MINISTRY OF EDUCATION & TRAINING

FPT UNIVERSITY

Capstone Project Document

Optimized Solution for Web Application Scanner

Group		
Group Members	Dương Đình Lộc	HE140882
	Nguyễn Gia Bách	HE141384
	Nguyễn Hoàng Long	HE141049
	Lê Trần Minh Quân	SE61880
	Nguyễn Hoàng Việt	HE140649
Supervisors	Mr. Hoàng Tuấn Anh, Mr. Hà Bách Nam	
Capstone Project code	IAP491_G3	

— Hanoi, December/2021 —

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGE

We would like to express our gratitude to the council for reading our capstone project. We hope you find it informative and useful. We are indebted to a variety of individuals for their efforts. Without the encouragement of several supporters, our capstone project would have taken much longer to finish. It's a pleasure to recognize everyone who has supported us over the last four months.

We must begin by thanking our instructors, Mr. Ha Bach Nam and Mr. Hoang Tuan Anh, for their assistance and advice during the semester. The helpful suggestions have been really beneficial in the development of our idea. Our success would not have been achieved without their help and direction. Due to a lack of expertise, experience, and time, some components may have inadequacies. As a result, we look forward to receiving your comments and further guidance.

Special thanks to the staff of FPT University for their support and guideline throughout our study during the last four years.

Finally, our thanks and appreciations also go to each and every team member for the support, hard work and professional working attitudes.

Many thanks!

ABSTRACTS

In the 4.0 technology revolution, digital transformation is an inevitable development trend, and next to it is the strong development of websites and web applications. Many Web Applications were developed to serve the purpose and provide the necessary services to the users. No systems are perfect as no Web Applications are free from bugs or errors. In order to build a functional Web Application that can survive cyber exploitations and attacks, cybersecurity is considered to be the critical factor.

For Web Applications to always be ready to respond to threats in cyberspace, it is necessary to be able to identify the vulnerabilities fast and accurately. Because of that, the Web Application Vulnerability Scanners are crucial with their ability to test and detect security vulnerabilities.

In our project, we will do deep research on the Web Application Vulnerability Scanners working mechanisms and methodologies using some of the top-ranking scanners. Then, we analyze them to find their remaining weaknesses and provide solutions for them.

1. INTRODUCTION

1.1. Project information

- Project name: Optimized Solution for Web Application Scanner (OSWAS)
- Project name in Vietnamese: Xây dựng giải pháp tối ưu cho phần mềm dò quét lỗ hổng ứng dụng Web
- Timeline: 10/08/2021 - 15/12/2021
- Source code: https://github.com/vtngx/oswas

1.2. People

1.2.1. Supervisors

Full Name	Email	Title
Hoàng Tuấn Anh	AnhHT68@fe.edu.vn	Mentor
Hà Bách Nam	NamHB@fsoft.com.vn	Mentor

1.2.2. Team members

	Full Name	ID	Email	Role
1	Dương Đình Lộc	HE140882	LocDDHE140882@fpt.edu.vn	Leader
2	Nguyễn Hoàng Việt	HE140649	VietNHHE140649@fpt.edu.vn	Member
3	Nguyễn Hoàng Long	HE141049	LongNHHE141049@fpt.edu.vn	Member
4	Nguyễn Gia Bách	HE141384	BachNGHE141384@fpt.edu.vn	Member
5	Lê Trần Minh Quân	SE61880	QuanLTMSE61880@fpt.edu.vn	Member

1.3. Project's background

Web Application crawlers and scanners are two important components in the Web Security world, which are extremely helpful when it comes to detecting vulnerabilities and securing the Web Applications.

Web Application crawlers and scanners come with multiple types; from free to paid, simple to complex. Each tool solves different problems and has its own strengths and weaknesses. Howeverver, the number of Web Crawlers and Web Application Scanners currently on the market is still growing as the web development is improving nonstop. The more the web development world upgrades, the more vulnerabilities are likely to be found.

As future security engineers, we want to contribute to this area by finding the untackled problems of the Web crawlers and scanners and provide optimized solutions for them.

1.4. The initial idea

We found that the world of Web Application Scanners and Crawlers is large but has not been fully developed. It is a fertile ground for further optimization and improvement.

Through our research on web crawlers and vulnerability scanners, we were able to identify several problems that have not been covered in any other projects. In this project, we intend to provide solutions to those problems that can be used in real scenarios.

2. IA PROJECT MANAGEMENT PLAN

2.1. Problem setting

2.1.1. Problem abstraction

2.1.1.1. About Web Application Vulnerability Scanners

Web Application Vulnerability Scanners are automated tools that scan web applications to look for input security vulnerabilities. They search for and report on what known vulnerabilities are present in an organization's IT infrastructure.

These scans can give an organization an idea of what security threats they may be facing by giving insights into potential security weaknesses present in their environment.

A large number of both commercial and open source scanners have their own strengths and weaknesses. In our project, we do research how the top-ranking scanners work and what weaknesses they currently have. We use Acunetix, Netsparker and Burp Suite Professional as our references:

- **Acunetix**

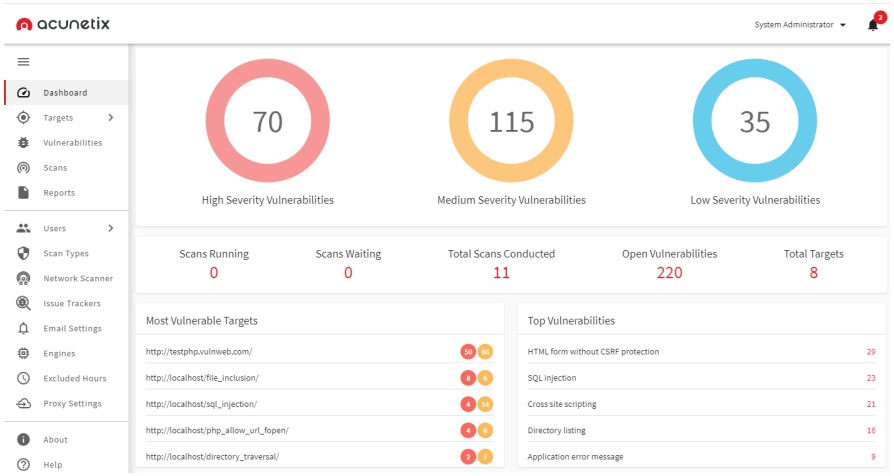


Figure 2.1.1.1a. Acunetix

Acunetix focuses on web application security testing for the most complex environments. Acunetix offers many innovative features, including advanced SQL injection and cross-site scripting (XSS) testing, advanced penetration testing tools, and extensive

reporting. Organizations can use Acunetix's API to connect to other security controls and software developed by third parties.

- **Netsparker**

Figure 2.1.1.1b. Netsparker

Netsparker is a cloud-based, automated vulnerability scanner. It detects and verifies vulnerabilities by exploiting them in a safe and read-only environment. Vulnerabilities are reported only after they are reproduced in a test environment to reduce false positives, saving security professionals significant time. Netsparker also features maintenance scheduling, OWASP top ten protection, database security audit, and asset discovery.

- **Burp Suite Scanner**

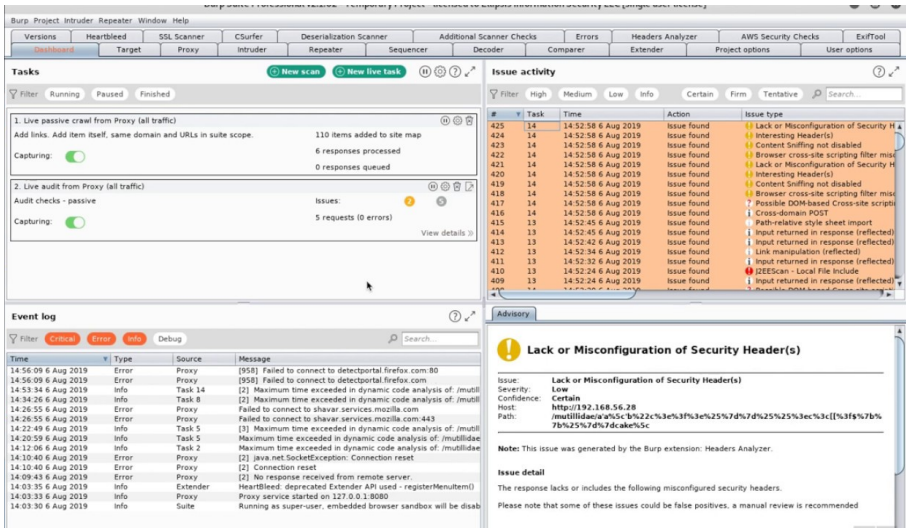


Figure 2.1.1.1c. Burp Suite Scanner

Burp Suite Scanner is a tool for performing automated scans of web sites, to discover content and audit for vulnerabilities. It uses PortSwigger's world-leading research to help its users find a wide range of vulnerabilities in web applications. It works in two main phases: *Crawling* (navigating around the application, following links and submitting forms) and *Auditing for vulnerabilities* (analyzing the application's traffic and behavior to identify security vulnerabilities and other issues).

WebApp Scanners working mechanisms:

Each WebApp Scanner has features and strengths, but in general, they all follow the same steps and mechanisms.

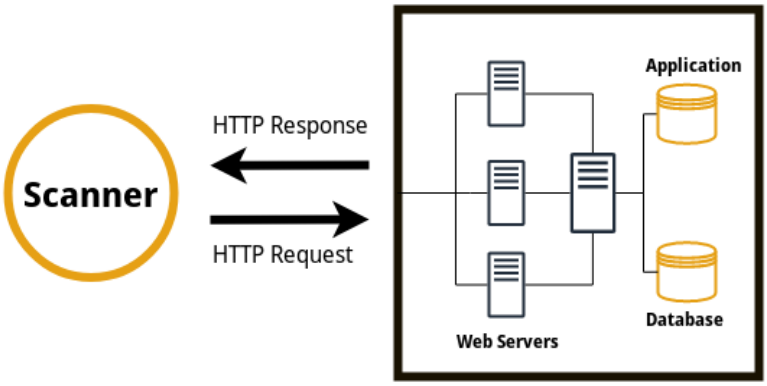


Figure 2.1.1.1d. How a Scanner perform tests

A WebApp Scanner approaches an application as a black-box, which means starting with only an URL of the application. The scanner then traverses through the application and retrieves necessary data (URLs, web traffic, etc.) and builds a site model. Then, it creates attack vectors based on the retrieved data and the selected test policy. Finally, it tests for vulnerabilities within the application by sending modified HTTP requests to the application and examining the HTTP response according to validated rules.

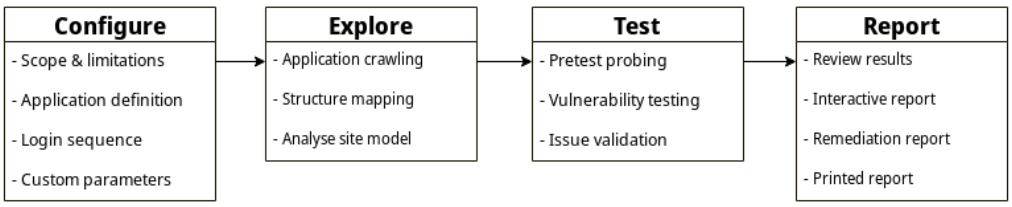


Figure 2.1.1.1e. WebApp Scanner general workflow

A scanner' workflow usually contains four stages:

- **Configure**
  In this stage, the scanner defines the working scope, limitations and the target application. The target Web Application provided in this step, then the scanner will define which areas it should work on by checking robots.txt file of the Web Application (if it exists) and performing the login sequence to check if it can work on the protected areas.
- **Explore**
  The scanner performs crawling on the target Web Application, using the configurations from the previous stage to identify which URL it should analyze, which data it should collect. In this process, the scanner acts as an honest user and goes through every URLs that it can find and satisfies the criterias in the Web Application. It then returns paths, web traffic and data, then creates a sitemap model and prepares data for the vulnerability testing stage.
- **Test**
  This stage is the primary part of the scanner. In this stage, the scanner uses the retrieved data from the previous stage and runs the tests to identify the vulnerabilities of the Web Application. This stage could be manual or automatic in different scanners.
- **Report**
  In this final stage, the scanner exports all the results of crawling and testing to the user. The report can be generated in many different types: interactive report, remediation report or printed report. The user then can review the results.

WebApp Scanners drawbacks and weaknesses:

After doing research on WebApp Scanners methodologies and testing our reference scanners, we can conclude that currently the scanners are well-developed and able to cover most cases, in terms of crawling and testing. However, we were able to find five problems those scanners have not yet fully solved:

- **Unoptimized methods for crawling protected areas**
  We found that the scanners we tested either do not allow user authentication, or simply ask users to provide login credentials before crawling, which does the job but does not cover all authentication methods.

- **Unable to crawl client-side rendering Web Applications**
Most of the scanners crawl new URLs in a web page through the page source, which do not work with client-side rendering Web Applications. This type of Web Application sends the browsers a JavaScript file and the browser does the rendering job, therefore the page source is either not available or simply lines of JavaScript code.

- **Crawl links by submitting forms does not support Vietnamese data**
In order to function globally, most WebApp Scanners are designed for English cases. Many scanners can crawl by submitting forms, but only in English. For Vietnamese Web Applications, the input data needs to be in Vietnamese for the most accurate results.

- **Authentication Bypass Vulnerabilities testing requires manual actions**
Authentication Vulnerability can be tested on some WebApp Scanners, but only manually.
For example, in order to test for this vulnerability in Burp Suite, we need to manually edit the requests, send and analyze the response using *Repeater*.

- **IDOR Vulnerabilities testing requires manual actions**
Insecure Direct Object Reference (IDOR) Vulnerability can be tested on some WebApp Scanners, but only manually.
For example, in order to test for this vulnerability in Burp Suite, we need to manually edit the requests, send and analyze the response using *Repeater*.

2.1.1.2. About Web Crawlers

A Web crawler sometimes called a spider or spider-bot and often shortened to a crawler, is an Internet bot that systematically browses the World Wide Web, typically operated by search engines for Web indexing (web spidering).

Crawlers consume resources on visited systems and often visit sites without approval. Issues of schedule, load, and "politeness" come into play when extensive collections of pages are accessed. Mechanisms exist for public sites not wishing to be crawled to make this known to

the crawling agent. For example, including a robots.txt file can request bots to index only parts of a website or nothing at all.

Internet pages are enormous; even the most significant crawlers fall short of making a complete index. For this reason, search engines struggled to give relevant search results in the early years of the World Wide Web, before 2000. Today, relevant results are provided almost instantly.

Crawlers can validate hyperlinks and HTML code. They can also be used for web scraping and data-driven programming.

The architecture of web crawler

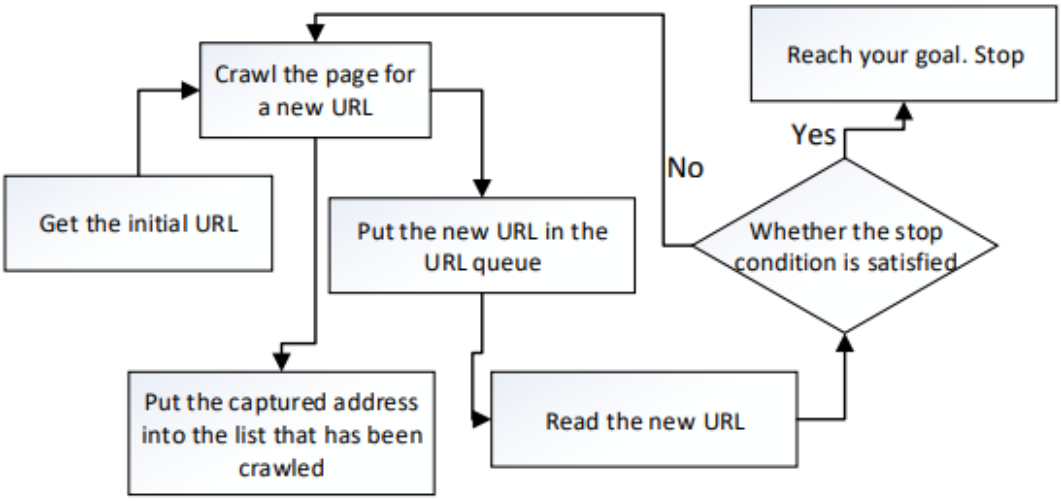


Figure 2.1.2.1. Standard workflow of a crawler

Basic Principle:

Crawlers mainly include downloaders, information extractors, schedulers, and crawl queues:

- Scheduler: seeds URL provided to download.
- Downloader: gets page information from the Internet and sends it to the information extractor, according to the instruction based on the extracting strategy.
- Information Extractor: determines the extracting strategy, receives and extracts the information and the next level URLs.
- Crawl Queue: contains URLs waiting to be called by the scheduler.

Classification of Crawlers:

- Generic Crawler (traditional crawler)

- Grabs all documents and links related.
- It usually runs for a long time and consumes a lot of disk space.

- Focus Crawler (topic crawler)

- Only crawl specific web pages, which can save time, disk space, and network resources.
- The difference between focus crawler & general crawler lies in 2 modules to filter web links: The web page judgment module and the URL priority ranking module.

- **Incremental Crawler**

- Uses a different strategy to update data when crawling pages.
- When it needs to re-crawl the page for updates, instead of traversing through all links in the network, it marks only expired pages. The crawler then only crawls the marked pages.
- Incremental crawling can be used in the following situations:
  - websites with new pages or daily news.
  - Websites where page content will be updated.
  - Determine if the content already exists in the DB.

- Distributed Crawler

- It runs on a group of computers, each of which runs a focused crawler.
- It can be divided into three types:
  - master/slave: is hosted by one machine (master) and controls the operation of the whole group. The host manages the list of URLs to be crawled, issues tasks, and monitors work status for each slave.
  - Autonomous: has no control host; it usually operates through the communication between each machine. There are two forms of communication: circular communication - 1-way transmission through a circle of machine; and Unicom communication - each machine has to communicate with all other machines.
  - Mixed (master/slave + autonomous): has a host in charge of task assignment, but slave machines can communicate with each other and have their task assignment function. The host only assigned a task to the slave if the slave failed its task.

Challenges of web crawler:

- Industrial-scale crawlers require high-load systems with enormous storage and bandwidth, which cost a lot in terms of infrastructure and support. You'll need a fleet of high-end servers to run one, along with a skilled team to implement and maintain it.

- Web crawling is a competitive field between crawlers and anti-crawling measures. Smarter crawlers will spend on processing more relevant content and avoid wasting precious clock cycles and bandwidth.

Challenges of crawling authentication-required (protected) sites:

Crawling protected areas is one of the hardest web crawling tasks. There are countless different authentication systems out there, and in order to make the crawling process completely automatic the crawler needs to support every single one – or else there will be a huge amount of content it won't be able to access.

Furthermore, entering protected areas also means the crawler has to handle sensitive data. It is necessary not to interfere in the functionalities and data of those areas, which is highly difficult for an automatic crawler to be detailed enough to gather all needed data and smart enough to avoid side effects on the website. Blindly following every link and performing every action you have access to can easily destroy data and cause legal action aimed at you and your irresponsible crawler. Several guidelines the crawler has to follow are:

- Not delete anything that users are allowed to delete
- Not try to change user data (usernames, passwords, etc.)
- Not perform content moderation actions (e.g. reporting other users' content)
- Not publish content
- Not dismiss user notifications

2.1.1.3. Insecure direct object references vulnerability (IDOR)

1. **Description**
   Insecure Direct Object Reference (called *IDOR* from here) occurs when an application exposes a reference to an internal implementation object. Using this way, it reveals the real identifier and format/pattern used of the element in the storage backend side. The most common example of it (although is not limited to this one) is a record identifier in a storage system (database, filesystem and so on).

2. **Common Consequences**
   - Vertical Access Control: Vertical escalation of privilege is possible if the user-controlled key is actually a flag that indicates administrator status, allowing the attacker to gain administrative access.
   - Horizontal Access Control: Horizontal escalation of privilege is possible (one user can view/modify information of another user).
   - Context-dependent Access Control: Access control checks for specific user data or functionality can be bypassed.
3. **Tests Objectives**
   - Identify points where object references may occur.
   - Assess the access control measures and test if they're vulnerable to IDOR.
4. **How to test**
   - To test for this vulnerability the tester first needs to map out all locations in the application where user input is used to reference objects directly. For example, locations where user input is used to access a database row, a file, application pages and more. Next the tester should modify the value of the parameter used to reference objects and assess whether it is possible to retrieve objects belonging to other users or otherwise bypass authorization.
   - The best way to test for direct object references would be by having at least two (often more) users to cover different owned objects and functions. For example two users each having access to different objects (such as purchase information, private messages, etc.), and (if relevant) users with different privileges (for example administrator users) to see whether there are direct references to application functionality. By having multiple users the tester saves valuable testing time in guessing different object names as he can attempt to access objects that belong to the other user.
5. **Summary**
   Insecure Direct Object References (IDOR) occurs when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files. Insecure Direct Object References allow attackers to bypass authorization and access resources directly by modifying the value of a parameter

used to directly point to an object. Such resources can be database entries belonging to other users, files in the system, and more. This is caused by the fact that the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks.

6. **Solution**

   - Using secure hashes instead of actual object references is another way to make it harder for attackers to tamper with user-controllable values.
   - Using Indirect Object References that are then internally mapped to actual objects. This means using a per-session reference map populated only with values valid for a specific user and associated with random, non-sequential keys.

2.1.1.4. Common web authentication mechanisms

1. **HTTP basic authentication (HTTP BA)**

   This is the simplest way to enforce access controls on web resources: no cookies, no session identifiers, no login pages, just standard HTTP headers.

   It creates a username and password style authentication for HTTP requests. This technique uses a header called Authorization, with a base64 encoded representation of the username and password. Depending on the use case, HTTP BA can authenticate the user of the application itself.

```
>>> import base64
>>>
>>> auth = "username:password"
>>> auth_bytes = auth.encode('ascii')
>>> auth_bytes
b'username:password'
>>> encoded = base64.b64encode(auth_bytes)
>>> encoded
b'dXNlcm5hbWU6cGFzc3dvcmQ='
>>> base64.b64decode(encoded)
b'username:password'
>>>
>>> "Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=" example.com # Authen header
```

Figure 2.1.2.4a. HTTP BA header encoding

   HTTP Basic Auth is a standardized way to send credentials. The header always looks the same, and the components are easy to

implement. It's easy to use and might be a decent authentication for applications in server-to-server environments.

When using basic authentication for an API, this header is usually sent in every request. The credentials become more or less an API key when used as authentication for the application.

Using basic authentication for authenticating users is usually not recommended since sending the user credentials for every request would be considered bad practice. If HTTP BA is only used for a single request, it still requires the application to collect user credentials. The user has no means of knowing what the app will use them for, and the only way to revoke access is to change the password.

2. **Session-based authentication**

   With session-based auth (or *session cookie auth* or *cookie-based auth*), the user's state is stored on the server's local storage. It does not require the user to provide a username or a password with each request. Instead, after logging in, the server validates the credentials then generates a session, stores it in a session store, and then sends the session ID back to the browser. The browser stores the session ID as a cookie, which gets sent anytime a request is made to the server.

   However, session-based authentication has some drawbacks:
   - Session-based auth is stateful. Each time a client requests the server, the server must locate the session in memory to tie the session ID back to the associated user. Therefore, it doesn't work well for RESTful services as REST is a stateless protocol.
   - Cookies are sent with every request, even if it does not require authentication.
   - Vulnerable to CSRF attacks.

