can still find a working shell with harder work.

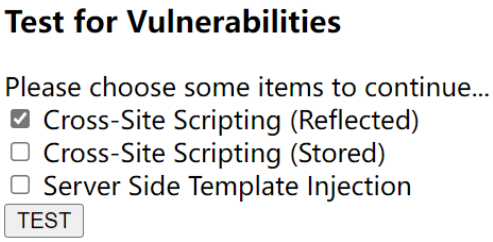
3.3 Application

In early term, I have developed a web application that contains the description, solution to some vulnerabilities and an interface to exploit them. Below is a simple example of it, which is an input box without any filter and can show what the user entered:

```
if request.method == 'POST':
     input = request.form.get('input')
     return 'Your input is: ' + str(input)
```

As a result, if we enter some script codes, they will be executed automatically by the server, just as the figures below show, the alert box appears, which means the script has been injected successfully.

This time, I take advantage of a vulnerability detector to detect the vulnerability in certain servers. On the client side, I offered some options for the user to choose:



By clicking **TEST** button, the data from the checkbox is sent to the server in the form of a list. Since each routes in the server can send this request, we defined a function named **check_list()** to test the vulnerabilities according to the options. Below is the code for this function:

```
def check_list(inputList):
    # test the vulnerabilities according to options
    for i in inputList:
        # do test things
        if i == 'R_XSS':
            print('R_XSS chosen')
        elif i == 'S_XSS':
            print('S_XSS chosen')
        elif i == 'SSTI':
            print('SSTI chosen')
```

Besides, I found a tool that can detect the XSS vulnerabilities automatically based on Python crawler, which can be combined with our application. The basic idea is that we can craft a series of malicious payloads, add them to the normal URL and send them to the server. By catching and analysing the response from the server, we can find out if the route contains some certain vulnerabilities. Below are some examples for the payloads:

```
    // a link says "click me" that can lead to pup-up prompt
    "><a href=javascript:prompt(1)>Clickme</a>

```

```
    // %28 represents "(" while %29 represents ")", thus the JS command is "confirm(1)", which means this is also a link that can lead to pop-up malicious script
    "><a href="javascript:confirm%28 1%29">Clickme</a>

    // the ciphertext encrypted by base64, whose plain text is "<svg/onload=alert(2)>"
    "><a href="data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+">click</a>

    // a text area that can be focused automatically and execute scripts
    "><textarea autofocus onfocus=prompt(1)>

    // a script coded by unicode, whose plain text is "confirm("1")"
    "><a/href=javascript:co\u006efir\u006d("1")>clickme</a>
```

As we can see, these payloads are mainly composed of scripts that can execute pup-up prompt command. Some of them are plain text while some are coded by Unicode, base64 or Url Code.

Here we take the core parts of the program to illustrate how it works. We use multi-thread to crawl the routes from the root URL, and for each route we get, we first use the function **download()** to test if this URL can respond to the GET request. If so, the status code of website should be 200 and then we can store this URL to the list for further actions. Below shows how this is implemented:

```
def download(self, url, htmls):
    if url is None:
        return None
    _str = {}
    _str["url"] = url
    try:
        r = requests.get(url, timeout=10)
        if r.status_code != 200:
            return None
        _str["html"] = r.text
    except Exception as e:
        return None
    htmls.append(_str)
```

After this, we pass the list of URLs to the test section. Below is how this works:

```
for _urlp in urls:
    for _payload in payload:
        _url = _urlp.replace("my_Payload", _payload)
        print("[xss test]:", _url)
        # test
        _str = download.get(_url)
        if _str is None:
            return False
        if _str.find(_payload) != -1:
            print("xss found:%s" % url)
return False
```

As we see, if the URL passes the test, the terminal will output the passed link and if not, it will also show the failed one.

4 Can the project be finished on time? [True/False]

True.

尚需完成的任务 Work to do:

Develop the advanced version of the application that can adapt to more vulnerabilities.
Develop improved MongoDB database for the application.
Continue to improve the server side of the application.

存在问题 Problems:

1. When I tried to exploit some vulnerabilities, the commands that I found didn't work anyway even if I made changes to it.
2. The tools and programs I found from the Internet contains massive errors, which is unusable.

拟采取的办法 Solutions:

1. Some commands are designed for Linux or other Operating Systems instead of Windows that I'm using. As a result, I have to use a virtual machine to try them.
2. The version of Python is changing, but the codes on the Internet not. Many codes that I found were written in Python 2, which is quite different from Python 3 currently. Thus, I have to correct and overwrite the out-of-date codes and change the libraries to make it works.