   Flow

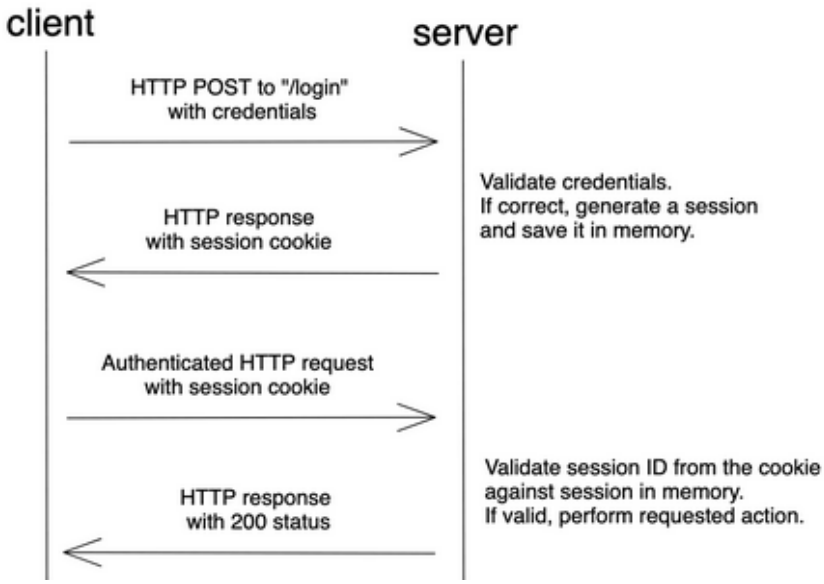


Figure 2.1.2.4b. Session-based authentication flow

3. **Token-based authentication**
   This method uses tokens to authenticate users instead of cookies. The user authenticates using valid credentials and the server returns a signed token. This token can be used for subsequent requests.

   The most commonly used token is a JSON Web Token (JWT). A JWT consists of three parts:

   - *Header* (includes the token type and the hashing algorithm used).
   - *Payload* (includes the claims, which are statements about the subject).
   - *Signature* (verify that the message wasn't changed along the way).

   All three are base64 encoded and concatenated using a dot (.) and hashed. Since they are encoded, anyone can decode and read the message. But only authentic users can produce valid signed tokens. The token is authenticated using the Signature, which is signed with a private key.

   Tokens don't need not be saved on the server-side. They can just be validated using their signatures. Recently, token adoption has increased due to the rise of RESTful APIs and Single Page Applications (SPAs).

Flow

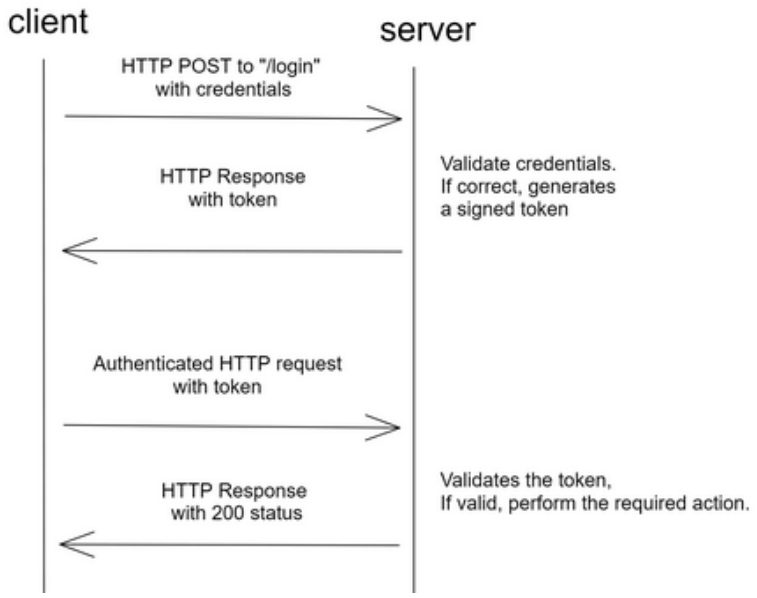


Figure 2.1.2.4c. Token-based authentication flow

4. **OAuth & OpenID**

   OAuth/OAuth2 and OpenID are popular forms of authorization and authentication. They are used to implement social login, which is a form of single sign-on (SSO) using existing information from a social networking service (*Facebook*, *Twitter* or *Google*) to sign in to a third-party website instead of creating a new login account specifically for that website.

   OAuth mechanism is basically an application that asks a user for access to services on behalf of the user and receives a token as proof that the user agreed. This type of authentication and authorization can be used when you need to have highly-secure authentication.

   Flow

   When visiting a website that requires you to log in, you can navigate to the login page and see a button called "*Sign in with Google*". You click the button and it takes you to the Google login page. Once authenticated, you're then redirected back to the website that logs you in automatically. This is an example of using OpenID for authentication. It lets you authenticate using an existing account (via an OpenID provider) without the need to create a new account.

After logging in, you navigate to the download service within the website that lets you download large files directly to Google Drive. This is possible because of OAuth, which allows granting permissions to access resources on another website. In this case, write access to Google Drive.

2.1.1.5. Authentication bypass vulnerability

1. **Description**
   This refers to an attacker gaining access equivalent to an authenticated user without ever going through an authentication procedure. This is usually the result of the attacker using an unexpected access procedure that does not go through the proper checkpoints where authentication should occur.

2. **Test Objectives**
   Black-Box Testing: There are several methods of bypassing the authentication schema that is used by a web application:
   - Direct page request (forced browsing)
   - Parameter modification
   - Session ID prediction
   - SQL injection

3. **Summary**

- In computer security, authentication is the process of attempting to verify the digital identity of the sender of a communication. A typical example of such a process is the log-on process. Testing the authentication schema means understanding how the authentication process works and using that information to circumvent the authentication mechanism.
- While most applications require authentication to gain access to private information or execute tasks, not every authentication method can provide adequate security. Negligence, ignorance, or simple understatement of security threats often result in authentication schemes that can be bypassed by simply skipping the log-in page and directly calling an internal page that is supposed to be accessed only after authentication has been performed.
- In addition, it is often possible to bypass authentication measures by tampering with requests and tricking the application into thinking that the user is already authenticated. This can be

accomplished either by modifying the given URL parameter, manipulating the form, or counterfeiting sessions.
- Problems related to the authentication schema can be found at different stages of the software development life cycle (SDLC), like the design, development, and deployment phases:

- In the design phase, errors can include a wrong definition of application sections to be protected, the choice of not applying robust encryption protocols for securing the transmission of credentials, and many more.
- In the development phase, errors can include the incorrect implementation of input validation functionality or not following the security best practices for the specific language.
- There may be issues during the application setup (installation and configuration activities) in the application deployment phase due to a lack of required technical skills or suitable documentation.

4. **Solution**

- The vendor provides a patched version for the affected and supported products, which should be installed immediately.

- The patch can be downloaded in generating GUI or by calling 'get patches on the command line interface.

2.1.1.6. Problem definition

Due to the research results on Web Application Vulnerability Scanners and Web Crawlers, we were able to identify five problems that most scanners have not solved completely. In this project, we intend to focus on providing solutions for all of those problems:

- Crawl protected areas (authentication-required web pages)
- Crawl client-side rendering Web Applications (E.g. using VueJS)
- Crawl links by submitting forms with Vietnamese data
- Automatically scan for Authentication Bypass Vulnerabilities
- Automatically scan for Insecure Direct Object References Vulnerabilities

Our main focus is to provide solutions to the problems above, not to develop a brand new Vulnerability Scanner. We expected to develop a fully automatic process, starting with crawling a Web Application, then connecting the web HTTP traffic to an external Vulnerability Scanner and testing for vulnerabilities, simultaneously.

2.1.1.7. Problems & solutions

- Crawl protected areas (authentication-required web pages)
The ideal way for this issue is to automate the authentication step before crawling. However, it is almost impossible to identify every web authentication method and create an automatic function for each of them.

We provide a much simpler solution:

Let the user manually authenticate interactively on the browser, then we start to crawl. This is not the perfect solution, but solves every problem.

The user needs to provide an authentication URL so that we can open it on the browser for them. In case the user does not have the authentication URL, we plan to create a functionality to automatically identify every possible authentication URLs for him.

- Crawl client-side rendering Web Applications (E.g. using VueJS)
Common Web Application Scanners can not crawl client-side rendering Web Applications because they work with the page source (E.g. Most Python crawlers use the Beautiful Soup library).

We use another method to crawl, which is through the DOM tree of the web page. Through the DOM tree, the web elements are accessed after being rendered. Selenium testing framework is the perfect choice for this solution. When testing a web page, Selenium opens the web onto the browser and interacts as a real user.

- Crawl links by submitting forms with Vietnamese data
When crawling a web page, we find every HTML form input inside it. Then, we try to fill custom data input them. If we identify a form input that requires a specific pattern (E.g. a phone number input requires Vietnamese phone pattern), we generate a custom input for it based on the pattern.

- Automatically scan for Authentication Bypass Vulnerabilities
For this problem, it is necessary that we retrieve not only URLs in the Web Applications but also the HTTP requests. We also need to crawl the Web Application both with and without authentication.

We then collect all URLs that only access with authentication. We create a script to strip all authentication headers from the requests

and re-send them as non-authenticated. Finally, we analyze the response.

- Automatically scan for Insecure Direct Object References Vulnerabilities

For this problem, it is necessary that we retrieve not only URLs in the Web Applications but also the HTTP requests. We also need to crawl the Web Application using two parallel user accounts and a high-level user account.

To test for horizontal IDOR vulnerability, we create a script to swap all authentication headers from the requests of the parallel users and re-send them. Finally, we analyze the response.

To test for vertical IDOR vulnerability, We then collect all URLs that only admin account access. We create a script to swap all authentication headers from the requests of the admin account with the normal user's and re-send them. Finally, we analyze the response.

2.2. Project organization

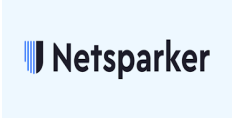
2.2.1. Role and responsibilities

	Name	Team Role	Responsibilities
1	Dương Đình Lộc	• Team Leader • Project Manager • Developer • Tester	- Planning and defining the scope of the project. - Create a schedule, set up the meeting between members to discuss the project, and make sure everyone is on the right direction. - Code, run tests & fix bugs
2	Nguyễn Hoàng Long	• Team Member • Reporter • Developer • Tester	- Code and run tests - In change of the project document - Take notes on meetings

3	Nguyễn Gia Bách	• Team Member • Developer • Tester	- Code, run tests & fix bugs - Research on Scanners and Proxy
4	Lê Trần Minh Quân	• Team Member • Developer • Tester	- Code, run tests & fix bugs - Research on Web App Authentication
5	Nguyễn Hoàng Việt	• Team Member • Technical Leader • Software Architect • Developer • Tester	- Design project workflows - Create coding modules - Assign coding functions to members - Support and review codes - Code, run tests & fix bugs

2.2.2. Tech Stack

Development	**Visual Studio Code** 	IDE for Python 3, HTML, CSS, JavaScript
	Github 	Source code version control
	Linux 	Operating System for development
	MongoDB 	MongoDB, a document database, used to store data (crawler, scanner results).

	Firefox 	Web browser, testing environment
	mitmproxy 	A free and open-source interactive HTTPS proxy.
	Selenium 	Selenium is a Web Testing Framework. We use Selenium as the main part of the crawler.
Reference Software	**Burp Suite Professional** 	Burp Suite is used as an automated Web Application Vulnerability Scanner.
	Acunetix 	Acunetix is an automated web application security testing tool. We use Acunetix as a reference when developing OSWAS
	Netsparker 	Netsparker is a configurable automated web application security scanner. We use Netsparker as a reference when developing OSWAS
Programming languages	**Python 3** 	Python is an interpreted, high-level, general-purpose programming language. We use Python as our main programming language.

	Javascript 	JavaScript is used to develop the Web UI of OWAS.
	HTML 	HTML is used to develop the Web UI of OWAS.
	CSS 	CSS is used to develop the Web UI of OWAS.
Communication	**Telegram** 	Group video calls, chat, file sharing. Contact with Supervisor.
	Google Meet 	Group video calls, contact with Supervisor.
	Facebook Messenger 	Group video calls, chat, file sharing.
	Mobile phone 	Instant calls.
	Gmail 	Share link resources and link between only team Members.

Document, taking note	**Word** 	Writing documents and reports for the project.
	PowerPoint 	Create and design a slideshow for presenting a capstone project.
	Draw.io 	Create and design diagramming and vector graphics for the document.

2.3. Project management plan

2.3.1. Tasks

2.3.1.1. Project initialization, planning

- Description: Learn about Web crawler and how to use Selenium
- Deliverables: Have knowledge about Web crawlers and learn how to use Selenium in Python.
- Resources: Online guides, research papers, etc.

2.3.1.2. Technical studies

- Description: Find and research existing suitable technologies for the project and choose an appropriate programming language and process model.
- Deliverables: Understand the fundamentals of the technology they use, how to use them, the advantages and disadvantages of each technology.
- Resources: Internet, book online, docs online, human resources.

2.3.1.3. Design and analysis project

- Description: Design workflow, database model, logical project model, how each feature works, and interact with users.
- Deliverables: Workflow, database model, logical project model.
- Resources: draw.io, chrome, firefox, human resources.

2.3.1.4. Implementation

- Description: Find and research existing suitable technologies for the project and choose an appropriate programming language and process model.
- Deliverables: Understand the fundamentals of the technology they use, how to use them, the advantages and disadvantages of each technology.
- Resources: Internet, book online, docs online, human resources.

2.3.1.5. Testing

- Description: Testing every function of the solution, does it work wrongly or adequately, searching for as many bugs as possible and fixing them, or minimizing the impact of those bugs if it is hard or impossible to fix.
- Deliverables: Test all functions, fix as many as possible bugs.
- Resources: VS code, Github, Internet, Chrome, Firefox.

2.3.1.6. Deployment

- Description: Deploy testing web application on the browser, wrap up and push source code to Github.
- Deliverables: Collecting data in websites and web applications.
- Resources: Git, Internet, Firefox.

2.3.1.7. Document development

- Description: Detailed presentation of what the project has done.
- Deliverables: A complete docs with no spelling errors.
- Resources: Word, google docs, translator, spell-check extension.

2.3.2. Task schedule sheet: Assignments and timetable

Activity	Detail	Start	End
Start Project	Create group	10/08/2021	10/12/2021
Research	Familiarize with Automation Testing, Selenium, Web Crawlers, Web Vulnerability Scanners. Define problems and research for solutions for the project.	01/09/2021	14/09/2021
Design	Backend module	14/09/2021	16/09/2021

Develop	Backend module	16/09/2021	20/11/2021
Test	Test solution	20/11/2021	30/11/2021
Report	Write a document and slide	20/11/2021	09/12/2021

2.3.3. All meeting minutes

Subject	Time	Location	Supervisor	Attendees	Secretary	Meeting Topic
First Meeting	12/08	Google Meet	NamHB, AnhHT	All group members	No	Team members familiarize with each other. Research project's topic and its solutions.
Weekly Meeting	20/08	Telegram	NamHB, AnhHT	All group members	VietNH	Design workflow. Assign tasks. Research about crawlers.
	18/09	Google Meet	NamHB, AnhHT	All group members	VietNH	Review problems need researching and checklists. Assign tasks.
	25/09	Google Meet	NamHB, AnhHT	All group members	VietNH	Review tasks. Discuss techniques. Assign tasks.
	16/10	Google Meet	NamHB, AnhHT	All group members	VietNH	Review tasks. Discuss techniques. Assign tasks.
	30/10	Google Meet	NamHB, AnhHT	All group members	VietNH	Review tasks. Discuss techniques. Assign tasks.

	13/11	Google Meet	NamHB, AnhHT	All group members	VietNH	Review tasks. Discuss techniques. Assign tasks.
	27/11	Google Meet	NamHB, AnhHT	All group members	VietNH	Review tasks. Assign new tasks. Discuss the situation and solution. Review the development progress.
	04/12	Google Meet	NamHB, AnhHT	All group members	VietNH	Review tasks. Discuss the situation and solution. Review project documents.
	11/12	Google Meet	NamHB, AnhHT	All group members	VietNH	Review codes, documents and slides.
	18/12	Google Meet	NamHB, AnhHT	All group members	VietNH	Rehearse presentation

3. Risk Assessment

3.1. The need for risk assessment

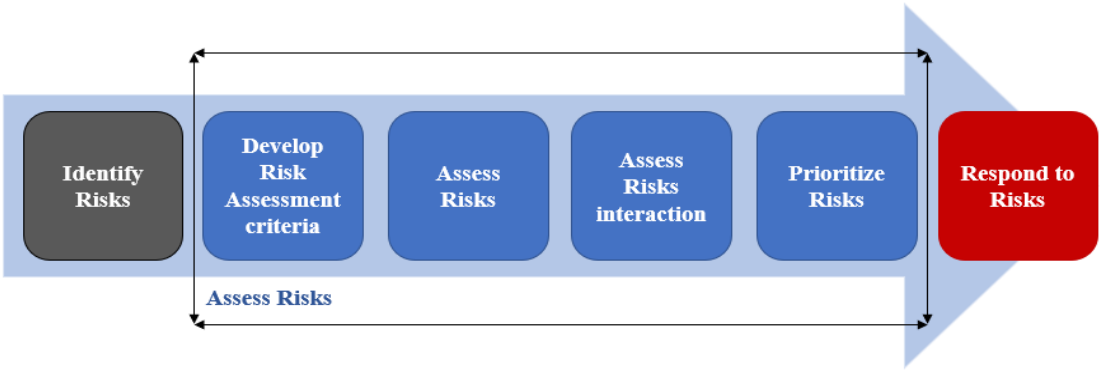


Figure 3.1. Risk assessment process

Risk Assessment is one tool for Business Impact Analysis. It is used to identify potential hazards, analyze them, and project the outcome if the risk or threat occurs. In short, we can also say that this process can be used to determine the potential impact resulting from the decision taken.

The Risk Assessment process is used to assess how significant the risks are, both individually and collectively, to focus on the most critical threat and opportunities and to lay out a procedure to respond to such threats and opportunities.

Process of Risk Assessment following the steps:

Step 1: Identify Risk

First things first, we need to see the risk associated with the decision to produce a comprehensive list of the risks related. Typically, we should also categorize the risk into Financial, Operational, Strategic, and Compliance.

At this stage, we understand the risks. While each risk captured is essential to the management at the functional or business unit level, the list requires prioritization to focus senior management and board attention on key risks. This prioritization is accomplished by performing the risk assessment.

Step 2: Develop Risk Assessment Criteria

This is where we start the risk assessment process, which aims at a risk assessment strategy. Here we develop a common set of risk assessment criteria. The risk is generally assessed in terms of the impact and likelihood.

Impact

Impact assessment criteria may include financial, reputational, regulatory, health, safety, security, environmental, employee, customer, and operational impacts.

Rating	Category	Definition
1	Insignificant	Negligible impact on project
2	Tolerable	It will only affect the project in a minor way
3	Undesirable	Will cause significant damage to the project
4	Major	Severe impact on project
5	Fatal	Catastrophic to the project's survival

Likelihood

The likelihood is the possibility that a given event will occur.

Likelihood can be expressed using quantitative terms such as:

Rating	Category	Definition
1	Very Low	Risk is improbable to occur
2	Low	Risk is unlikely to occur
3	Medium	There is a possible chance that the risk will occur
4	High	The risk will likely occur
5	Very High	It is almost a certainty that the risk will occur

Step 3: Assess Risks

This phase consists of assigning values to each risk and opportunity using the criteria defined earlier. Risks can be assessed using both qualitative & quantitative techniques. Such used methods can anyone of the following mentioned techniques but are not limited to these:

- Analysis of Existing Data
- Analysis of Historical Data
- Interviews and Cross-Functional Workshops
- Surveys
- Benchmarking
- Scenario Analysis
- Causal At-Risk Models

Step 4: Assess Risk Interactions

Usually, risks don't exist in isolation. Organizations recognize the importance of managing risk interactions. Individual risks have their potential, but when they interact with other events and conditions, they cause great damage or create significant opportunities.

Step 5: Prioritize Risk

Risk prioritization is the process of determining risk management priorities by comparing the level of risk with all the other potential risks and tolerance levels.

Step 6: Respond to the Risks

The results of the risk assessment process are used as a primary input to management's response to risks accordingly.

- The usual responses may be one of the following:
- Accept the risk as inherent
- Reduce the risk by placing control activities.
- Share with other projects, parties, departments, functions, etc.
- Avoid the risk if it is fatal for the company.

We perform a cost-benefit analysis at this stage, a response strategy is formulated, and risk response plans are developed.

3.2. Identify critical information assets.

3.2.1. Information asset classification

All information assets must be classified according to their level of sensitivity as follows:

Type	Description	Risk Level
Published	Low-sensitive information. Information that is not protected from disclosure, that if disclosed will not jeopardize the privacy or security of agency employees, clients, and partners. This includes information regularly made available to the public through electronic, verbal, or hardcopy media.	Low

Limited Use	Sensitive information that may not be protected from public disclosure but, if made easily and readily available, may jeopardize the privacy or security of agency employees, clients, partners. Must follow its disclosure policies before providing this information to external parties.	Medium
Restricted	Sensitive information intended for limited business use may be exempt from public disclosure because, among other reasons, such disclosure will jeopardize the privacy or security of agency employees, clients, partners, or individuals who otherwise qualify for an exemption. Information in this category may be accessed and used by internal parties only when specifically authorized to do so in the performance of their duties. External parties requesting this information for authorized agency business must be under a contractual obligation of confidentiality with the agency before receiving it.	High
Critical	Information that is deemed extremely sensitive and is intended for use by named individual(s) only. This information is typically exempt from public disclosure because, among other reasons, such disclosure would potentially cause significant damage or injury, including death to the named individual(s), agency employees, clients, partners, or cause substantial harm to the agency.	Critical

3.2.2. System Characterization

3.2.2.1. System Component

Hardware Component:

Team members	Item	Note
Dương Đình Lộc	Laptop	Development/test environment
Nguyễn Gia Bách	Laptop	Development/test environment
Nguyễn Hoàng Việt	Laptop, PC	Development/test environment
Nguyễn Hoàng Long	Laptop	Development/test environment
Lê Trần Minh Quân	Laptop, PC	Development/test environment

Software component:

Item	Description
Database	Use MongoDB to store data
Burp Suite Professional	Capturing HTTP requests and using it to automatically scan websites for security vulnerabilities.
Source-code	Build solution for scanner

3.2.2.2. Users of the system

Developer:

- Code back-end, front-end, Database connection, agent modules.
- Fix bugs, handle errors and update functions.

Tester:

- Testing the solution's features and reporting any issues to the developer for correction.

User:

- Use the solution to scan vulnerabilities on websites or web applications.

3.2.2.3. Security and compliance requirements

To ensure system security, all members must have complied with the following security requirements:

- Do not install unidentified software on the laptops, workstations.
- Regular update, check version of antivirus software.
- Protect source code, model components of the system.

3.2.2.4. Information protection priorities

No	Information Assets	Priority
1	System Configuration	High
2	Source Code	High
3	Database	High
4	Third-party and Open-source tool Configuration	High
5	Scanned Website	Medium
6	Manual of system	Low

3.3. Risk identification

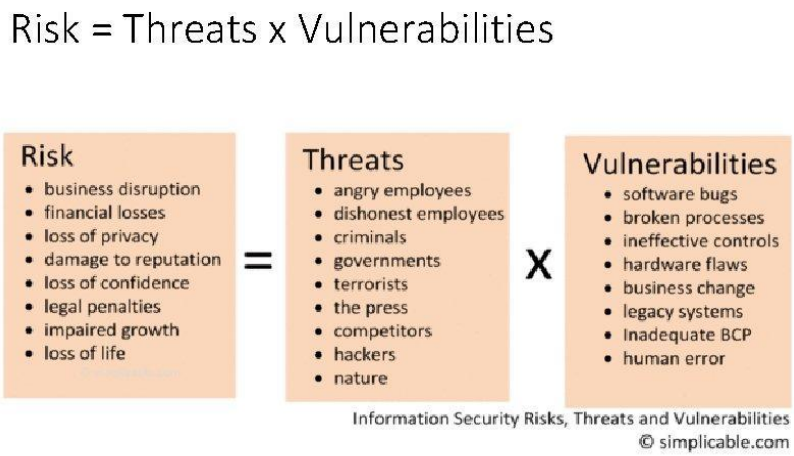


Figure 3.3. A basic formula for describing risk

However, that is not a legitimate mathematical formula in the usual sense. Threats and vulnerabilities do not necessarily have numerical quantities associated with them. Instead, the connection between the two is represented by the formula. If we are able to determine the value of the asset, the formula is slightly adjusted is:
Total Risk = Threat * Vulnerability * Asset Value

3.3.1. Threat identification

Threats are classified into two types: human and environmental. Human threats may either be internal (e.g., irate employees, unethical employees) or external (e.g., terrorists, hackers, competitors).

Environmental threats, such as power outages and natural catastrophes, might cause the system to fail.

Type of threat	Reason
Environmental	- Natural disasters: earthquakes, floods or hurricanes. - Electrical problems, fires. - Long-term power failure. - Degraded equipment. - Poor network connection.

Internal	Employees who are dissatisfied with their jobs or the company Records can be collected and deleted through the portal and by the administrator. Change the settings or shut down the service to make the whole organization scan. If an agent service is shut down or misconfigured, the entire organization will be scanned. They have the ability to gather information on a company or organization.
External	The hacker who wants to attack for information collection or privilege escalation.

3.3.2. Vulnerability identification

Vulnerability	Description
NoSQL Injection	A malicious NoSQL query/processing can be executed because the parameter input is inserted into the query statement to the NoSQL database
Insufficient logging and monitoring	Missing security critical information logs or lack of proper log format, context, storage, security and timely response to detect an incident or breach.
Sensitive data exposure	Can occur when a web application does not adequately protect sensitive information from being disclosed to attackers. This can include information such as credit card data, medical history, session tokens, or other authentication credentials.

3.4. Risk analysis

Risk analysis is the process of determining the probability of an undesirable event happening in the private, public, or environmental sectors. Risk analysis is the examination of the inherent uncertainty associated with a particular course of action. It encompasses the uncertainty associated with forecasted cash flow streams, the probability of a project's success or failure, and possible future institutional states.

3.4.1. Likelihood assessment

When a successful assault on a system's vulnerabilities occurs, it is critical to conduct an evaluation that identifies all the associated losses and damage. The following table summarizes the assets that might be affected by a security system failure.

Impact	Low	Medium	High
Data loss	No loss	System internal information leaked.	System internal information leaked.
Trust and reputation damage	Reduce the amount of trust of users.	Users may lose trust in the business and come to see it negatively.	Users may fully lose trust in the business and take legal action against the company.
Misconfiguration	Reducing the efficiency of the system.	Wrong output results. Data being modified.	Create serious system vulnerabilities.

3.4.2. Impact assessment

The likelihood assessment will determine the likelihood that a suspected vulnerability will be investigated, on a scale of "Rare" to "Almost certain."

Likelihood	Rare	Unlikely	Possible	Likely	Almost certain
Data loss			✔		
Trust and reputation damage				✔	
Misconfiguration				✔	

3.4.3. Risk determination (Rating)

3.4.3.1. Risk-level Matrix

	Likelihood				
Impact	**Rare**	**Unlikely**	**Possible**	**Likely**	**Almost certain**
Low	Low	Low	Moderate	Moderate	Moderate
Medium	Low	Low	Moderate	High	High
High	Moderate	Moderate	Moderate	High	High

Vulnerability	Likelihood	Impact	Risk Rating
NoSQL Injection	Possible	High	Moderate
Unrestricted File Upload	Possible	High	Moderate
Sensitive data exposure	Possible	High	Moderate
Insufficient logging and monitoring	Possible	High	Moderate

3.4.3.2. Description of Risk Level

Risk Level	Risk Description
High	- System processes and/or stores confidential or restricted data - System is highly trusted by UI networked systems - System provides a critical or campus-wide service
Moderate	- System processes and/or stores non-public or internal-use data - System is internally trusted by other networked systems - System provides a normal or important service
Low	- System processes and/or stores public data - System is easily recoverable and reproducible - System provides an informational / non-critical service

3.5. Control identification and assessment

3.5.1. Risk monitoring and controlling

We must monitor and control risk in order to:

- Ensure the execution of risk management policies and evaluate their efficacy.
- Risks that have been recognized must be monitored.
- Monitoring leftover hazards and identifying new risks that may arise during project execution.

3.5.1.1. Input to risk monitoring and control

Risk Monitoring and Control requires the following:

- Risk management plans.
- Risk identification provides recognized risks and owners, risk responses, triggers, and warning signals.

3.5.1.2. Output from risk monitoring and control

- Risks, causes, and consequences of actual responses are being updated.
- Corrective action involves putting contingency plans or alternatives in place.
- Recommended Preventive Actions are used to guide the project and ensure that it follows the project management plans.
- Assets of the Organizational Process Updates include information gathered via risk management, procedures collected and saved for future use, templates for the risk management strategy, and lessons learned.
- Project Management Plan Updates to project management plans as a consequence of requested revisions being approved.

3.5.2. Preventive measures

For technological safeguards to protect the system, the system's primary risk element is humans. Here are some precautions against risks posed by internal and external individuals.

Human Threat	Object	Control Methods
Internal	Irate employees, unethical employees	Permission to use in service-oriented systems, Authentication, Authorization, and Access control.
External	Terrorists, competitors, hackers	Using IDS/IPS, FIM for intrusion detection and system integrity checking.

For threats from the system, we do frequent backups and utilize the program to verify that no active services are available.
Keeping track of erroneous system events and resolving them as quickly as feasible.

4. Risk Management Plan

A risk management plan, also known as a "risk mitigation plan," is a well-defined document that specifies how to deal with specific risks and what management measures must be taken against those risks in order to minimize or eliminate threats to the project's tasks and outcomes.
Risk management plans provide teams with an understanding of the steps they must take in order to identify, assess, and respond to all of the risks that may arise throughout the course of a project.
When you're planning the project, risks are still uncertain: they haven't happened yet. But eventually, some of the risks that you plan for do happen, and that's when you have to deal with them. There are four basic ways to handle a risk:

- **Avoid**: The best thing to do with risk is to avoid it! If you can easily prevent it from occurring, it definitely won't hamper your project.
- **Mitigate**: Now, if you can't avoid risk, you sure can mitigate it. This means taking some action against it that will do as little damage to your project as possible.
- **Transfer**: Another great way to deal with a risk is by transferring it to a third party. The best and most common process to do this is to buy insurance.
- **Accept**: Finally, if nothing works then you have to accept it. By accepting the risks, you can work in the direction to treat them in advance.

4.1. Objectives of RMP

4.1.1. List of threats & vulnerabilities

Component	Threats/Vulnerabilities
Database	NoSQL Injection
Application	Insufficient logging and monitoring
	Sensitive data exposure

4.1.2. List of recommendations to reduce the risks

4.1.2.1. Technical

- **NoSQL Injection**

- Use a sanitization library. For example, mongo-sanitize or mongoose.
- If you can't find a library for your environment, cast user input to the expected type. For example, cast usernames and passwords to strings.
- In the case of MongoDB, never use where, mapReduce, or group operators with user input because these operators allow the attacker to inject JavaScript and are therefore much more dangerous than others. For extra safety, set javascriptEnabled to false in mongod.conf, if possible.
- Additionally, always use the least-privilege model: run your application with the lowest privileges possible so that even if it gets exploited, the attacker cannot access other resources.

❖ **Insufficient logging and monitoring**

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that can be easily consumed by centralized log management solutions.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.
- Establish or adopt an incident response and recovery plan.

❖ **Sensitive Data Exposure**

- Classify data processed, stored or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.
- Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization

by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for responses that contain sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.
- Verify independently the effectiveness of configuration and settings.

4.1.2.2. Non-Technical

Training all members on risk awareness ensures that these risks are avoided.

4.2. Describing procedures for accomplishment

- ✔ Create a backup for the server and database.
- ✔ Regularly update operating system and software.
- ✔ Update firewall and policy.
- ✔ Create a business continuity plan, recovery plan.

4.3. Reporting requirements

4.3.1. Present recommendations

In this section, we identify threats and vulnerabilities and offer countermeasures.
The following information should be included in this section:
System and Application risk
Cause: Human threats: Hackers, users.
Criteria: The criteria that will allow the threat to be effective.

- Source code flaws
- Source code flaws can lead to SQL injection, disclosure of information
- Outdated operating system: The system will have lots of vulnerabilities when they have not been patched.
- Antivirus software not installed and updated: The antivirus must be installed and updated to detect new malware.
- Misconfiguration: Misconfiguration can lead to some system vulnerabilities.

Effect:

- The system may not normally be operated and could even stop running.
- Loss of confidentiality, integrity, or availability.

Response recommendations:

- Install antivirus software and keep it updated.
- Keep the system updated.
- Reconfigure the system.
- Audit source code to find security flaws.

4.3.2. Document management response to recommendations

All members will consider what to do after all the recommendations are given.
Recommendations can be accepted, deferred, or modified.

- **Accept:** Recommendations are approved. They will be added to the Plan of Action and Milestones for tracking.
- **Defer:** Recommendations can also be deferred. They might still be considered to be implemented at a later time depending on a specific situation.
- **Modify:** Recommendations will be modified to best fit the system.

Recommendation	Accept	Defer	Modify
Training all members about risk awareness.	✔		
Update antivirus and all other software.			✔
Create a backup, recovery plan.			✔
Use strong passwords for databases.	✔		
Audit source code to find security flaws.	✔		
Keep the system updated.			✔

4.3.3. Document and track implementation of accepted recommendations

- **Training all members about risk awareness.**
  The training will be finished as soon as feasible.
- **Backup system configuration and database.**
  Important files should be backed up frequently, once a week or right before and after a big change in the system.
- **Use strong passwords for databases.**
  Passwords must include:
  - Character type: Uppercase, lowercase alphanumeric characters, special characters.
  - Character length: 8 or more characters
- **Reconfiguring the system.**
  Check the system to enable necessary services and disable unused ones.
- **Audit source code to find security flaws.**
  Process after a module is finished coding.

5. Specifications, Development & Implementation

5.1. Data flow diagram (DFD) of our system

The vertical diagram is used to create our system data flow diagram. We selected the vertical diagram because it is simple to draw and illustrate the system's operation, making it much easier to comprehend for the reader. Modules are operated sequentially from top to bottom.

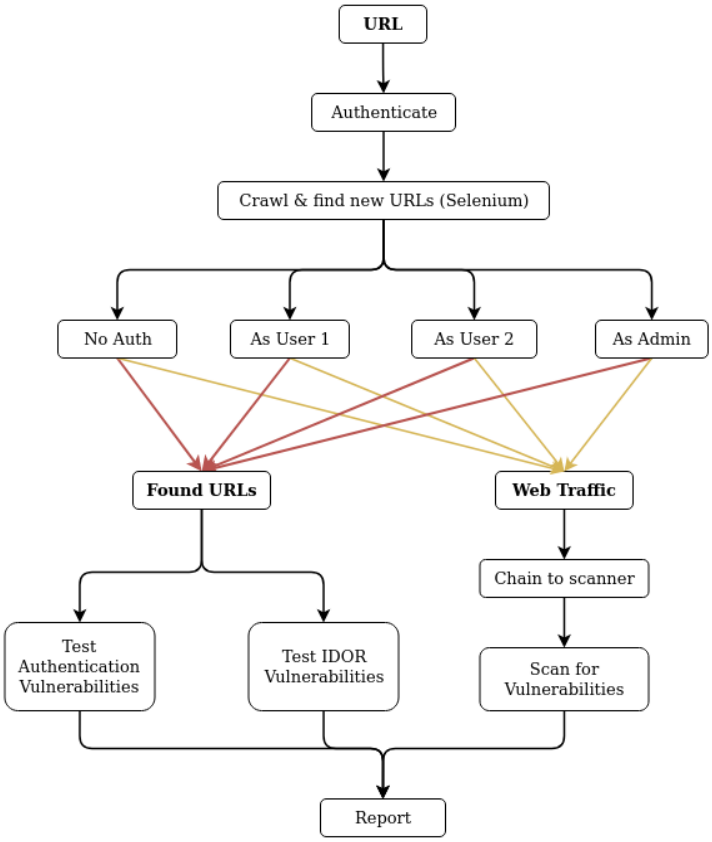


Figure 5.2. Data flow diagram

First, determine the URL needed to crawl and check if this is the URL to log in. Next, start crawling data and find new URLs from the original URL. This step divides into 4 cases to implement: no account, user1, user2, admin. Crawl each case and test two vulnerabilities: IDOR and bypass authentication. Simultaneously, sending web traffic of each case to the scanner to perform the task of scanning for vulnerabilities. Finally, the results of the two sections are aggregated and reported.

5.2. Logical model of the project

5.2.1. Burp Suite Professional

Burp Suite is one of the most popular penetration testing and vulnerability finder tools and is often used for checking web application security. “Burp”, as it is commonly known, is a proxy-based tool used to evaluate the security of web-based applications and do hands-on testing. It has a robust and modular framework and is packed with optional extensions that can increase web application testing efficiency. In the project, Burp suite is used for capturing HTTP requests and also using it to automatically scan websites for security vulnerabilities.

5.2.2. Database Design

OSWAS requires a database in order to store the target URLs, crawled URLs and details of those URLs. We identified that the requirements for the database are query performance, write performance and easy scalability in order to handle querying hundreds or thousands URLs. Therefore, we chose MongoDB (a NoSQL document-oriented database) for the software in order to fulfill the requirements according to its advantages in write performance and scalability.

The database is designed with two main components: target URLs and crawled URLs. We create a collection for each component: *Target* (for target URLs) and *Link* (for crawled URLs). In each collection, besides the URL, we also store some other details of the URL. Each document in the *Target* collection represents each time a target URL is crawled; each document in the *Link* collection represents an URL that is found when crawling a Target and each *Target* document contains multiple *Link* documents, which is connected through property *target_id*.

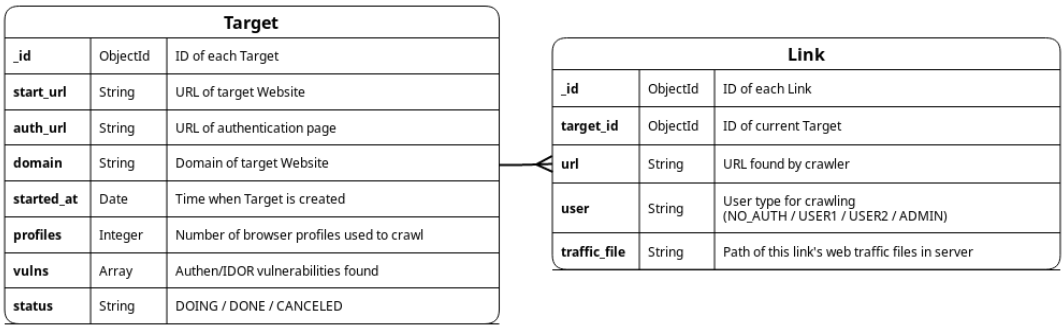


Figure 5.3.2. Database design

Target collection contains:

- **_id**: ObjectID of the Target
- **start_url**: The URL of the Website/Web-App using to crawl
- **auth_url**: The authentication URL of the Target
- **domain**: The domain of the Target
- **started_at**: The time when the Target is started crawling
- **profiles**: Number of user profiles used to crawl
- **vulns**: Authentication/IDOR vulnerabilities that found by our software
- **status**: Crawling status of the Target

Link collection contains:

- **_id**: ObjectID of the Link
- **target_id**: ObjectID of the Target that this Link is belongs to
- **url**: The URL of a new link found in a Target
- **user**: The user profile (scope) that the Target is crawling in
- **traffic_file**: Path of this Link's web traffic files that are saved on the server

5.2.3. Test Environment

OSWAS is combined between a Web Crawler and a Web Scanner, therefore the testing environment should be a Web Application. As mentioned above, one of our main goals is to provide a solution for crawling a client-side rendering Web Application. Therefore we use VueJS for developing the Frontend.

Based on the structure of the university's CMS, we built a university management web application, hosted by *Heroku* (a free deploying and hosting service) at domain *http://not-fap.herokuapp.com/*.

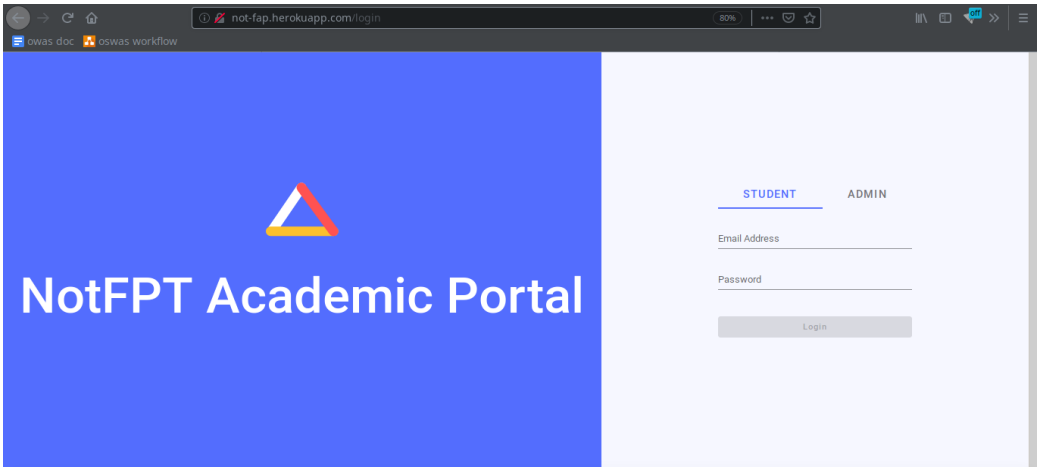


Figure 5.3.3a. Testing site default page

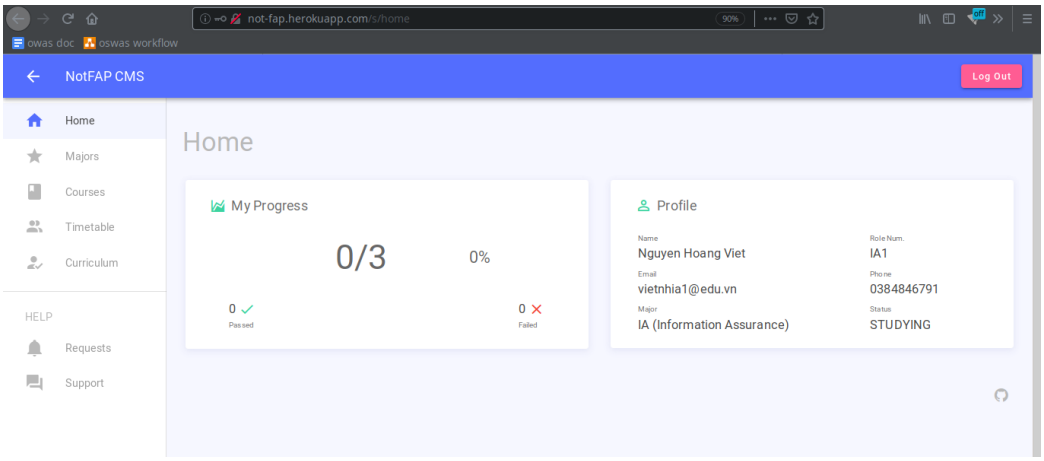


Figure 5.3.3b. Testing site dashboard (student account)

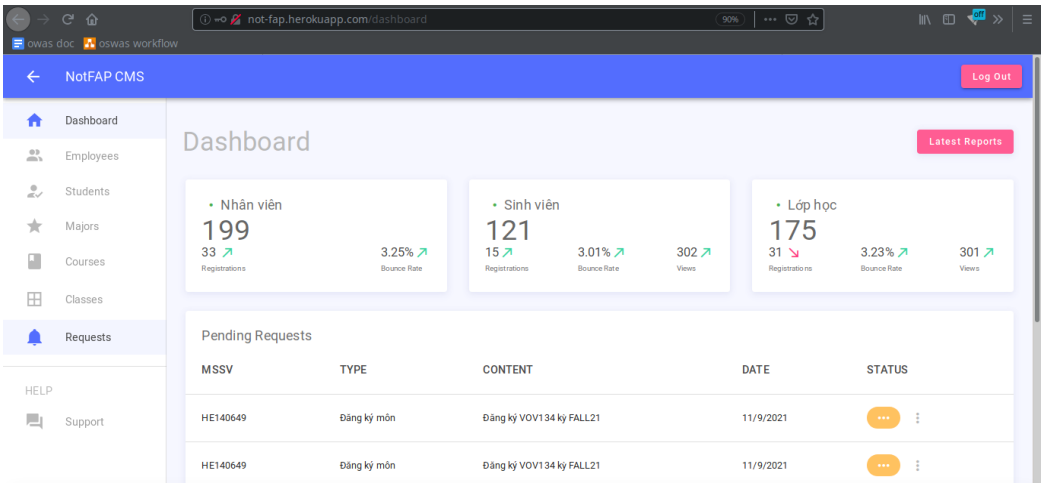


Figure 5.3.3c. Testing site dashboard (admin account)

This web application simulates the most basic functionalities of a university management site:

- User authentication
- Admin/student authorization
- Manage students
- Manage teachers
- Manage university majors
- Manage university courses

In the development process, we will use this web application to test both the crawling and the vulnerability scanning functionalities.

5.3. Building functional modules

5.3.1. Python

- Python is the language of the Python Interpreter and those who can converse with it. An individual who can speak **Python** is known as a **Pythonista**. It is a very uncommon skill, and may be hereditary. Nearly all known Pythonistas use software initially developed by ***Guido van Rossum***.

- There are many advantages of Python:

- Presence of third-party modules.
- Extensive support libraries (NumPy for numerical calculations, Pandas for data analytics etc).
- Open source and community development.
- Versatile, Easy to read, learn and write.
- User-friendly data structures.
- High-level language.
- Dynamically typed language (No need to mention data type based on the value assigned, it takes data type).
- Object-oriented language.
- Highly Efficient (Python's clean object-oriented design provides enhanced process control, and the language is equipped with excellent text processing and integration capabilities, as well as its own unit testing framework, which makes it more efficient.).

- Why do we use Python for crawlers ?

- Web Crawler is a software/tool that is used to crawl any website to extract the data/link.

- Python is mostly known as the best web scraper language. It's more like an all-rounder and can handle most of the web crawling related processes smoothly.
- Beautiful Soup is one of the most widely used frameworks based on Python that makes scraping using this language such an easy route to take. Beautiful soup is a Python library that's designed for a fast and highly efficient web scraper. Some of the notable features are Pythonic idioms for navigation, searching, and modifying a parse tree. Beautiful Soup can also convert incoming documents to Unicode and outgoing documents to UTF-8. Beautiful Soup works on popular Python parsers like lxml and html5lib, which allow you to try different parsing methodologies.
- In our project, we mainly use Selenium to access web elements through DOM trees besides using the web source, in order to maximize the amount of data we can collect. For this reason, Python offers a lot of advantages: Selenium helps the API used in Python connect to the browser, the junction of Python and Selenium offers an easy API to write functional tests by utilizing Selenium WebDriver in a perceptive way and the libraries of Python are rich standards that help the testers in writing efficient automation scripts for the test automation suite.