论文结构 Structure of the final report:

Abstract

1 Introduction

2 Background

2.1 Client-Side Technologies
HTML, CSS and JavaScript

2.2 Server-Side Technologies
Python Flask, Node.js, Django, Jinja2 and the integration of Flask and JS.

2.3 Vulnerabilities in Web Applications
OWASP Top 10, Python Flask, Node.js, MongoDB

3 Design and Implementation

3.1 Client
3.2 Server
3.3 Other Tools Used

4 Results and Discussion
The operations on the client and server and the outcome of them.

5 Conclusion and Further Work
Summarize the whole work, find the problems and solutions in the future.

北京邮电大学 本科毕业设计（论文）教师指导记录表
Project Supervision Log

学院 School	International School	**专业 Programme**	**Internet of Things Engineering**		
姓 Family name	Song	**名 First Name**	Linxuan		
BUPT 学号 BUPT number	2018213147	**QM 学号 QM number**	190018720	**班级 Class**	2018215119
论文题目 Project Title	Improving security of web applications based on mainstream technology				

Please record supervision log using the format below:

Date: dd-mm-yyyy
Supervision type: face-to-face meeting/online meeting/email/other (please specify)
Summary:

Date: 26-10-2021
Supervision type: online meeting
Summary: Discussed the need of the project and the initial version of the draft specification.

Date: 5-11-2021
Supervision type: online meeting
Summary: Discussed the progress and clarified the background knowledge needed.

Date: 15-11-2021
Supervision type: face-to-face meeting
Summary: Discussed the things should be adjusted in the draft specification and the progress.

Date: 23-11-2021
Supervision type: online meeting
Summary: Discussed the progress and the ideas for the basic version of the application.

Date: 1-12-2021
Supervision type: online meeting
Summary: Discussed the progress.

Date: 7-12-2021
Supervision type: online meeting
Summary: Discussed the progress.

Date: 13-12-2021
Supervision type: face-to-face meeting
Summary: Discussed the progress and clarified the early-term and mid-term progress.

Date: 16-12-2021
Supervision type: online meeting
Summary: Discussed the progress.

Date: 22-12-2021
Supervision type: online meeting
Summary: Discussed the progress.

Date: 28-12-2021
Supervision type: online meeting
Summary: Discussed the progress.

Date: 4-1-2022
Supervision type: online meeting
Summary: Discussed the progress.

Date: 11-1-2022
Supervision type: online meeting
Summary: Discussed the progress and the next steps to do.

Date: 28-1-2022
Supervision type: online meeting
Summary: Checked the progress.

Date: 28-1-2022
Supervision type: online meeting
Summary: Discussed next steps to do.

Date: 8-2-2022
Supervision type: online meeting
Summary: Discussed the progress.

Date: 11-2-2022
Supervision type: email
Summary: Checked the progress report and clarified what to do next.

Date: 15-2-2022
Supervision type: online meeting
Summary: Discussed the progress and next steps.

Date: 22-2-2022
Supervision type: online meeting
Summary: Discussed the progress before mid-term and the report.

Date: 28-2-2022
Supervision type: online meeting
Summary: Discussed the mid-term progress.

Date: 3-3-2022

Supervision type: online meeting
Summary: Discussed what to do for the next steps.

Date: 7-3-2022
Supervision type: online meeting
Summary: Discussed the progress

Date: 10-3-2022
Supervision type: online meeting
Summary: Discussed the progress and report.

Date: 10-3-2022
Supervision type: online meeting
Summary: Discussed the progress and report.

Date: 15-3-2022
Supervision type: online meeting
Summary: Discussed the progress and report.

Date: 22-3-2022
Supervision type: online meeting
Summary: Discussed the progress and report.

Date: 24-3-2022
Supervision type: online meeting
Summary: Discussed the progress and report.

Date: 28-3-2022
Supervision type: online meeting
Summary: Discussed the progress and report.

Date: 30-3-2022
Supervision type: online meeting
Summary: Discussed the progress and report.

Date: 8-4-2022
Supervision type: online meeting
Summary: project meeting.

Date: 12-4-2022
Supervision type: online meeting
Summary: project meeting.

Risk and environmental impact assessment

This project studies mainstream web technologies and their related vulnerabilities. The application of this system uses some external APIs and payloads for vulnerability check and performance test. As a result, there may be some risks in terms of program, data, network, etc. Based on this, the assessment table for the risks are shown below.

Description of Risk	Description of Impact	Likelihood Rating	Impact Rating	Preventative Actions
Data Leaked	The user data in MongoDB database got leaked	1-Rare	2-Very Serious	Use advanced encryption method for database
API Not Supported	The external API stops maintenance	3-Moderate	3-Very Serious	Enable automatic update in IDE and search for related functions
Insufficient Memory Storage	The memory storage of the machine run out	2-Unlikely	1-Minor	Clear the disk regularly for free space
The Application Gets Attacked	Attacker or virus on the machine	1-Rare	4-Major	Enable the safety center to keep the machine safe