5.3.2. Selenium

Selenium is an open-source testing tool that allows users to test web applications across different browsers and platforms. Selenium includes a suite of software that developers can use to automate web applications including IDE, RC, WebDriver and Selenium grid, which all serve different purposes.

The Beginning - Selenium Integrated Development Environment (IDE)

Selenium IDE (Integrated Development Environment) is the Firefox plug-in tool that allows you to record, playback, edit and debug test scripts to export them in a preferred programming language. By providing a test domain specific language, Selenese, IDE lets you test without learning a new scripting language and makes it easier to send commands to the browser and automate testing.

The Evolution - Selenium Remote Control (RC) and WebDriver

Selenium RC (Remote Control)

Selenium RC was a method used to run the test script in different browsers by launching them as a proxy server, but has since been abandoned since the creation of WebDriver. Selenium WebDriver is what you would use in testing today.

WebDriver directly communicates with the browser from the operating system making it faster and more accurate and also supports more browsers and operating environments than RC including the HtmlUnit browser, and has a more concise API.

Optimization – Selenium Grid

Selenium Grid allows you to run multiple tests at the same time on different machines and browsers in order to speed up testing. For CI motions, testing demands to be done at a rapid pace, and running tests sequentially can be painfully slow. Running in parallel on a grid can improve the speed of your testing greatly.

The main component of Selenium we use is Selenium WebDriver.

5.3.3. Crawler Algorithm

In our solution, we use the Tree traversal algorithm. Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. The generally used ways for traversing trees are: Depth First Traversal (Inorder, Preorder, Postorder) and Breadth-First or Level Order Traversal. Here we specifically chose Breadth-First Traversal to resolve our problem.

Breadth-first search (**BFS**) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

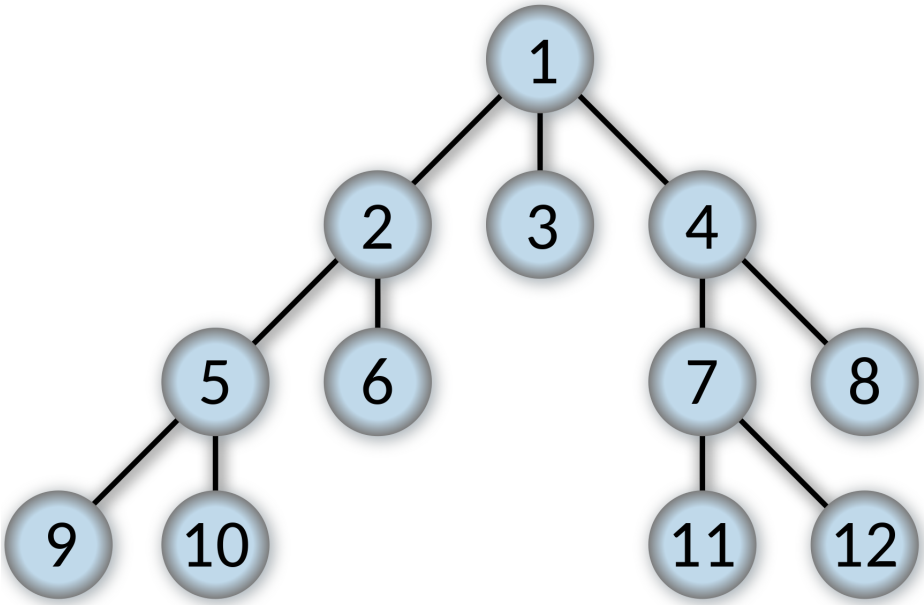


Figure 5.4.3. Breadth-first Tree Traversal

For example, in a chess endgame a chess engine may build the game tree from the current position by applying all possible moves, and use breadth-first search to find a win position for white. Implicit trees (such as game trees or other problem-solving trees) may be of infinite size; breadth-first search is guaranteed to find a solution node if one exists.

In contrast, (plain) depth-first search, which explores the node branch as far as possible before backtracking and expanding other nodes, may get lost in an infinite branch and never make it to the solution node.

5.4. Development modules

5.4.1. Module I - Authentication sequence optimization

Module I is the first phase of the application. The main purpose of this module is to configure the crawler with the URL of a WebsiteWeb Application to perform crawling and its authentication page. We intend to focus on optimizing the identification of the authentication page, which is necessary to perform crawling authentication-required pages, by using multiple methods on it: input by user, path scanning by *dirsearch* and identifying by signatures.

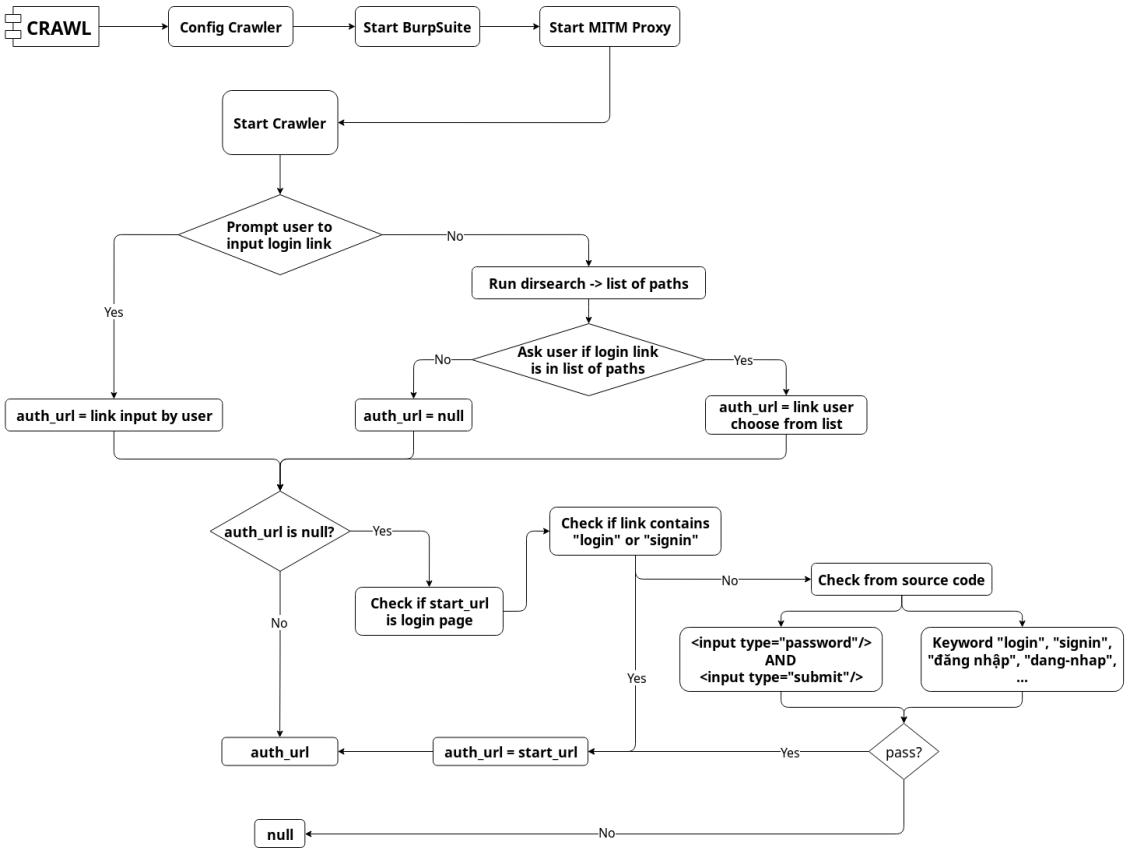


Figure 5.5.1. Workflow of Module I

Configuration: The first step is to start BurpSuite, configure the default port on the *proxy* tab to 8888 with all interfaces, then start MITM Proxy. Prompt user to input URL of the Website/Web Application used to crawl. Create a new *Target* in the database, set the URL that will be used to perform crawling to ***start_url***; set the *null* default value to ***auth_url***.

Start Crawler

Step 1: After setting up the configurations, the system will ask the user to input the URL of the authentication page. In this step, there would be 2 cases. If the user has the authentication page URL, we set the URL inputted by the user to the field *auth_url* of the Target. If the user does not have the authentication page URL, the system will start running tool *dirsearch* (a web path scanner tool) to find URLs that could be the authentication page URL. The user will then be shown a list of links found by *dirsearch*. The user can select a link in the list as the authentication page, the selected link is then set as *auth_url* of the Target. Otherwise, if the user does not select any link in the list, *auth_url* is set to *null*.

Step 2: If the field *auth_url* is not *null* (meaning user has inputted it or selected from the *dirsearch* list), returns *auth_url*. Otherwise, we check if the *start_url* is either the login page or contains the link of the authentication page. Firstly, we check if the link contains the word "*login*" or "*signin*". If the check fails, we will use the source code of the *start_url* page to check for signatures of the authentication pages. In authentication pages, the source code usually contains a form where users can fill in the username and password (input type=”password”) input fields and the submit button (*input type=”submit”* or *button type=”submit”*) or includes keywords like "*login*", "*signin*" or "*dangnhap*". We check if the source code of the *start_url* page contains any of those keywords. If any of those conditions is satisfied, we set *start_url* as *auth_url* and return it; otherwise, we conclude that this website does not have a login link and set *auth_url* as *null*.

Source code:

Function for starting BurpSuite

```
# start burp suite

def start_burp(self):
  command = "./burp/burp-rest-api.sh --headless.mode=false
--config-file=./burp/project-config.json --user-config-file=./burp/user-config.json"
  burpApi = subprocess.Popen(command, shell=True)
  return burpApi.pid
```

Shell script for starting burp-rest-api

```
#!/bin/bash -xe
SCRIPTPATH="$( cd "$(dirname "$0")" >/dev/null 2>&1 ; pwd -P )"
CLASSPATH="$(echo $SCRIPTPATH/build/libs/*.jar | tr ' ' ':'):$(echo
$SCRIPTPATH/*.jar | tr ' ' ':')"
java -noverify -javaagent:/home/oswas/oswas/bin/burp/burploader.jar -cp
"$CLASSPATH" org.springframework.boot.loader.JarLauncher $@
```

Shell script for starting burp-rest-api

```
# run proxy shell script
script_p = f"{os.getcwd()}/../app/lib/save_respone.py"
cmd = f'qterminal -e mitmdump -s {script_p} --mode upstream:http://127.0.0.1:8888
--ssl-insecure'
mitmproxy = subprocess.Popen(cmd, shell=True)
```

Function for starting crawler

```
def start(self):
  while True:
    self.start_url = input('Enter Web URL to start:')
    if self.start_url:
      if not self.start_url.endswith("/"):
        self.start_url += "/"
      self.Target = self.db.create_target({
        "start_url": self.start_url,
        "auth_url": None,
        "domain": urlparse(self.start_url).netloc,
        "status": TargetStatus.DOING,
        "started_at": str(datetime.datetime.utcnow()),
        "profiles": 0,
        "vulns": []
      })
      os.makedirs(f"output/{self.Target}", exist_ok=True)
      break
    else:
      print('PLEASE INPUT VALID URL')
```

Function for finding authentication page URL

```
def find_auth_link(self):
  url = self.start_url
  auth_url = None

  # prompt user to input auth_link
  choice = input("Do you have the URL of this Website's login page? (y/N): ")

  if not choice:
    choice = 'n'

  if choice.lower() == 'y':
    auth_url = input(" > Please enter URL of login page: ")
  elif choice.lower() == 'n':
```

```
    tmp = urlparse(url).netloc
    if tmp == '':
      parse_url = url
    else:
      parse_url = tmp

    # find links using dirsearch
    ds = Dirsearch(url, parse_url)
    ds._run()
    ds_links = ds.getURL()

    if len(ds_links):
      # ask user to find login_url from list
      auth_url = Utils.pager_input(ds_links, 20)
    else:
      print(" !!! 0 LINKS FOUND FROM DIRSEARCH")

      # check if start_url is auth_url
      if Utils.check_authlink(url):
        auth_url = url

  # update auth_url of Target
  if auth_url:
    self.auth_url = auth_url
    self.db.update_target(
      self.Target,
      { 'auth_url': auth_url }
    )

  return auth_url   # returns None or login link
```

Functions for installing & running dirsearch

```
# clone from github & install
def _install():
  if not os.path.exists('dirsearch'):
    os.system('git clone https://github.com/maurosoria/dirsearch.git')
    os.chdir('dirsearch')
```

```
    os.system('pip3 install -r requirements.txt')
    os.chdir("../")

# run dirsearch
def _run(url):
  path = os.getcwd()
  os.chdir('dirsearch')
  os.system(f'python3 dirsearch.py -u {url} -e php,js,asp,aspx,html -i 200-399 -x
400-599 --format simple -o ../{self.output_file}')
  os.chdir("../")
```

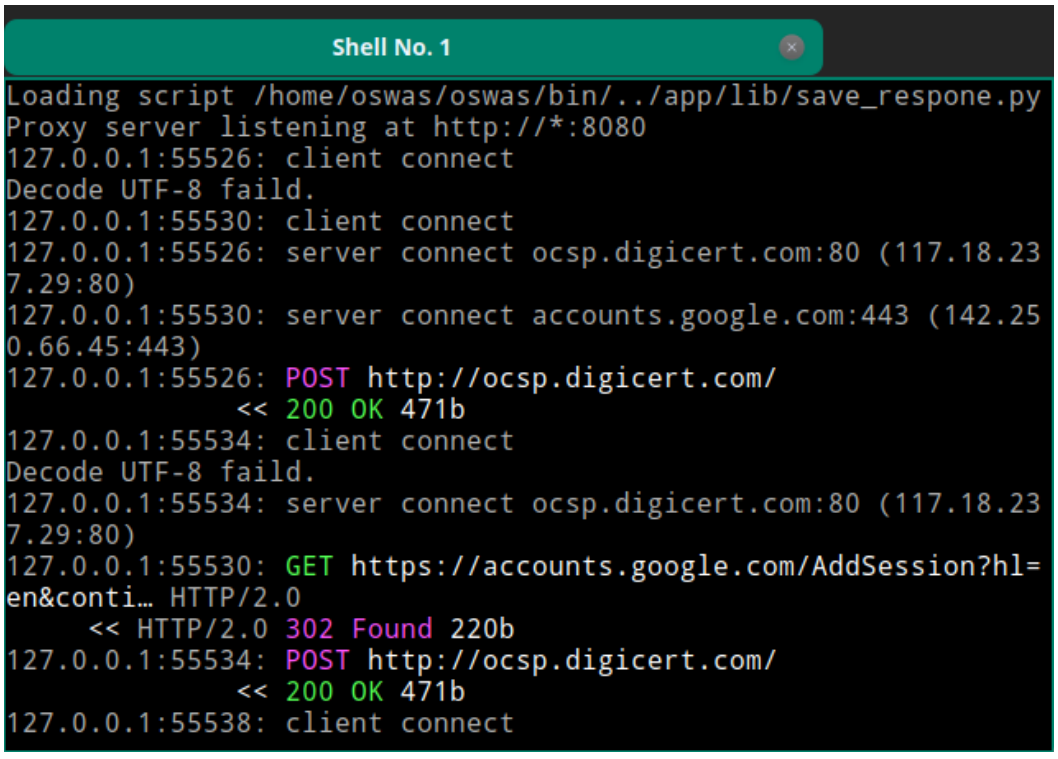
Function for identifying authentication page by signatures

```
AuthKeywords = [
 'Login', 'Log in' , 'login', 'loginform' , 'log in',
 '/login', 'Sign In',  'Sign in' , 'signin', 'sign in',
 'sign_in', '/sign_in', '/signin', 'Đăng nhập', 'dang-nhap',
 'dangnhap', '/dangnhap',
]

def check_authlink(url) -> bool:
  keywords = AuthKeywords
  html_output_name = urlparse(url).scheme

  res = requests.get(url)
  html_doc = res.text

  with open(html_output_name, 'w') as f:
   f.write(str(res.text))
   f.close()

  a_file = open(html_output_name)
  lines = a_file.readlines()

  # check if source code contains
  # any word in keyword dictionary
  for line in lines:
   if [ele for ele in keywords if (ele in str(line))]:
    a_file.close()
```

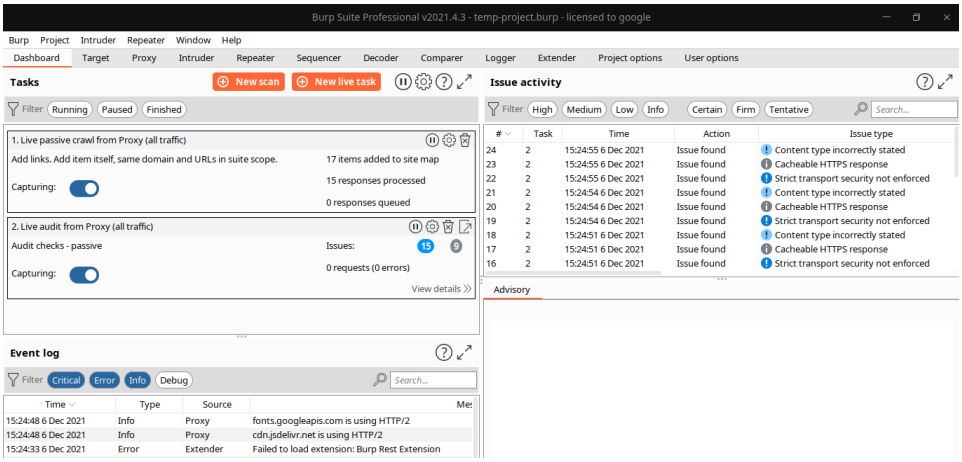
```
    os.remove(html_output_name)
    return True

os.remove(html_output_name)

# check type="submit" and type="password"
soup = bs4.BeautifulSoup(html_doc, 'html.parser')

eles = soup.select("form input[type=submit]")
eles1 = soup.select("form input[type=password]")

if (bool(eles) and bool(eles1)):
  return True

return False
```

Output:

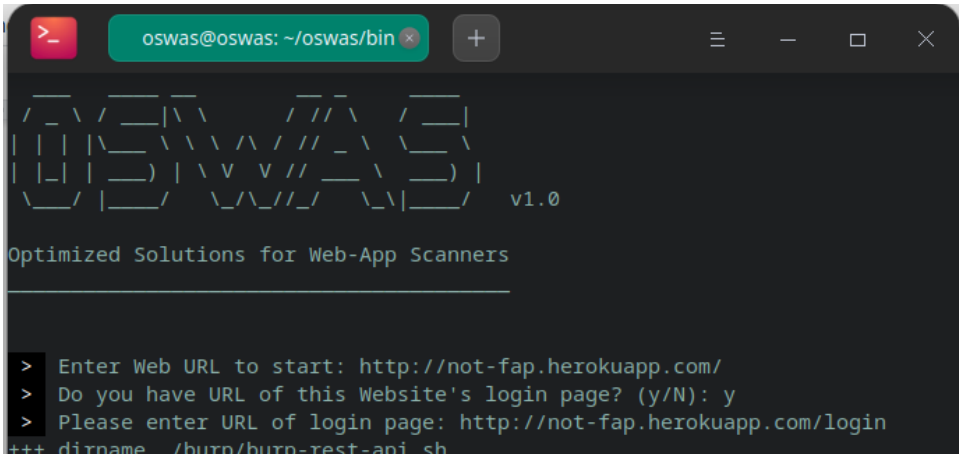
- MITM Proxy running



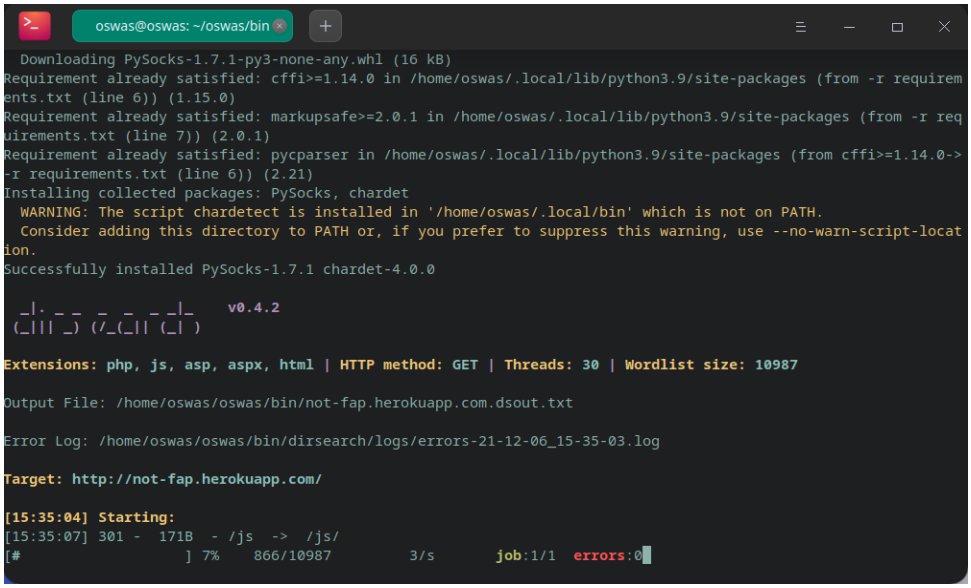
- Burp Suite Scanner running



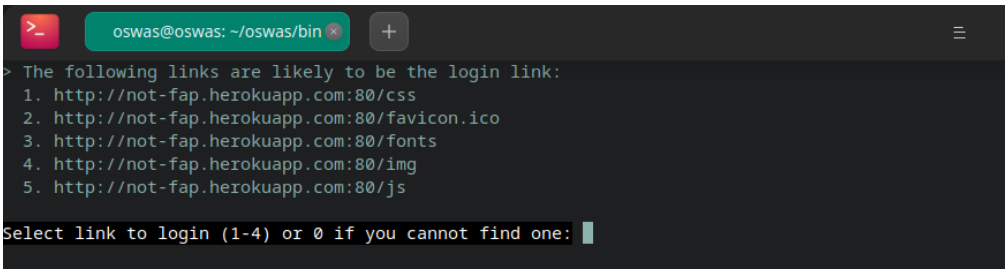
- Starting application



- Running dirsearch (case authentication URL is not provided)



- dirsearch results & user menu



5.4.2. Module II - Crawl

Module II is the phase of crawling a website, running on one single browser profile. In this module, there are two main jobs needed to be done: scrapping every link from the website (both authentication-required and non authentication-required links) using the URL provided from Module I and capturing their web traffic, which would be used for vulnerabilities scanning later on.

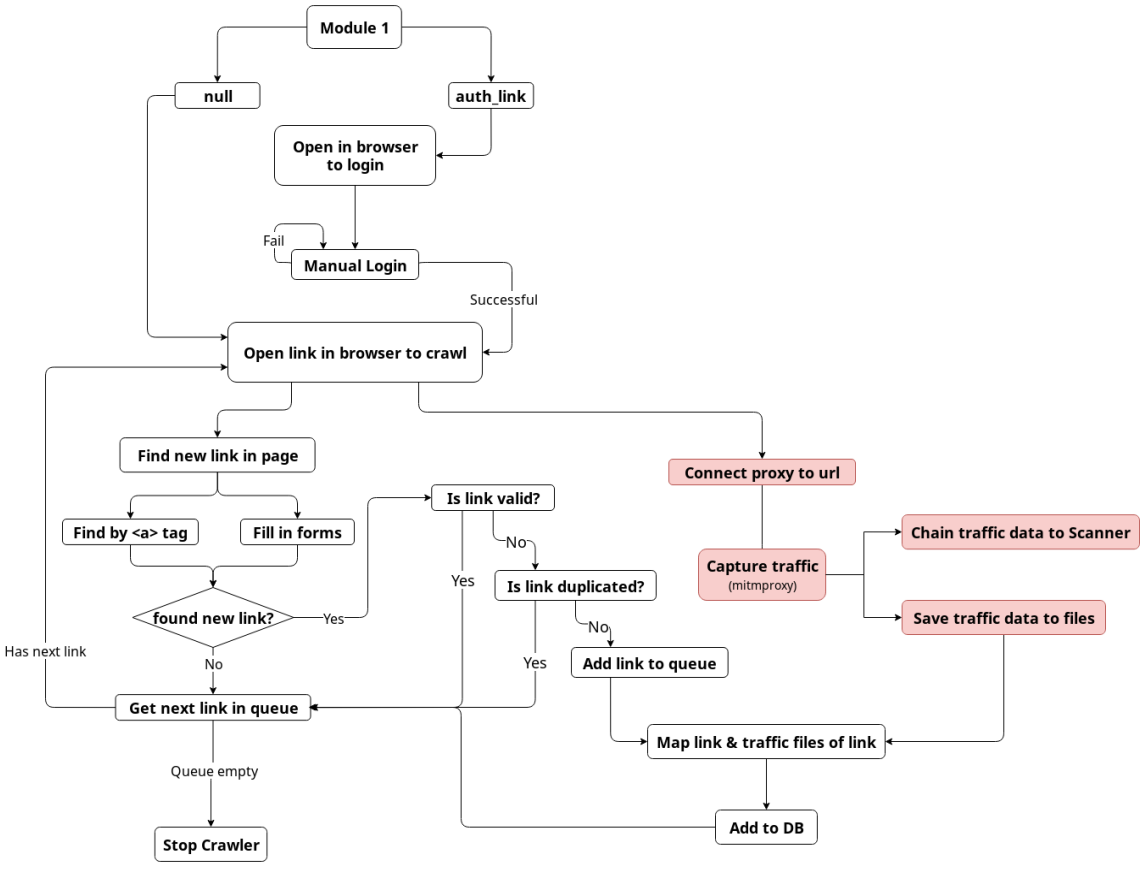


Figure 5.5.2. Workflow of Module II

Module II will use the result of Module I to begin with, which is the authentication page URL or null. Then, the crawling process will perform in two scenarios:

- **Scenario 1**: Module I returns *auth_url* (the URL of the authentication page)

The first step is to start a new Selenium Webdriver (a web browser), open the *auth_url* and manually login to the website with a user account. The web browser will automatically save the credentials/authentication information into the local storage, so we can crawl from authentication-required pages.

In the next step, we initiate a new queue to store the links then open the *start_url* and start crawling. As soon as the page is fully loaded, three jobs would be performed simultaneously:

- **Job 1:** The MITM proxy captures, stores web traffic of the current page into files.

- **Job 2:** The MITM proxy chains all the web traffic to the scanner (in this case, we use BurpSuite Professional).

- **Job 3:** The system finds and scraps all new links found in the current page. Firstly, we find all valid and unique links in the page through the property *href* of the <a> HTML tag. Then, we check if the page contains any input forms and fill the forms with custom dummy data then submit them, which can lead to some new links and new web traffic. The new links found from these steps will then be added to the queue. Finally, we map the new links to their web traffic files by adding the path of the traffic file to field *traffic_file* and update these links to the database.

Finally, we get the next link from the queue, open it on the browser and perform crawling on it. This process repeats until the queue is empty, then we stop the crawler. Module II returns a list of all the links found from the *start_url*.

- **Scenario 2**: Module I returns *null*

In this case, we do not have the URL authentication page so we can only crawl non authentication-required pages. The system starts a browser and opens *start_url*.

The following steps are similar to scenario 1:

- The system simultaneously captures, stores web traffic files and chains the web traffic to the scanner using the MITM proxy.
- At the same time, the system retrieves the new links from the current page by checking the property *href* of the <a> HTML tag and submitting forms.
- The new links that are found are added to the queue, then will be mapped with their traffic files and updated to the database
- The process repeats until the queue is empty then returns a list of all the links found from the *start_url*.

Source Code:

Config Firefox profile

```
proxy = '127.0.0.1:8080'
firefox_capabilities = webdriver.DesiredCapabilities.FIREFOX
firefox_capabilities['marionette'] = True
firefox_capabilities['proxy'] = {
    "proxyType": "MANUAL",
    "httpProxy": proxy,
    "sslProxy": proxy,
    "socksProxy": proxy,
    "socksVersion": 5
}
firefox_capabilities['acceptInsecureCerts'] = True
firefox_capabilities['acceptSslCerts'] = True
```

Functions for starting crawler

```
def start_crawler(self, auth_url, driver, user_type):
    print(f"START CRAWLER {self.start_url} | PROFILE {user_type}\n")

    self.internal_urls = set()
    self.external_urls = set()
    self.urls = Queue()
    self.done_links = []
    self.db_links = []

    if auth_url and not Utils.eq_urls(auth_url, self.start_url):
      self.db_links.append({
        "_id": ObjectId(),
```

```
    "target_id": self.Target,
    "url": auth_url,
    "user": user_type,
    "traffic_file": None,
  })

  driver.get(self.start_url)
  self.crawl(self.start_url, driver, user_type)
  driver.close()

  crawled_links = self.db.get_output_links(self.Target, user_type)

  return crawled_links
```

Functions for crawling

```
BLACKLIST = [
 'logout', 'Logout', 'log out', 'Log out', 'thoat', 'Thoat',  'dang xuat', 'Dang Xuat',
  'dang xuat', 'Dang xuat', 'logOut', 'LogOut', 'log Out', 'Log Out', 'dangXuat',
 'DangXuat',  'dang Xuat', 'Dang Xuat'
]

def crawl(self, url, driver, user_type):
  # remove redundant files in /tmp
  os.system("rm -rf *")
  os.mkdir(f"../output/{self.Target}/{user_type}")

  # crawl a web page and get all links
  links = self.get_all_website_links(url, driver)

  while True:
    # open the first link we found in new tab
    new_link = links.get()
    print(f"CRAWL {new_link}")

    blacked = False
    for word in BLACKLIST:
      if word in new_link:
        blacked = True
```

```
    if blacked: continue

    try:
      # open new Tab
      driver.execute_script("'window.open("");'")
      driver.switch_to.window(driver.window_handles[1])

      driver.get(new_link)
      self.crawl_from_forms(driver)

      # redirected link handler
      current_url_in_browser = driver.current_url
      if current_url_in_browser != new_link:
        if current_url_in_browser not in Utils.format_urls(self.internal_urls):
          links.put(current_url_in_browser)
      else:
        links = self.get_all_website_links(current_url_in_browser, driver)

      driver.close()

      # mapping link - traffic files
      new_link_id = ObjectId()
      traffic_path = Utils.map_link_traffic(new_link, str( self.Target), str(new_link_id),
user_type)
      self.db_links.append({
        "_id": new_link_id,
        "target_id": self.Target,
        "url": new_link,
        "user": user_type,
        "traffic_file": traffic_path,
      })

      if len(self.db_links) == 100:
        self.db.create_links_multi(self.db_links)
        self.db_links = []

      driver.switch_to.window(driver.window_handles[0])
      print(f"DONE {new_link}")
      self.done_links.append(new_link)
```

```
    except Exception as e:
      print(f"FAIL {new_link}")
    if links.empty():
      self.db.create_links_multi(self.db_links)
      self.db.update_target(
        self.Target,
        { 'status': TargetStatus.DONE }
      )
      return
```

Functions for finding new links in web page

```
def get_all_website_links(self, url, driver):
  # domain name of the URL without the protocol
  domain_name = urlparse(url).netloc
  try:
    all_a_tags = WebDriverWait(driver, 10).until(
      EC.presence_of_all_elements_located(
        (By.CSS_SELECTOR, "a")
      )
    )
  except:
    all_a_tags = []

  for a_tag in all_a_tags:
    href = a_tag.get_attribute("href")
    if not href:
      # href empty tag
      continue

    # join the URL if it's relative (not absolute link)
    href = urljoin(url, href)
    parsed_href = urlparse(href)

    # remove URL GET parameters, URL fragments, etc.
    href = parsed_href.scheme + "://" + parsed_href.netloc + parsed_href.path

    if parsed_href.query:
      href = href + "?" + parsed_href.query
```

```
    if "pdf" in urlparse(href).path:
      self.internal_urls.add(href)
      continue

    if not Utils.is_url_valid(href):
      # not a valid URL
      continue

    if href in self.internal_urls:
      # already in the set
      continue

    if domain_name != urlparse(href).netloc:
      # external link
      if href not in self.external_urls:
        self.external_urls.add(href)
      continue

    self.urls.put(href)
    self.internal_urls.add(href)
  return self.urls
```

Functions for crawl new links by filling & submitting forms

```
def crawl_from_forms(self, driver):
    form_spider = FormSpider(self.MAX_TRIES, self.MAX_WAIT, driver)
    selects = []
    inputs = []
    textareas = []

    # find inputs/textareas/selects
    selects = form_spider.find_elements_select()
    inputs = form_spider.find_elements_input()
    textareas = form_spider.find_elements_textarea()

    elements_to_fill = \
      list(map(WebElementObj.web_ele_2_select, selects)) + \
      list(map(WebElementObj.web_ele_2_input, inputs)) + \
```

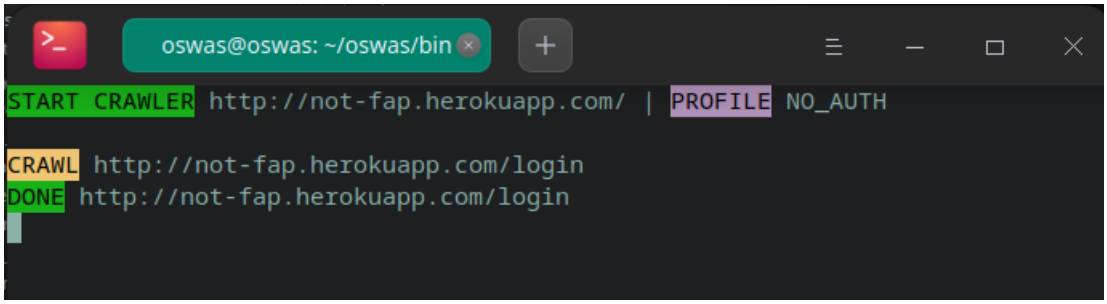
```
    list(map(WebElementObj.web_ele_2_textarea, textareas))

  for element in elements_to_fill:
    # fill select boxes
    if element.type == 'select':
      select = element.element
      form_spider.fill_select(select)
    # fill inputs
    elif element.type == 'input':
      input = element.element
      form_spider.fill_input(input)
    # fill textareas
    elif element.type == 'textarea':
      textarea = element.element
      form_spider.fill_textarea(textarea)

  # find submit buttons/inputs
  submits = form_spider.find_elements_submit()

  # open each form to new tabs
  for submit_element in submits:
    if submit_element.is_enabled() and submit_element.is_displayed():
      ActionChains(driver)\
        .key_down(Keys.CONTROL)\
        .click(submit_element)\
        .pause(5)\
        .key_up(Keys.CONTROL)\
        .perform()
  # get number of open tabs
  open_tabs = driver.window_handles

  # only run when at least 1 new tab is opened
  if len(open_tabs) > 2:
    for i in range(len(open_tabs)-1, 1, -1):
      driver.switch_to.window(driver.window_handles[i])
      self.urls.put(driver.current_url)
      self.internal_urls.add(driver.current_url)
      driver.close()
    driver.switch_to.window(driver.window_handles[1])
```

Function for mapping link with traffic files

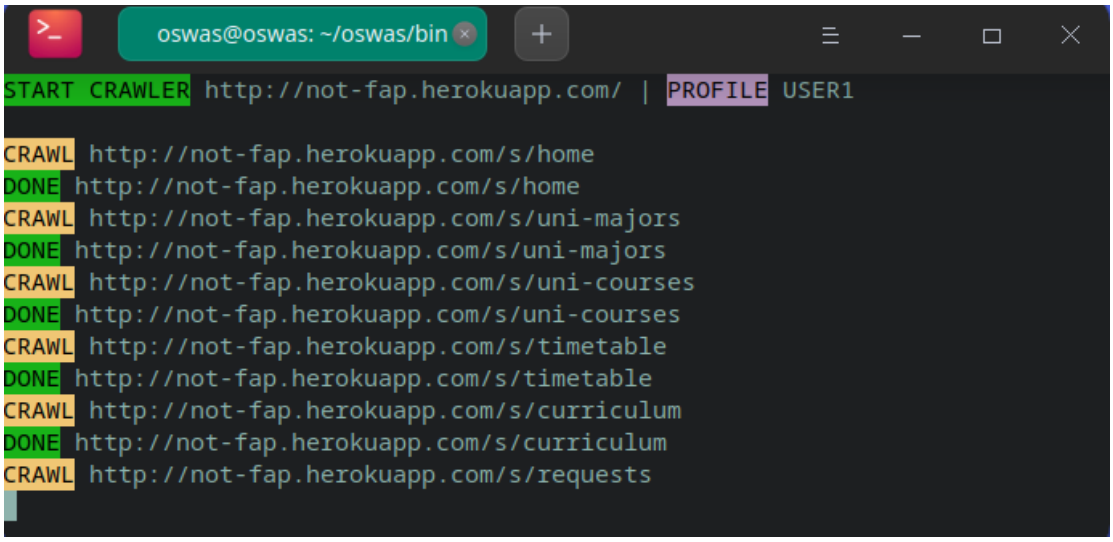
```
def map_link_traffic(link, target_id, output_dir, user_type):
  parent_dir = f"../output/{target_id}/{user_type}"
  path = os.path.join(parent_dir, output_dir)

  os.makedirs(path, exist_ok=True)
  if len(os.listdir('./')) != 0:
    os.chdir("../")
    os.system(f"mv ./tmp/* ./output/{target_id}/{user_type}/{output_dir}")
    os.chdir("tmp")

  return f"./output/{target_id}/{user_type}/{output_dir}"
```

Output:

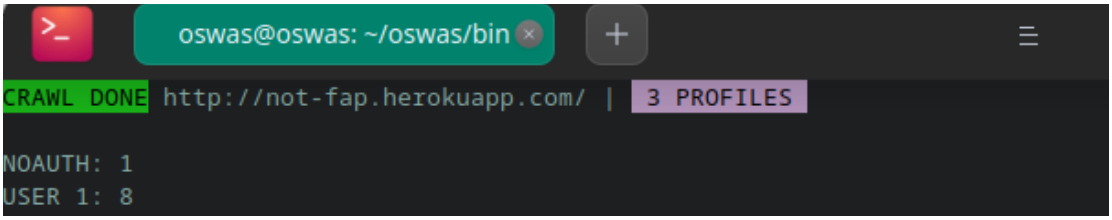
- Running crawler (non-authentication profile)



- Running crawler (authenticated with a user account)



- Crawler output

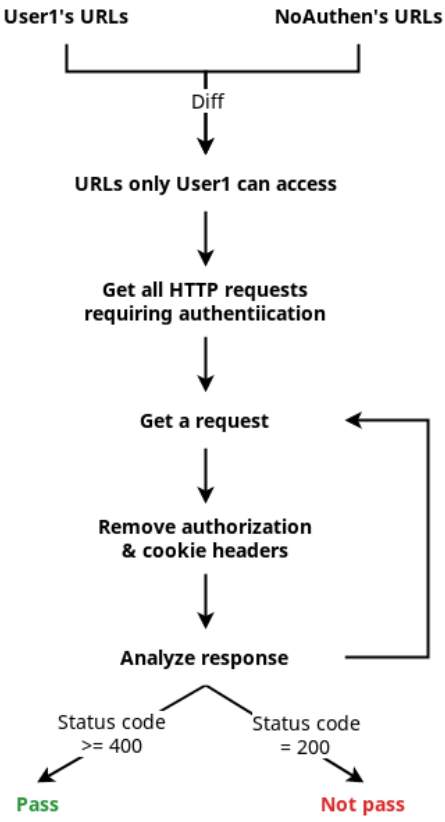


5.4.3. Module III - Vulnerabilities Testing

Module III is the last phase of the process. The main purpose of this module is to wrap up the crawler and perform the scanning process. Besides chaining the web traffic the an external Web Application Scanner, we intend to create our own vulnerability scanner, in order to specifically handle some vulnerabilities that most scanners currently could not, which are *Authentication Bypass Vulnerability* and *Insecure Direct Object References (IDOR) Vulnerability*.

Each vulnerability requires a specific strategy to handle:

- **Authentication Bypass Vulnerability:**

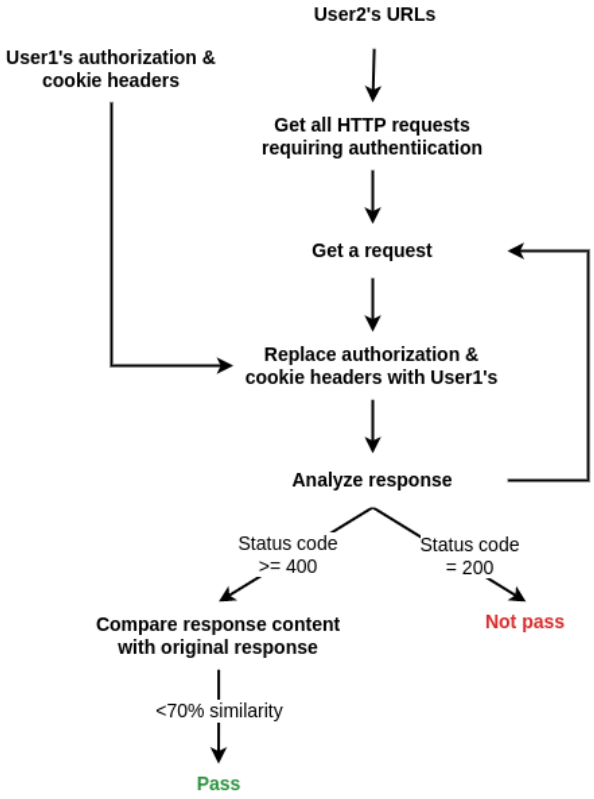


This vulnerability usually can be found from authentication-required an URL or an API that can be accessed when no authentication is provided, which can easily lead to sensitive data leakage and destruction. This is caused by missing authentication functionality when developed by uncareful developers.

In order to identify this vulnerability, the first thing we need is to retrieve every link that requires authentication. We run the crawler with 2 profiles: no authentication provided (NO AUTH profile) and authenticated with a user account (USER 1 profile), then we collect every link that only the USER 1 profile has. We then check the HTTP requests that are sent from these links and identify every link that contains any authentication signatures (Authorization header, *Cookie* header) in its HTTP request. This final list we retrieve contains every authentication-required link

Then, for each link in the list, we edit its HTTP request, strip all of its authentication signatures (Authorization header, *Cookie* header) and resend the edited HTTP request using Python's *request* library. If any link receives 2xx (success) or 3xx (redirection) response status code, we identify that it has Authentication Bypass Vulnerability.

- **Horizontal IDOR**

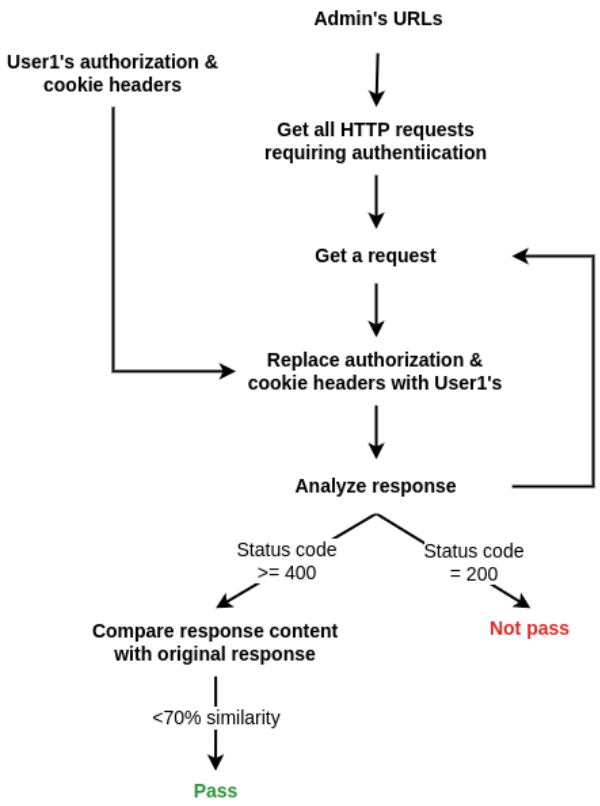


This vulnerability is caused by the lack of horizontal authorization between two users who have the same privilege level, which can lead to horizontal escalation of privilege (one user can view/modify information of another user).

Handling this vulnerability requires two lists of links that two users having the same privilege level get access to, the two users are referred to as USER 1 and USER 2. Then, we run the test for each user profile. The test will be performed in the following steps:

- Retrieve every authentication signature of USER 2 from USER 2's HTTP requests (*Authorization* header or *Cookie* header)
- For each link of USER 1, edit its HTTP request, swap the *Authorization/Cookie* header with USER 2's and then resend the request.
- Examine the response: if a link returns the status code 2xx (success) or 3xx (redirected), it means that USER 2 can get access to USER 1's link
- In some cases, two users can access the same link, but the response content is different based on their authentication (E.g /about-me API returns data based on user authentication token). In order to eliminate this false-positive result, we then compare the original response content with the re-sent response content: if the two contents are mostly similar (more than 70% similarity amount), we mark the link with a possible horizontal IDOR vulnerability.
- Perform all previous steps again, on USER 2's links using USER 1's authentication signatures, to check for horizontal IDOR vulnerabilities of USER 2.

- **Vertical IDOR**



Vertical Access Control aims to control the restrictions to access functions according to the user roles. This vulnerability can be found when a low-level user has access to assets of a higher-level user (E.g employee account can access to an URL that only admin is authorized).

For this test, we need two lists of URLs retrieved from two user profiles (USER & ADMIN). Then, the test if performed in the following steps:

- Compare USER URL list to ADMIN URL list and extract the URLs that only appear in ADMIN (the URLs only ADMIN has access to) into a new list (referred to as DIFF_LIST)
- Retrieve every authentication signature of USER from USER's HTTP requests (*Authorization* header or *Cookie* header)
- For each link in DIFF_LIST, edit its HTTP request, change the *Authorization/Cookie* headers to the USER's headers then resend the request.
- Examine the response: if a link returns the status code 2xx (success) or 3xx (redirected), it means that USER can get access to ADMIN's link and we mark the link with a possible horizontal IDOR vulnerability. In order to eliminate this false-positive result, we

then compare the original response content with the re-sent response content: if the two contents are mostly similar (more than 70% similarity amount), we mark the link with a possible vertical IDOR vulnerability.

- **Text Difference Comparison Algorithm**

In order to compute the text difference between two HTTP responses, we use Python package *difflib*'s SequenceMatcher class.

The difflib module provides classes and functions for comparing sequences. It can be used to compare files and can produce information about file differences in various formats.

This class can be used to compare two input sequences or strings. In other words, this class is useful to use when finding similarities between two strings on the character level.

The basic idea behind *SequenceMatcher()* is to find the longest contiguous matching subsequence (LCS) that contains no "junk" elements. Junk are the things that we don't want the algorithm to match on, like blank lines in ordinary text files, <P> lines in HTML files, etc. This does not yield minimal edit sequences, but does tend to yield matches that "look right" to people.

The ratio() function returns the similarity score (float in [0,1]) between input strings and sums the sizes of all matched sequences returned by the get_matching_blocks() function. It calculates the ratio as:

$$\text{Ratio} = 2.0 * \frac{M}{T}$$

where M="matches" and T="total number of elements" in both the sequences.

This piece of code is used to calculate the similarity percentage between the original response content and the response of the modified request

```
response_old  # original response
response_new  # response of modified request

# get response text difference
diff_amount = SequenceMatcher(None, response_old.text, response_new.text).ratio() * 100
```

- **Workflow**

The specific workflow of MODULE III is shown below:

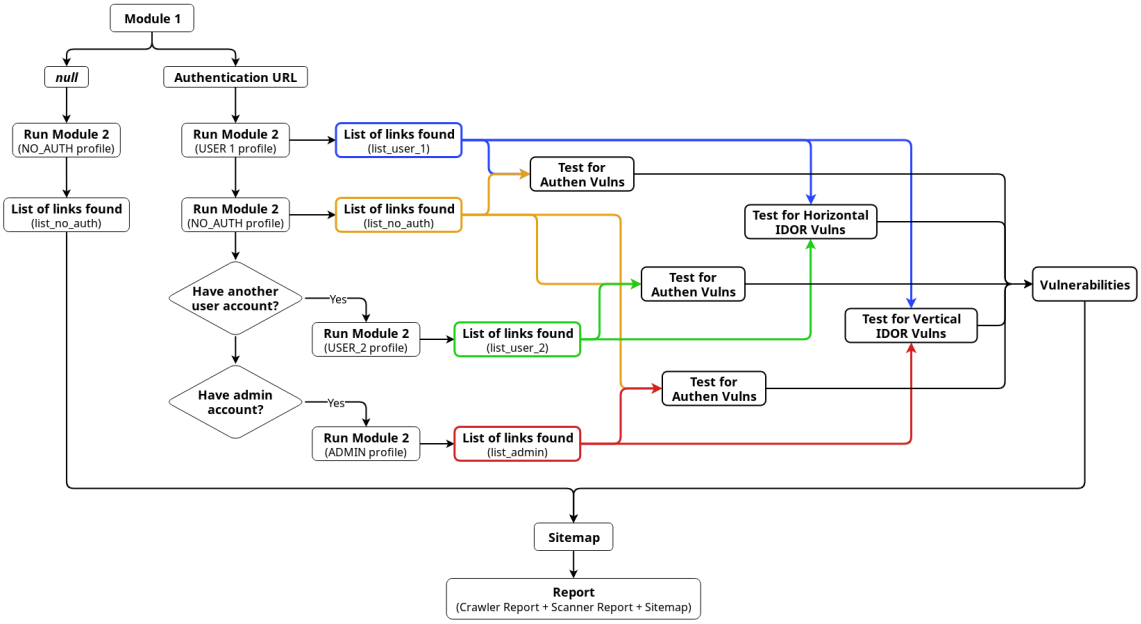


Figure 5.5.3. Workflow of Module III

Wrapping up the crawler

From Module I and Module II, we achieve a complete crawling process running with a single user profile. As the ideology of handling the vulnerabilities shown above, there are four user profiles that need to be crawl in order to cover all cases:

- non-authentication (referred to as NO_AUTH)
- low-level user account (referred to as USER 1 and USER 2)
- high-level user account (referred to as ADMIN)

If Module I returns an authentication URL, we can at least crawl two user profiles: NO_AUTH and USER 1. We run Module 2 with USER 1 first, then with NO_AUTH profile. We then ask if the user wanted to use another user account to crawl (this user account must be in the same privilege level as the first user account). If the user has another user account, we then have the results of USER 2. Finally, we ask if the user had a high-level user account (this user account must have a higher privilege level than the first user account). If the user can provide a high-level user

account, we can get the result of ADMIN. After the crawling phase, we perform the vulnerability scanning in the following cases:

- NO_AUTH & USER 1: Test for Authentication Bypass Vulnerability
- NO_AUTH & USER 2: Test for Authentication Bypass Vulnerability
- NO_AUTH & ADMIN: Test for Authentication Bypass Vulnerability
- USER 1 & USER 2: Test for Horizontal IDOR Vulnerability
- USER 1 & ADMIN: Test for Vertical IDOR Vulnerability

The scanning process returns a list of vulnerabilities then synchronizes it to the database.

If Module I does not return an authentication URL, we only run Module 2 (crawl and chain web traffic to Scanner) with NO_AUTH profile and skip the scanning process.

We then add a minor function to generate a sitemap, based on the links we crawled.

Finally, we generate a crawler & scanner report, which contains the result of the crawler, the identified vulnerabilities and the sitemap, and shows it to the user.

Source Code:

Complete crawler code flow

```
# crawl no authen
if auth_url is None:
  profiles_count += 1

  res_noauth = project.start_crawler(
    None,
    webdriver.Firefox(capabilities=firefox_capabilities),
    UserCrawlType.NO_AUTH
  )
else:
  driver_user_1 = webdriver.Firefox(capabilities=firefox_capabilities)
  if Utils.prompt_login(auth_url, driver_user_1):
    profiles_count += 1
    res_user_1 = project.start_crawler(
      auth_url,
      driver_user_1,
      UserCrawlType.USER1
```

```
    )
    profiles_count += 1
    res_noauth = project.start_crawler(
      None,
      webdriver.Firefox(capabilities=firefox_capabilities),
      UserCrawlType.NO_AUTH
    )

    # crawl with user 2
    has_user_2 = Utils.yes_no_question('Do you have another user account?')
    if has_user_2:
      driver_user_2 = webdriver.Firefox(capabilities=firefox_capabilities)
      if Utils.prompt_login(auth_url, driver_user_2):
        profiles_count += 1
        res_user_2 = project.start_crawler(
          auth_url,
          driver_user_2,
          UserCrawlType.USER2
        )

    # crawl with admin
    has_admin = Utils.yes_no_question('Do you have an admin account?')
    if has_admin:
      driver_admin = webdriver.Firefox(capabilities=firefox_capabilities)
      if Utils.prompt_login(auth_url, driver_admin):
        profiles_count += 1
        res_admin = project.start_crawler(
          auth_url,
          driver_admin,
          UserCrawlType.ADMIN
        )
```

Function for running vulnerability scanner

```
def run_scanner(self, links_noauth, links_user1, links_user2, links_admin):
  print('SCANNING FOR VULNERABILITIES...\n')

  # scan authen vuln (user)
  if isinstance(links_user1, list) and len(links_user1) > 0:
```

```
    # get links only in appears links_user1
    diff_noauth_user = self.get_diff(links_user1, links_noauth)
    self.scan_authen_vuln(diff_noauth_user)

  # scan authen vuln (admin)
  if isinstance(links_admin, list) and len(links_admin) > 0:
    # get links only appears in links_admin
    diff_noauth_admin = self.get_diff(links_admin, links_noauth)
    self.scan_authen_vuln(diff_noauth_admin)

  # scan vertical IDOR (user 1 & admin)
  if isinstance(links_user1, list) and len(links_user1) > 0 and isinstance(links_admin, list)
and len(links_admin) > 0:
    self.scan_vtc_idor(links_admin, links_user1)

  # scan horizontal IDOR (user 1 & user 2)
  if isinstance(links_user1, list) and len(links_user1) > 0 and isinstance(links_user2, list)
and len(links_user2) > 0:
    self.scan_hrz_idor(links_user1, links_user2)

  res = [i for n, i in enumerate(self.VULN_LINKS) if i not in self.VULN_LINKS[n + 1:]]
  print(f"\nVulns: {len(res)}\n\n")
  return res
```

Function for scanning authentication bypass vulnerability

```
VULN_LINKS = []
AUTH_HEADER = 'authorization'
COOKIE_HEADER = 'cookie'
CSRF_SIG = ["csrf-token", "csrf_token", "token"]

def scan_authen_vuln(self, links):
  for link in links:
    AUTH_FILES = []
    origin_header = {}
    test_header = {}
    method = ""

    origin_path = self.cwd + link["traffic_file"]
```

```
# r=root, d=directories, f=files
for r, d, f in os.walk(origin_path):
  for file in f:
    if '.txt' in file:
      url = PurePath(r).name.replace("\\\\", "://")
      os.chdir(r)
      with open(file) as content:
        content_l = content.read().lower()
        if self.AUTH_HEADER in content_l or self.COOKIE_HEADER in content_l:
          shutil.copy(os.path.join(r, file), self.testing_path)
          AUTH_FILES.append({
            "url": url,
            "file": file
          })

os.chdir(self.testing_path)

for item in AUTH_FILES:
  URL, FILE = item.values()
  body_data = []

  # get original headers & data
  with open(FILE, "r") as f_origin:
    lines = f_origin.read().splitlines()

    if lines[0].split()[1].startswith('http'):
      url = lines[0].split()[1]
    else:
      url = URL + lines[0].split()[1]

    method = lines[0].split()[0]
    is_header = True

    for line in lines[1:]:
      while is_header:
        if line == "\n" or line == "":
          is_header = False
          break
```

```
            x = line.split(':', 1)
            origin_header[x[0].strip()] = x[1].strip()
            break
          else:
            body_data.append(line)

        # map request body
        # try map to JSON format
        json_data = {}
        params = {}
        if len(body_data) > 0:
          if str(body_data[0]).startswith("{") and
str(body_data[len(body_data)-1]).endswith("}"):
            with open(f'request_{FILE}', 'w') as f_req:
              for data in body_data:
                f_req.write(f'{data}\n')
            with open(f'request_{FILE}', 'r') as f_req_r:
              json_data = json.load(f_req_r)
          else:
            for param in body_data:
              p = param.split('&')
              if len(p) > 0:
                for value in p:
                  v = value.split('=')
                  if len(v) > 0:
                    params[v[0].strip()] = v[1].strip()

        # create testing request - remove auth headers
        with open(f"testing_{FILE}", "w") as f_noauth:
          tmp = None
          for line in lines:
            line_l = line.lower()
            if line_l.startswith(self.AUTH_HEADER) or
line_l.startswith(self.COOKIE_HEADER):
              tmp = line
            if line != tmp:
              f_noauth.write('{}\n'.format(line))

        # get testing headers & data
```

```
        with open(f"testing_{FILE}", "r") as f_noauth_r:
          lines_noauth = f_noauth_r.read().splitlines()

          for line in lines_noauth[1:len(lines_noauth) - 1]:
            if line == "\n" or line == "":
              break
            x = line.split(':', 1)
            test_header[x[0].strip()] = x[1].strip()

        perform_test = False

        # send requests & get responses
        if method.lower() == 'get':
          perform_test = True
          res_o = requests.get(url, headers=origin_header)
          res_t = requests.get(url, headers=test_header)
        elif method.lower() in ['post', 'put']:
          res_o = requests.post(url, headers=origin_header, json=json_data,
params=params)
          res_t = requests.get(url, headers=test_header, json=json_data, params=params)

        if perform_test:
          is_vuln = False
          # compare response text difference
          diff_amount = SequenceMatcher(None, res_o.text, res_t.text).ratio() * 100

          # add to vuln list if 2 responses are >= 70% similar
          if res_t.status_code < 400:
            if diff_amount > 70:
              is_vuln = True
              self.VULN_LINKS.append({
                "type": "Authentication Vulnerability",
                "link": link["url"],
                "vuln_link": f"{method} {url}",
                "diff": diff_amount
              })

          if is_vuln:
```

```
        print(f'FOUND AUTH. VULN: {method.upper()} {url} {res_t.status_code}
{"{:.2f}".format(diff_amount)}')

    os.remove(FILE)
    os.remove(f"testing_{FILE}")
```

Function for collecting a user's authentication headers

```
AUTH_HEADER = 'authorization'
COOKIE_HEADER = 'cookie'

def get_user_tokens(self, links):
  headers = []
  for link in links:
   if link["traffic_file"]:
    path = self.cwd + link["traffic_file"]

    # r=root, d=directories, f=files
    for r, d, f in os.walk(path):
     for file in f:
      if '.txt' in file:
       os.chdir(r)
       with open(file, "r") as content:
        content = content.read()
        content_l = content.lower()
        if self.AUTH_HEADER in content_l or self.COOKIE_HEADER in content_l:
         lines = content.splitlines()
         for line in lines:
          line_l = line.lower()
          if line_l.startswith(self.AUTH_HEADER) or
line_l.startswith(self.COOKIE_HEADER):
           x = line.split(':', 1)
           headers.append({
            "h": x[0].strip(),
            "c": x[1].strip(),
           })

  # remove duplicates and return list of headers
  return [i for n, i in enumerate(headers) if i not in headers[n+1:]]
```

Function for testing IDOR vulnerability between 2 users

```
VULN_LINKS = []
AUTH_HEADER = 'authorization'
COOKIE_HEADER = 'cookie'
CSRF_SIG = ["csrf-token", "csrf_token", "token"]

def scan_idor(self, test_links, user_links, type):
  # get user auth/token headers
  user_auth_headers = self.get_user_tokens(user_links)

  # get files having auth headers
  for link in test_links:
   if link["traffic_file"]:
    TEST_FILES = []
    origin_header = {}
    test_header = {}
    method = ""

    origin_path = self.cwd + link["traffic_file"]

    # r=root, d=directories, f=files
    for r, d, f in os.walk(origin_path):
     for file in f:
      if '.txt' in file:
       url = PurePath(r).name.replace("\\\\", "://")
       os.chdir(r)
       with open(file) as content:
        content = content.read()
        content_l = content.lower()

        # check if request has CSRF token
        has_csrf = False
        lines = content.splitlines()
        for line in lines:
         line_l = line.lower()
         if any(word in line_l for word in self.CSRF_SIG):
          has_csrf = True
```

```
        if not has_csrf:
          # check if request file has auth/cookie header
          if self.AUTH_HEADER in content_l or self.COOKIE_HEADER in content_l:
            shutil.copy(os.path.join(r, file), self.testing_path)
            TEST_FILES.append({
              "url": url,
              "file": file
            })

os.chdir(self.testing_path)

for item in TEST_FILES:
  URL, FILE = item.values()
  body_data = []

  # get original headers & data
  with open(FILE, "r") as f_origin:
    lines = f_origin.read().splitlines()

    method = lines[0].split()[0]
    url = URL + lines[0].split()[1]
    is_header = True

    for line in lines[1:]:
      while is_header:
        if line == "\n" or line == "":
          is_header = False
          break
        x = line.split(':', 1)
        origin_header[x[0].strip()] = x[1].strip()
        break
      else:
        body_data.append(line)

  # map request body
  # try map to JSON format
  json_data = {}
  params = {}
```

```
        if len(body_data) > 0:
          if str(body_data[0]).startswith("{") and
str(body_data[len(body_data)-1]).endswith("}"):
            with open(f'request_{FILE}', 'w') as f_req:
              for data in body_data:
                f_req.write(f'{data}\n')
            with open(f'request_{FILE}', 'r') as f_req_r:
              json_data = json.load(f_req_r)
          else:
            for param in body_data:
              p = param.split('&')
              if len(p) > 0:
                for value in p:
                  v = value.split('=')
                  if len(v) > 0:
                    params[v[0].strip()] = v[1].strip()

        for i, header in enumerate(user_auth_headers):
          test_file = f"testing_{i+1}_{FILE}"

          # create testing request files - add custom auth headers
          with open(test_file, "w") as f_test:
            for line in lines:
              line_l = line.lower()
              if line_l.startswith(self.AUTH_HEADER) or
line_l.startswith(self.COOKIE_HEADER):
                f_test.write(f'{header["h"]}: {header["c"]}\n')
              else:
                f_test.write(f'{line}\n')

          # get testing headers & data
          with open(test_file, "r") as f_test_r:
            lines_test = f_test_r.read().splitlines()
            for line in lines_test[1:len(lines_test) - 1]:
              if line == "\n" or line == "":
                break
              x = line.split(':', 1)
              test_header[x[0].strip()] = x[1].strip()
```

```
        perform_test = False

        # send requests & get responses
        if method.lower() == 'get':
          perform_test = True
          res_o = requests.get(url, headers=origin_header)
          res_t = requests.get(url, headers=test_header)
        elif method.lower() in ['post', 'put']:
          perform_test = True
          res_o = requests.post(url, headers=origin_header, json=json_data,
params=params)
          res_t = requests.get(url, headers=test_header, json=json_data, params=params)

        if perform_test:
          is_vuln = False
          # compare response content text difference
          diff_amount = SequenceMatcher(None, res_o.text, res_t.text).ratio() * 100

          # add to vuln list if 2 responses are >= 70% similar
          # & status code != 401 and 403
          if res_t.status_code < 400:
            if diff_amount > 70:
              is_vuln = True
              self.VULN_LINKS.append({
                "type": type,
                "link": link["url"],
                "vuln_link": f"{method} {url}",
                "diff": diff_amount
              })

            if is_vuln:
              print(f'FOUND IDOR. VULN: {method.upper()} {url} {res_t.status_code}
{"{:.2f}".format(diff_amount)}')

        os.remove(test_file)
      os.remove(FILE)
```

Function for testing horizontal IDOR vulnerability

```
def scan_hrz_idor(self, links_user1, links_user2):
  # test IDOR for user 1
  self.scan_idor(links_user1, links_user2, "Horizontal IDOR")
  # test IDOR for user 2
  self.scan_idor(links_user2, links_user1, "Horizontal IDOR")
```

Function for testing vertical IDOR vulnerability

```
def scan_vtc_idor(self, links_admin, links_user):
  # get links only admin user can access
  diff_links = self.get_diff(links_admin, links_user)
  # scan vertical IDOR
  self.scan_idor(diff_links, links_user, "Vertical IDOR")
```

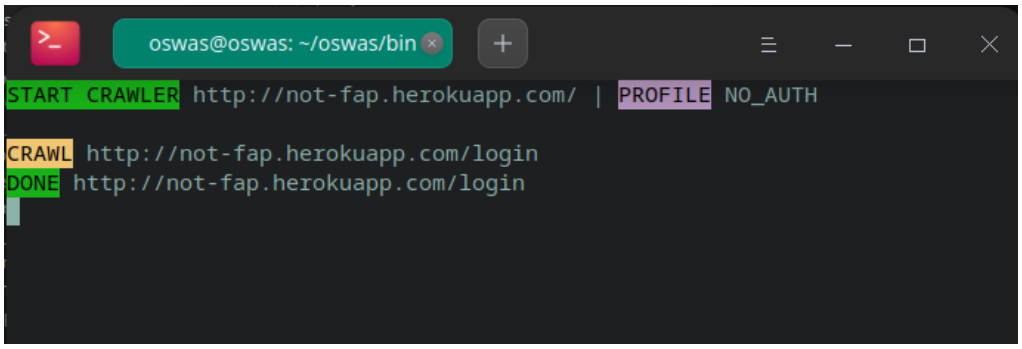
Function for creating XML sitemap

```
def build_xml(self):
    with open(f"sitemap.xml", "w") as f:
      f.writelines("<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<urlset
xmlns=\"http://www.sitemaps.org/schemas/sitemap/0.9\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:schemaLocation=\"http://www.sitemaps.org/schemas/sitemap/0.9
http://www.sitemaps.org/schemas/sitemap/0.9/sitemap.xsd\">\n")
      for url in self.url_list:
        clean_url = url.replace(self.starting_url, "/")

        if clean_url[-1:] != "/":
          clean_url += "/"

        self.crawl_depth = clean_url.count("/") - 1

        if clean_url != "/":
          self.parent = self.get_parent(clean_url)
        else:
          self.parent = "/"

        if self.parent not in self.parent_urls and self.parent != "/":
```

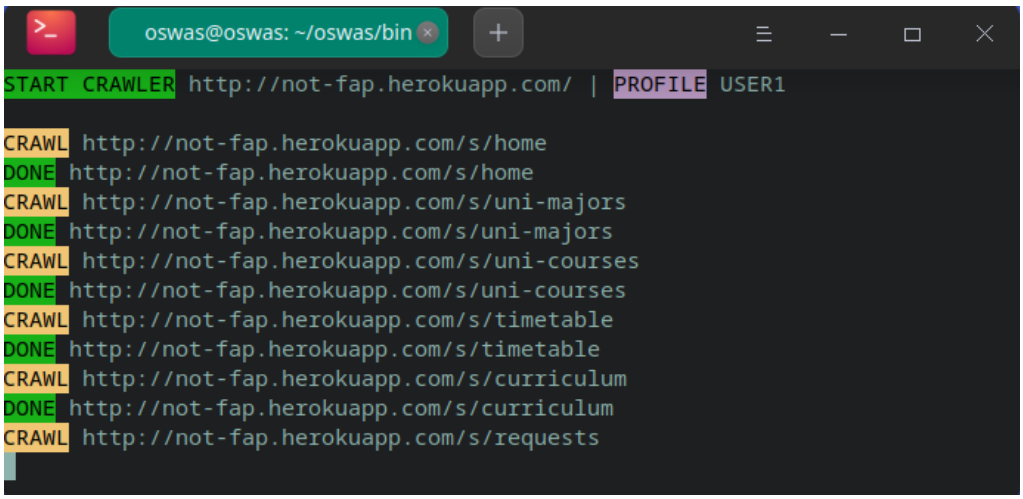
```
            self.parent_urls.append(self.parent)

        self.add_to_tree(clean_url)

        f.writelines("  <url>\n")
        f.writelines("    <loc>" + url + "</loc>\n")
        f.writelines(
            "    <lastmod>" + datetime.datetime.now().strftime("%Y-%m-%d") +
"</lastmod>\n")
        f.writelines("    <changefreq>daily</changefreq>\n")
        f.writelines("    <priority>1.0</priority>\n")
        f.writelines("  </url>\n")
    f.writelines("</urlset>")
```

Output:

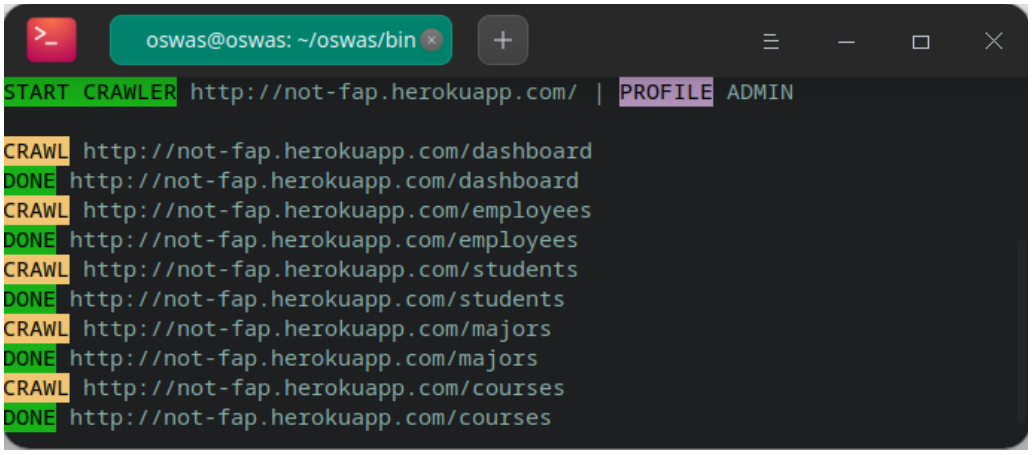
- Running crawler (non-authentication profile)



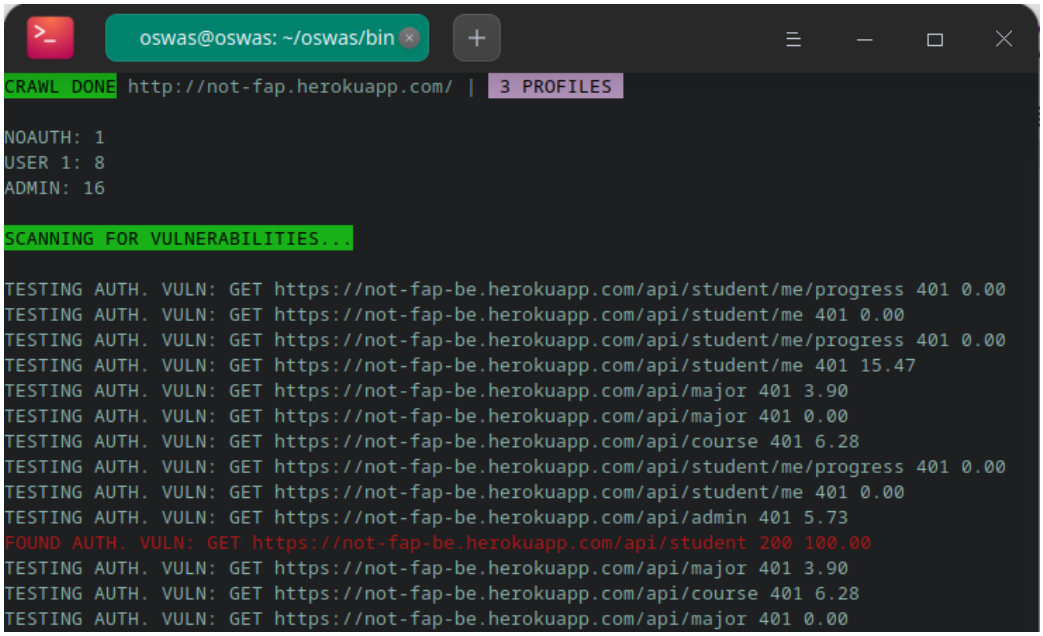
- Running crawler (authenticated with a user account)



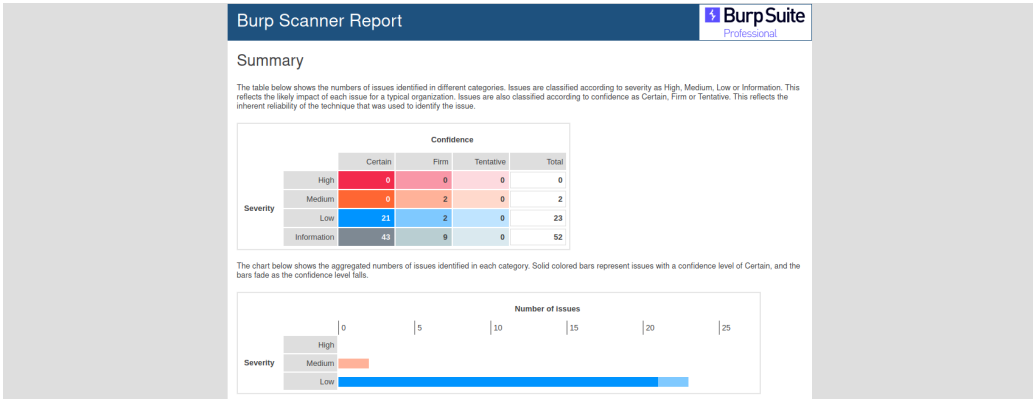
- Running crawler (authenticated with an admin account)



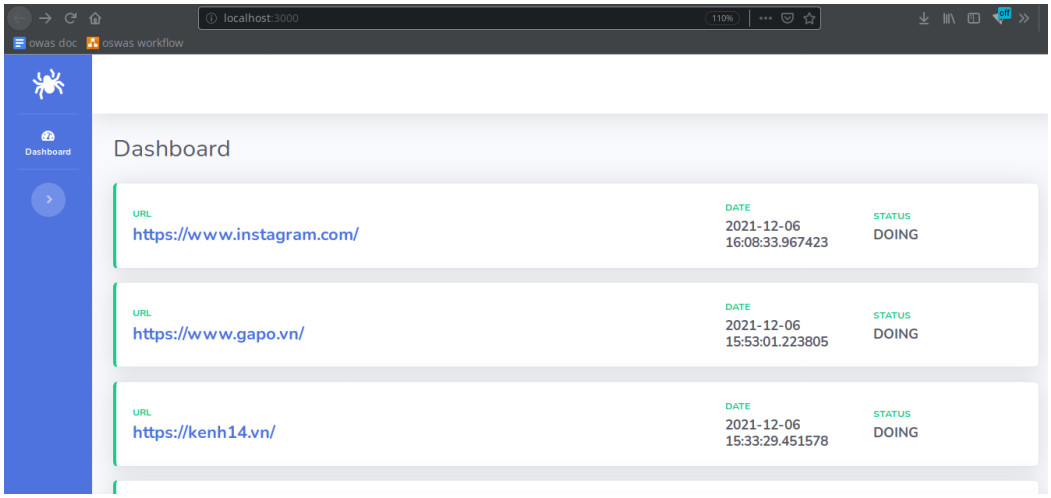
- Testing Authentication Bypass & IDOR vulnerabilities



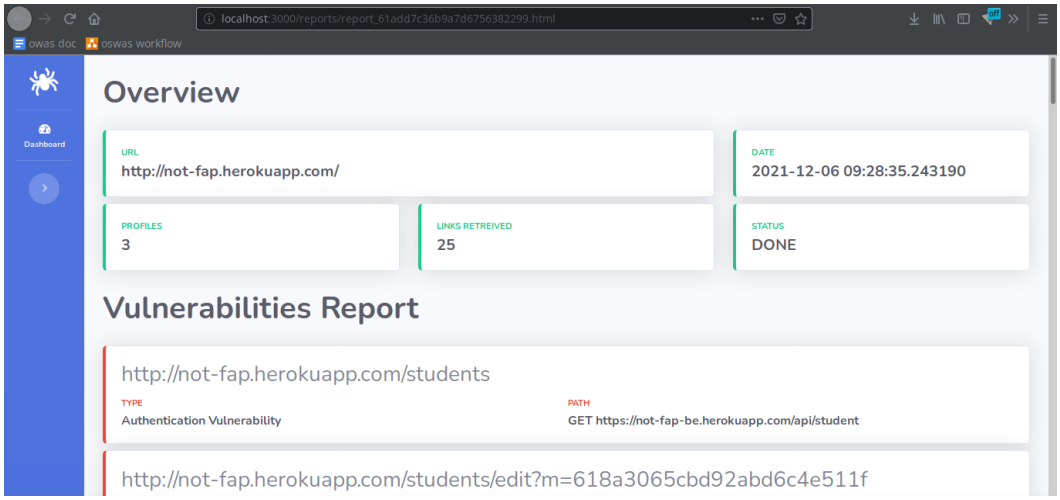
- Scanner report (using Burp Suite Scanner)



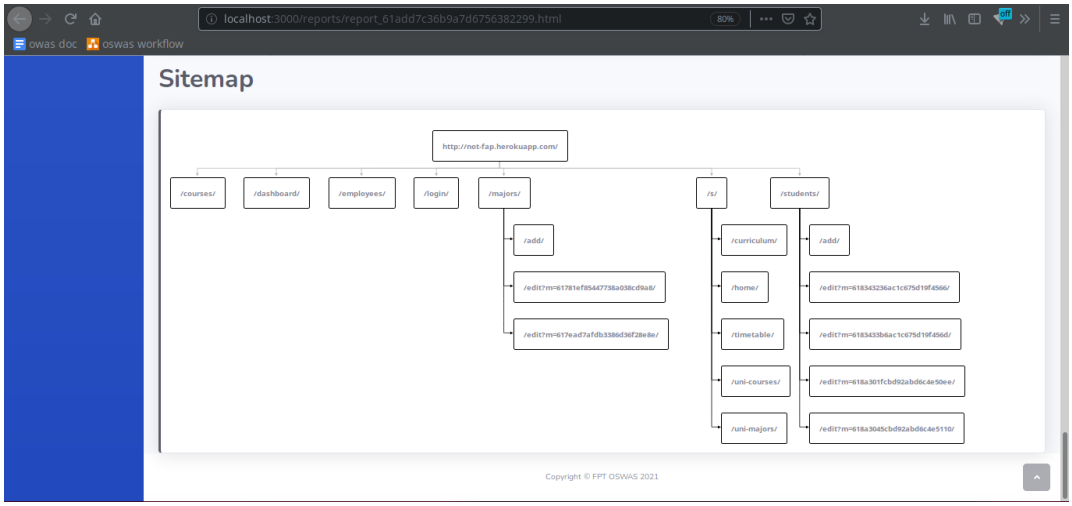
- Report dashboard UI



- Crawler & scanner report UI



- Sitemap UI



6. Project Validation

6.1. Crawling results

We performed the crawler on several web applications. Based on the Alexa's top-ranking web applications, we choose the websites that satisfy the following criterias:

- E-commerce web applications.
- Online forums.
- Social network web applications.
- Content management sites (CMS) web applications.

Crawling results:

- Perform crawling a CMS web application: ***not-fap.herokuapp.com***
  - Total runtime: 35 minutes
  - Results: 24 links

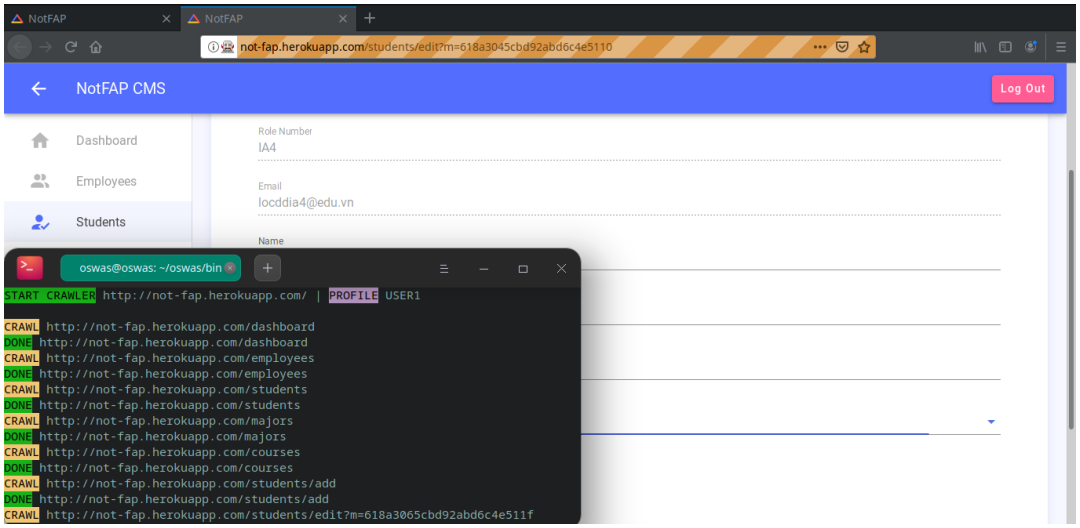


Figure 6.1a. Crawling not-fap.herokuapp.com

- Perform crawling on a social media web application: ***www.instagram.com***
  - Total runtime: 11 hours 47 minutes
  - Result: 1051 links

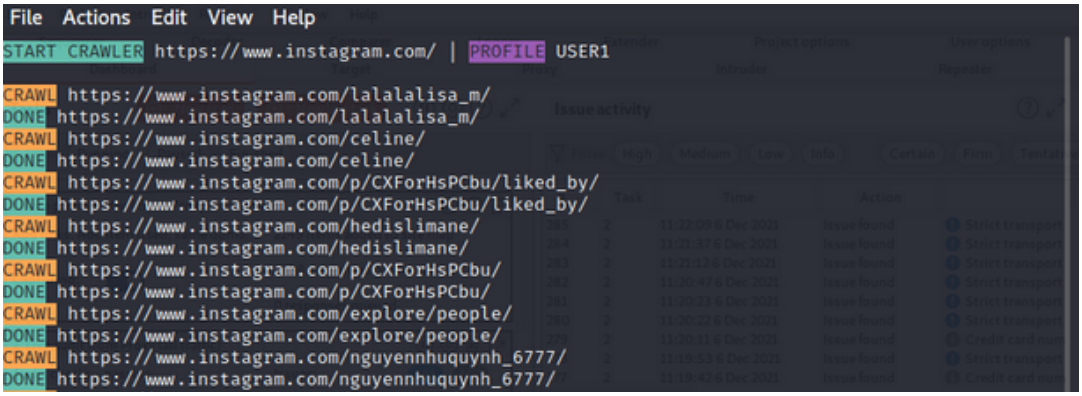


Figure 6.1b. Crawling www.instagram.com

- Perform crawling on a social media web application: **www.gapo.vn**
  - Total runtime: 9 hours 35 minutes
  - Result: 854 links

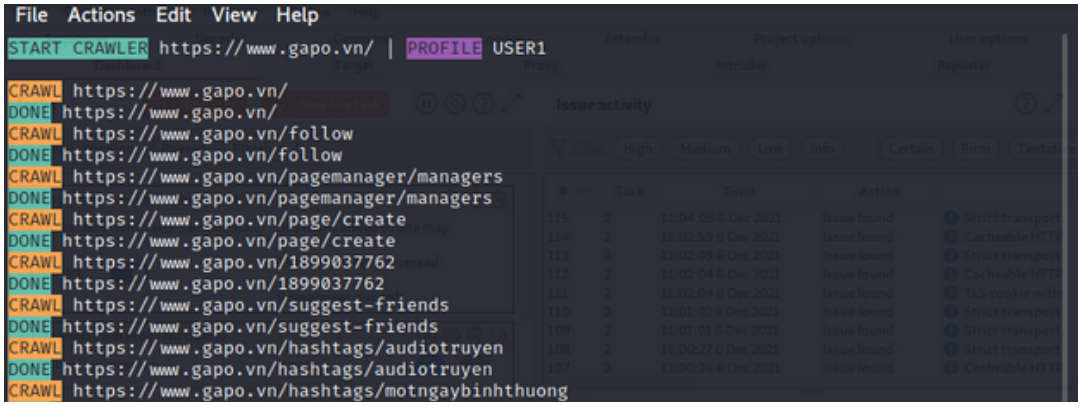


Figure 6.1c. Crawling www.gapo.vn

- Perform crawling on an e-commerce web application: ***www.shopee.vn***
  - Total runtime: 12 hours 33 minutes (Banned IP)
  - Result: 1183 links

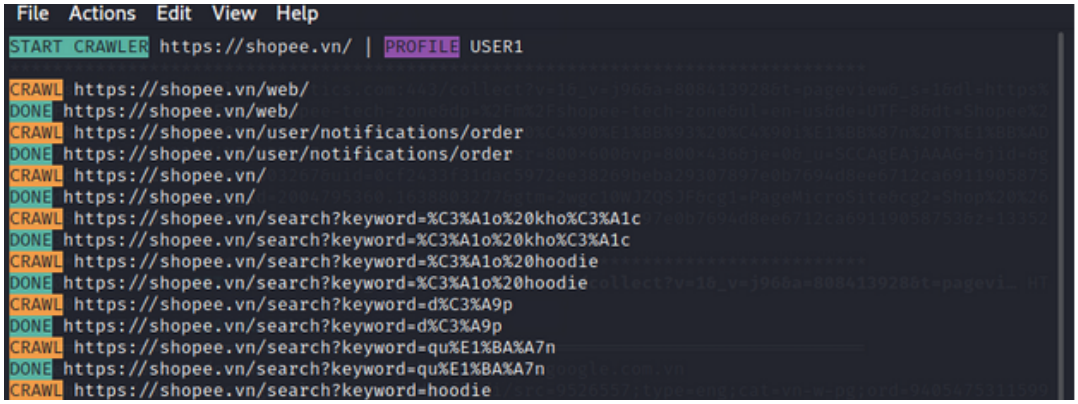


Figure 6.1d. Crawling www.shopee.vn

- Perform crawling on a news web application: ***www.kenh14.vn***
    - Time crawl: 16 hours 04 minutes
    - Result: 993 links

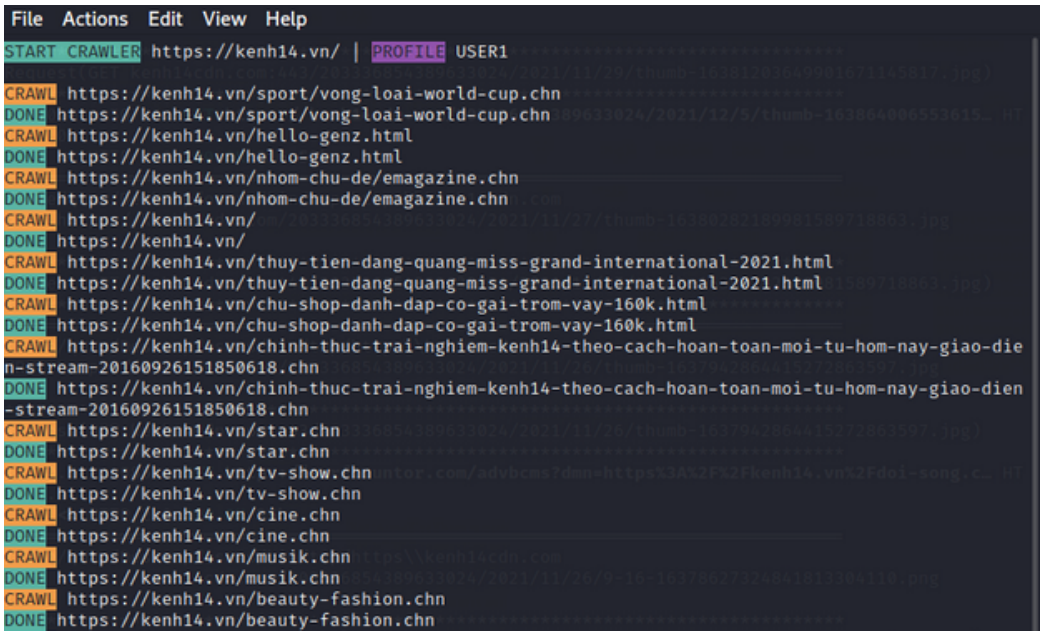


Figure 6.1e. Crawling www.kenh14.vn

- Performing crawling on an online forum web application: ***www.stackoverflow.com***
    - With user authentication:
    - Time crawl: 10 hours 28 minutes
    - Result: 850 link (Banned IP)

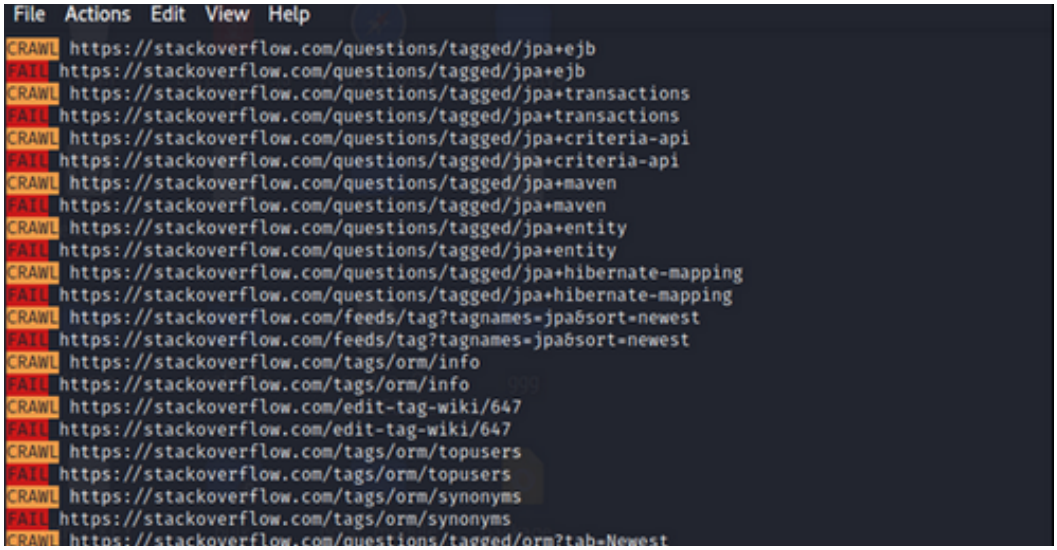


Figure 6.1f. Crawling www.stackoverflow.com (with authen)

- Without authentication:

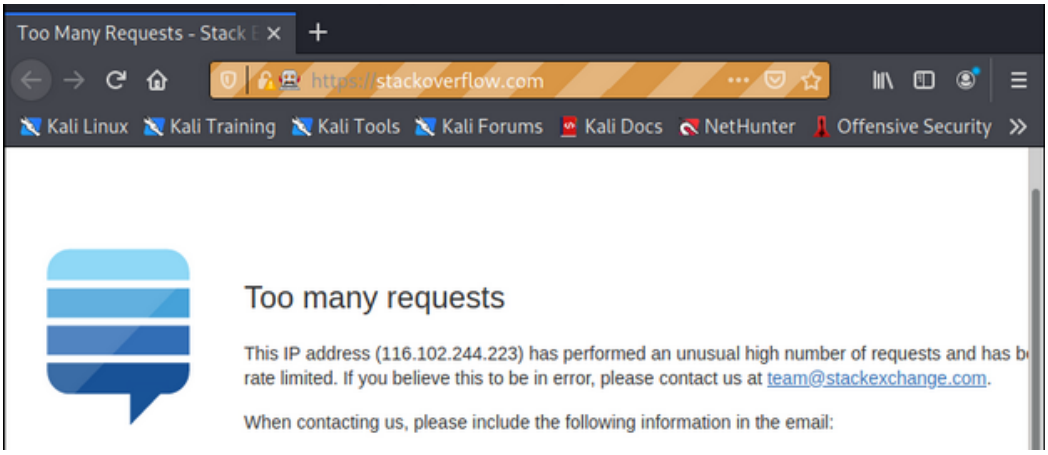


Figure 6.1g. Crawling www.stackoverflow.com (without authen)

- Performing crawling on a forum web application: ***www.whitehat.vn/***
  - With user authentication:
  - Time crawl: 6 hours 40 minutes
  - Result: 541 link (Banned IP)

Figure 6.1h. Crawling www.whitehat.com

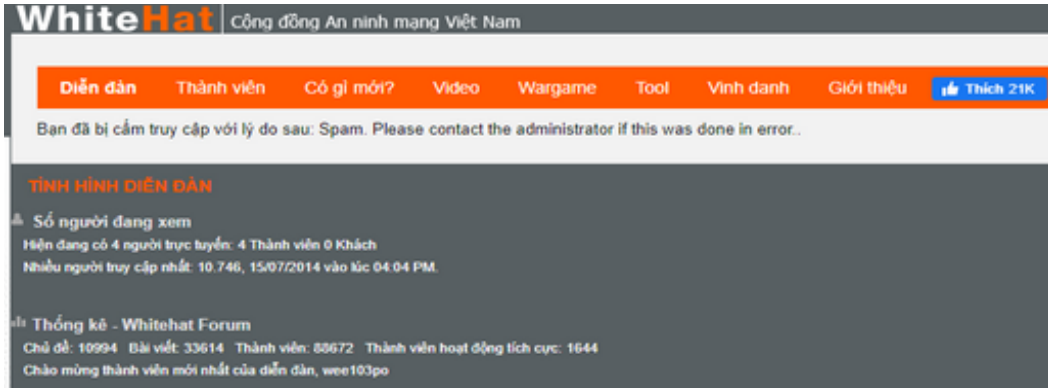


Figure 6.1i. Crawling www.whitehat.com - Error

6.2. Vulnerability scanning results

One of the biggest problems of OSWAS is that it takes a lot of time crawling, which can take up to 24 hours per user profile when crawling a big web application. Therefore, we could not manage to test the scanning phase in any real websites.

The result shown below is tested on our testing web application:

- Performing scan (vulnerabilities testing):

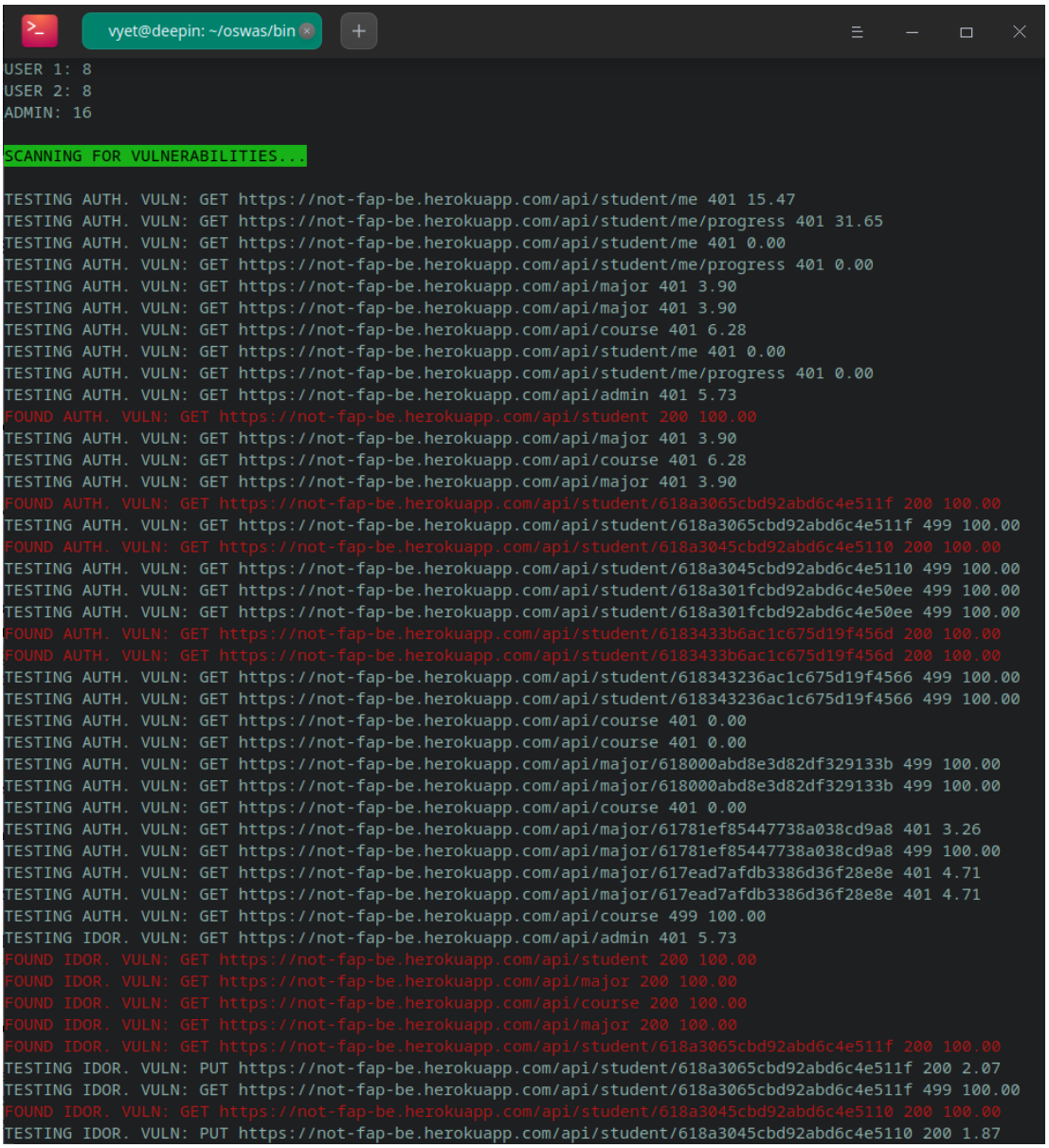
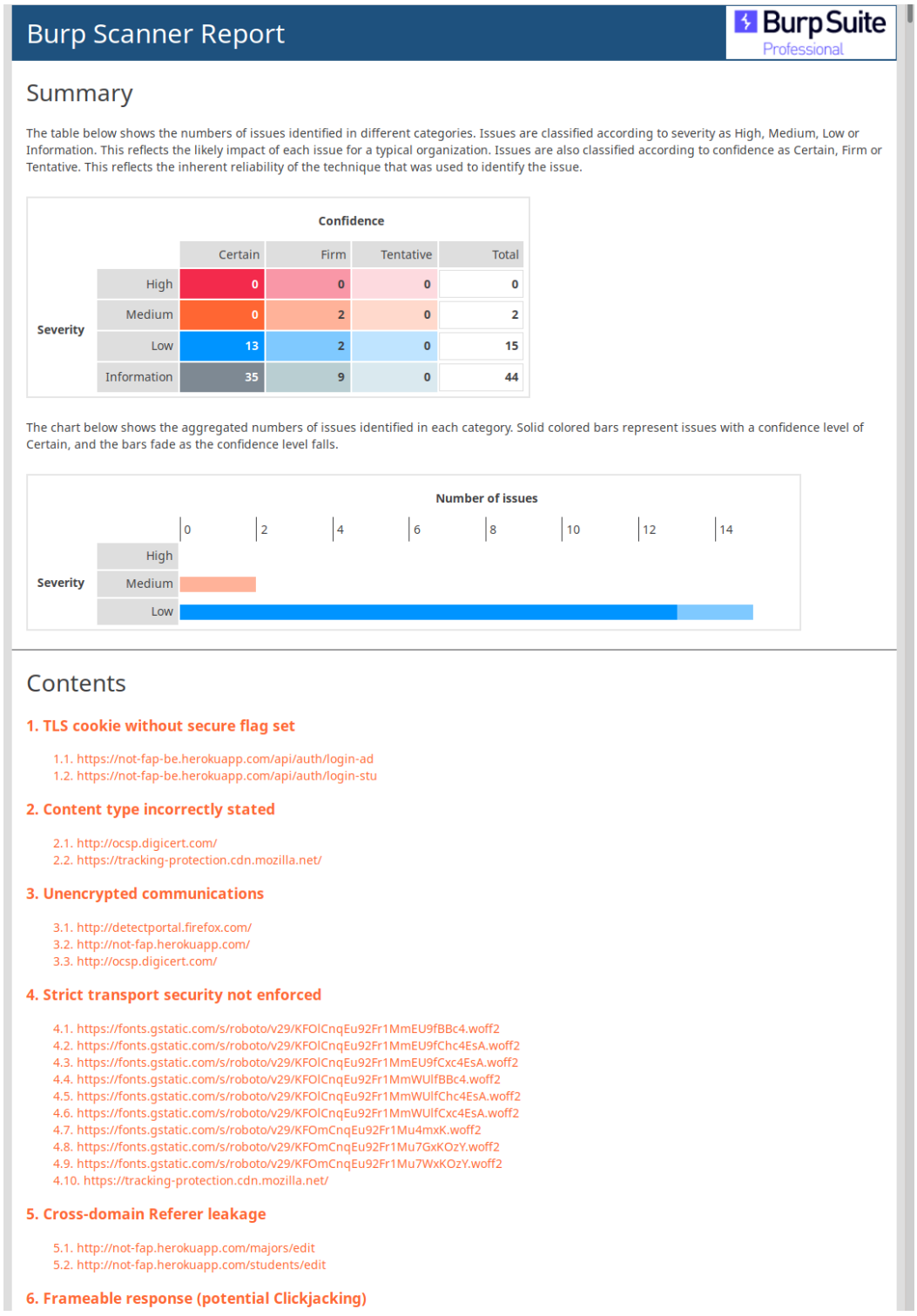


Figure 6.2a. Vulnerability scanning in test environment

- Testing results (Generated Burp Suite Scanner Report):



Burp Scanner Report

Burp Suite Professional

Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low or Information. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

Severity	Confidence			
	Certain	Firm	Tentative	Total
High	0	0	0	0
Medium	0	2	0	2
Low	13	2	0	15
Information	35	9	0	44

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.

Contents

Figure 6.2b. Vulnerability scanning results (Burp Suite Scanner)

- Testing results (Generated OSWAS Report):

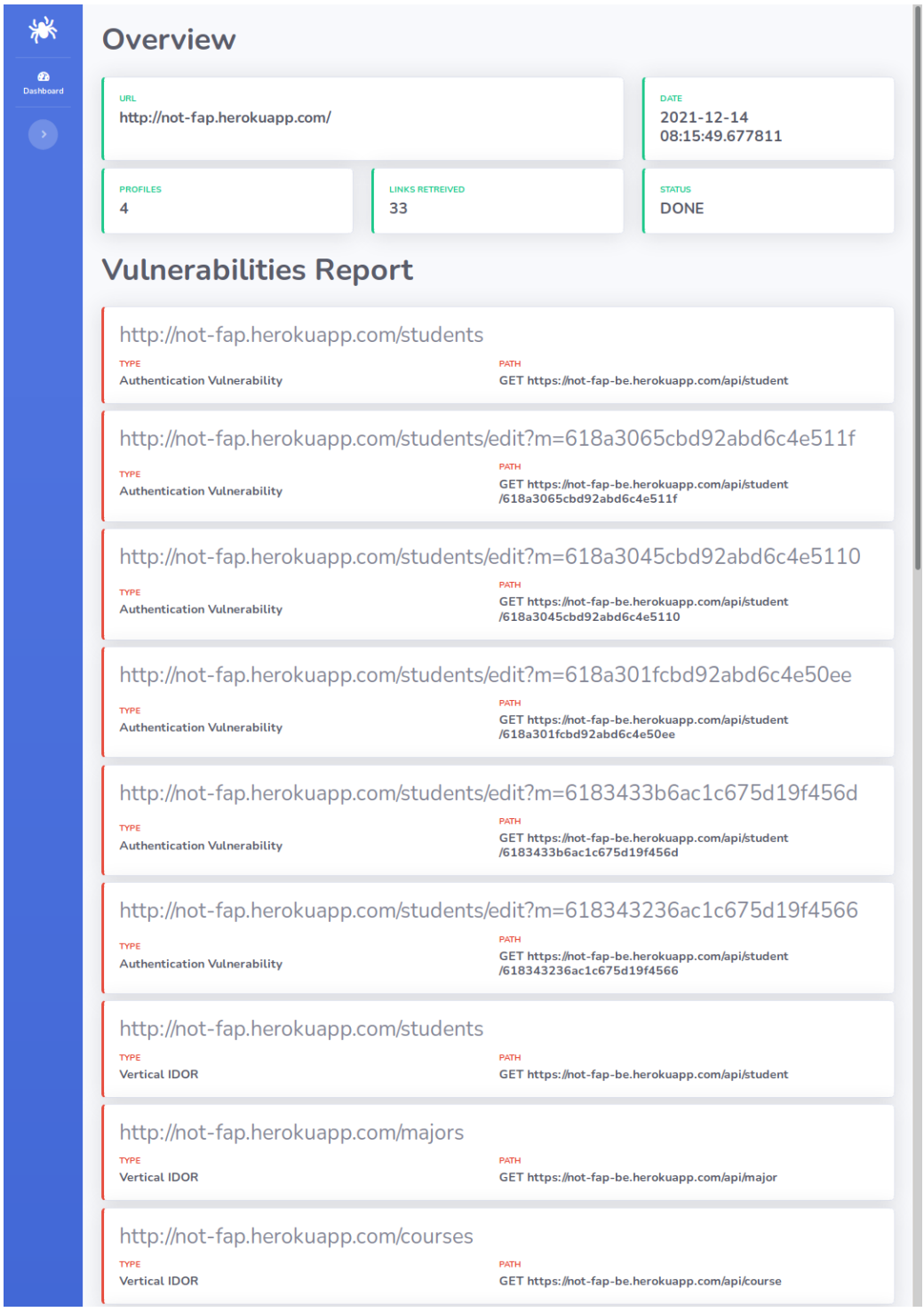


Figure 6.2c. Vulnerability scanning results (OSWAS)

6.3. Comparing to other softwares

6.3.1. OSWAS vs crawlers

- **Successfully crawl protected areas**:

  - Most open source crawlers we tested showed that they were unable to crawl protected areas. Some provided authentication functionality, but only with a few authentication methods.
  - Our method of using Selenium combined with letting the user authenticate manually proves to work well. Through the test cases, OSWAS was able to perform crawling authentication-required pages after user authentication.

- **Successfully crawl client-side rendered websites**:

  - When testing with our testing WebApp (not-fap.herokuapp.com), all the crawlers we tested failed to work.
  - As we used Selenium to crawl by accessing the DOM tree, OSWAS worked beautifully and was able to crawl the testing WebApp.

6.3.2. OSWAS vs Burp Suite Scanner

6.3.2.1. Handling *IDOR Vulnerability*

- In Burp Suite, to check for IDOR vulnerabilities, the user needs to manually intercept the request in the "Proxy" tab and then send the request to the "Intruder" tab.

- OSWAS successfully performed testing for the IDOR *Vulnerability* automatically (both *Horizontal IDOR* and *Vertical IDOR*).

Results from testing on our test environment:

- Test case scenarios:

The testing web application has APIs for listing all students and viewing a student's information. Only admin accounts are supposed to be able to access this API, while the student accounts are not. We edited the related APIs so that while they still require authentication, they do not check for the authorization of the requesting user. This basically means that all the users in the system, either admins or

students, can view all students' information, which causes a Vertical IDOR Vulnerability.

The testing web application also has an API for viewing a student's own profile, which is the *about me page*. All student accounts have access to this API, but a student account is supposed to only view his profiles. However, this API checks the request's token to identify which student account is requesting, so if we simply swap student A's token with student B's token, we can view student B's profiles. This causes a Horizontal IDOR vulnerability, where users on the same privilege level can access each other's assets.

- Test results:

OSWAS shows in its report the identified IDOR vulnerabilities in multiple links, such as not-fap-be.herokuapp.com/api/major/:id, not-fap-be.herokuapp.com/api/student/me/progress and not-fap-be.herokuapp.com/api/student/me

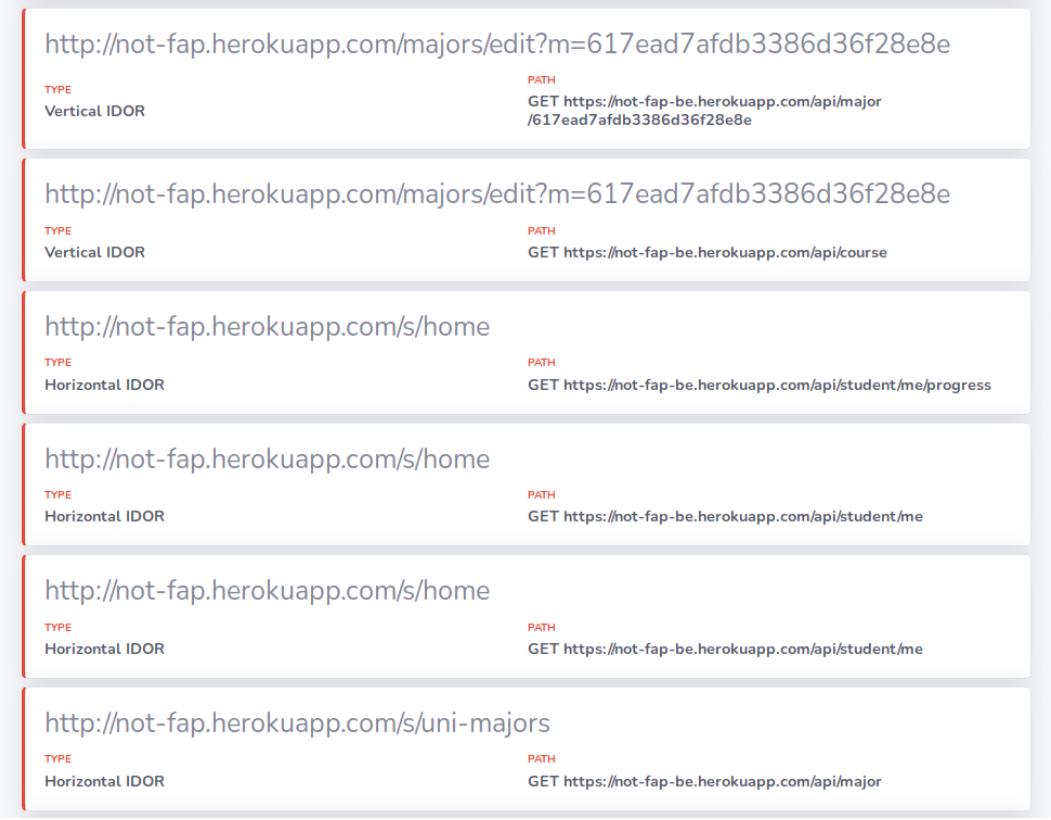


Figure 6.4.1. IDOR Vulnerability test results

6.3.2.2. Handling *Authentication Vulnerability*

- In Burp Suite, users have to manually run tests for this vulnerability by finding the authentication-required URLs, sending their request to the *repeater* tab, removing the authentication headers/signatures then analyzing the responses.

- OSWAS adopted the same methodology as above but performed it automatically.

Results from testing on our test environment:

- Test case scenario:

The web application has an API for viewing students' information. Only admin accounts are supposed to be able to access this API. We removed the authentication step in this API, which causes an Authentication Vulnerability that anyone can access this API and view any student's information.

- Test results:

OSWAS report shows the Authentication Vulnerabilities in multiple links, such as not-fap-be.herokuapp.com/api/student *and* not-fap-be.herokuapp.com/api/student/:id

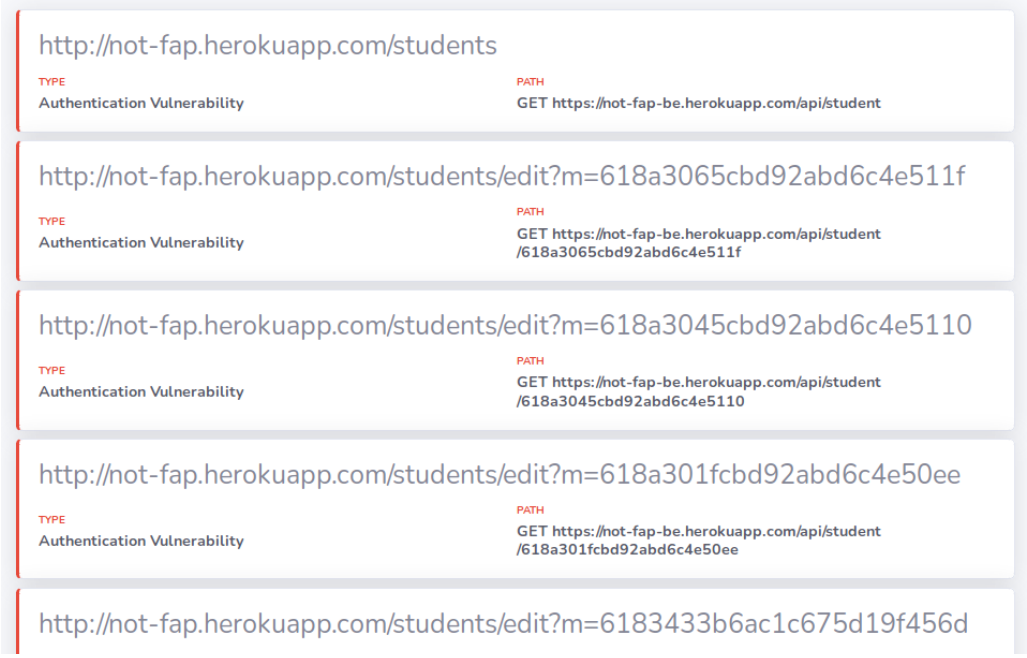


Figure 6.4.2. Authentication Vulnerability test results

7. Conclusion

7.1. Conclusion

Nowadays, web development is an extremely wide area. Building a crawler and scanner that is smart and detailed enough to cover all types of web applications is a highly sophisticated and complex process as each web application is built differently.

Our solution, OSWAS, mainly focuses on solving some major problems that crawlers and scanners have not tackled yet, which are crawling protected areas and scanning Authentication Vulnerability and IDOR Vulnerability. Until now we have managed to successfully complete developing and fully testing our tool.

However, OSWAS was only built in a short amount of time, therefore it still has several flaws and unimplemented functionalities that we have not managed to tackle. However, we find that OSWAS provides some promising ideas that solves some difficult problems of the current crawlers and scanners. With the implementation of new ideas and technologies in future development, OSWAS still has a lot of game-changing offers to the playground of Web Crawlers and Scanners

7.2. Limitations

1. Time & resource consuming:

In the current state of development, OSWAS usually takes up to 12-24 hours when crawling an average-size web application (E.g. *www.kenh14.vn* takes 17+ hours to crawl).

Solution: Implement Python multithreading and Selenium Grid to increase performance.

2. Haven't fully resolved all test cases of multipart/form-data HTTP POST/PUT request:

The vulnerability testing step requires us to parse HTTP request body from a file to Python object. Currently, OSWAS can only identify request parameters and request body data in JSON form.

Solution: By researching or developing another tool or library may help us identify and parse other types of request body (E.g. form-data).

3. Exception dictionaries are not broad enough:

Exception dictionaries take time and trials to update gradually.

Solution: Building these dictionaries requires more time analyzing the web applications. With regular updates and research, we will be able to create complete dictionaries.

4. Only support Linux and Firefox browsers:

Our initial plan was to be able to run OSWAS on both Windows and Linux and on multiple browsers. However, OSWAS is only tested and operates on Linux and Firefox.

Solution: In order for OSWAS to operate on multiple platforms and browsers, we need more time to research, develop and test.

5. Perform inefficiently on large scale Web Applications

Unable to perform the complete process on large scale Web Applications (Facebook, Youtube, StackOverflow, etc.) due to the Web-Apps' request blockage and the server's lack of time and resources..

7.3. Future Updates

OSWAS is currently only in the very first steps of development. It has a lot of remaining drawbacks, flaws that need to be updated and features that we have not implemented yet. In the future, in order to upgrade our solution, we would propose some ideas and features:

- Support multiple operating systems (Windows, MacOS) and web browsers (Chrome, Edge, IE, Opera).
- Implement Python multithreading and Selenium Grid to perform crawling with multiple browsers at the same time.
- Upgrade the proxy in order to be able to capture and identify links through the JavaScript actions (E.g. clicking buttons).
- Implement Machine Learning to optimize the scanning process by identifying the vulnerabilities smarter and faster.
- Implement code quality & security analyzing tools (E.g. SonarQube) for better product quality.

References

1. Julie Peterson. Apr 20, 2021. The Top 11 Web Vulnerability Scanners. https://www.whitesourcesoftware.com/resources/blog/the-top-web-vulnerability-scanners/

2. Mayra Cortes. Feb 27, 2020. Vulnerability Scanning: Pros, Cons & Best Practices. https://www.cynexlink.com/2020/02/27/pros-and-cons-of-vulnerability-scanning/

3. Ian Muscat. Aug 17, 2017. The difference between Vulnerability Assessment and Penetration Testing. The Acunetix Blog https://www.acunetix.com/blog/articles/difference-vulnerability-assessment-penetration-testing/

4. Stages of Scanning. Netsparker. https://www.netsparker.com/support/stages-scanning/

5. Sonali Saikia. July 17, 2021. Java to Python: A Paradigm Shift in Automated Testing with Selenium. https://www.techvariable.com/blog/java-to-python-a-paradigm-shift-in-automated-testing-with-selenium/

6. Amal Shaji. Dec 23rd, 2020. Web Authentication Methods Compared. https://testdriven.io/blog/web-authentication-methods/

7. How to Crawl a Protected Website: In-Depth Look (November 11, 2015). Redwerd. https://redwerk.com/blog/in-depth-look-at-automated-crawling-of-website-protected-areas/

8. Testing for Bypassing Authentication Schema. OWASP. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/04-Testing_for_Bypassing_Authentication_Schema

9. Insecure direct object references (IDOR). PortSwigger. https://portswigger.net/web-security/access-control/idor

10. Using Burp to Test for Insecure Direct Object References. PortSwigger. https://portswigger.net/support/using-burp-to-test-for-insecure-direct-object-references

11. Crawljax, a dynamic (JavaScript-based) Web Applications automatic crawler. https://github.com/crawljax/crawljax

12. Nikhil Jaiswal, Apr 15, 2019. SequenceMatcher in Python. https://towardsdatascience.com/sequencematcher-in-python-6b1e6f3915fc