

**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY**  
**UNIVERSITY OF INFORMATION TECHNOLOGY**  
**FACULTY OF COMPUTER NETWORKS**  
**AND COMMUNICATIONS**

**NGUYEN DINH KHA**

**SPECIALIZED PROJECT REPORT**

**An Advanced Framework for Web Defensive Deception Using  
Reinforcement Learning**

**TP. HỒ CHÍ MINH, 2023**

**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY**  
**UNIVERSITY OF INFORMATION TECHNOLOGY**  
**FACULTY OF COMPUTER NETWORKS**  
**AND COMMUNICATIONS**

**NGUYEN DINH KHA – 20520562**

**SPECIALIZED PROJECT REPORT**

**An Advanced Framework for Web Defensive Deception Using  
Reinforcement Learning**

**PROJECT SUPERVISOR**

Mas. DO HOANG HIEN

**TP. HỒ CHÍ MINH, 2023**

## **ACKNOWLEDGEMENTS**

Dear Mr. Do Hoang Hien,

I would like to express my deepest gratitude to you for your dedicated guidance, invaluable knowledge, and continuous support throughout my project on 'A Web-based Deception Strategy Framework Using Reinforcement Learning'. Your care and patience have not only helped me successfully complete this project but have also taught me many valuable lessons for my academic career and future professional life. Please accept my sincere and utmost appreciation.

Kind regards,

Nguyen Dinh Kha.

# MỤC LỤC

Chương 1. Giới thiệu.....	3
1.1. Tổng quan về an ninh mạng và mô phỏng Web .....	3
1.2. Mục tiêu của đề án.....	3
1.3. Phương pháp nghiên cứu .....	3
Chương 2. Cơ sở lý thuyết và các nghiên cứu liên quan.....	4
2.1. Cơ sở lý thuyết.....	4
2.1.1. An toàn Mạng: Đảm bảo tính toàn vẹn, bí mật và tính sẵn có trong không gian mạng .....	4
2.1.2. Lừa dối Phòng thủ trong An ninh mạng: Chiến lược và Tác động Tâm lý      4	
2.1.3. Học Tăng cường: Thích ứng và Phát triển Hệ thống Phòng thủ An ninh mạng      4	
2.1.4. Tổng quan về HoneyHTTPD.....	6
2.1.5. Tích hợp Học Tăng cường với HoneyHTTPD .....	6
2.1.6. Mô phỏng Trang Đăng nhập DVWA .....	6
2.2. So sánh nghiên cứu của mình với các tác giả khác. ....	7
Chương 3. Phân tích thiết kế hệ thống .....	7
3.1. Định nghĩa vấn đề.....	7
3.2. Phương pháp thực hiện .....	7
3.3. Kiến trúc tổng thể .....	8
3.4. Kiến trúc của từng thành phần.....	9
Chương 4. Hiện thực hệ thống .....	11

Chương 5. Thực nghiệm và đánh giá.....	17
Chương 6. Kết luận và hướng phát triển.....	23

## **DANH MỤC TỪ VIẾT TẮT**

RL – Reinforcement Learning

DVWA – Damn Vulnerable Web Application

SSL / TLS – Secure Sockets Layer/Transport Layer Security

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

## ABSTRACT

In the ever-changing landscape of cybersecurity, deceptive defense strategies have become crucial for protecting web applications from malicious activities. This paper introduces an innovative framework named 'A Web-based Deception Strategy Framework Using Reinforcement Learning (RL),' leveraging the combination of web server deception techniques and the adaptive intelligence of Reinforcement Learning (RL). The aim of this defensive framework is to enhance the security of web applications by deceiving and thwarting potential attackers through a fake HoneyHTTPD server mimicking the login page of the Damn Vulnerable Web Application (DVWA).

The methodology includes integrating an RL model with HoneyHTTPD, a customizable and deceptive HTTP server. The RL model is meticulously designed to analyze incoming HTTP requests and determine optimal deceptive responses, with the goal of diverting and analyzing unauthorized access attempts. The server configuration is adjusted to simulate the DVWA login page, creating a convincing target for attackers while protecting actual web applications.

Results demonstrate the effectiveness of the defensive framework in adapting in real-time to various attack strategies. By utilizing the RL model, the HoneyHTTPD server successfully deceives and logs unauthorized access attempts, providing valuable insights into attack patterns. Moreover, the adaptability of the RL model ensures continuous learning and improvement in deception strategies, significantly enhancing the server's capability to mitigate potential threats over time.

This study contributes to the field of cybersecurity by proposing a novel method that combines defensive deception principles with the self-learning capabilities of RL. The framework offers a proactive defense mechanism, becoming a valuable asset for organizations looking to enhance the protection of their web applications against the increasingly sophisticated cyber threats. The integration of RL with HoneyHTTPD opens new directions in developing more complex and resilient web cybersecurity systems

## **Chapter 1. Introduction**

### **1.1. Overview of Cybersecurity and Web Simulation**

The current cybersecurity landscape is marked by increasingly sophisticated cyber attacks, against which traditional defensive measures, relying on fixed and predetermined rules, often fail to effectively counter. This has led to an urgent need for more flexible and adaptive security strategies. Web simulation, or honeypots, serves as a powerful tool in attracting and analyzing attacker behaviour, thereby enhancing the defensive capabilities of the system.

### **1.2. Project Objectives**

The primary objective of this project is to develop 'A Web-based Deception Strategy Framework Using Reinforcement Learning (RL)'. This defensive framework aims to integrate the adaptive learning capabilities of Reinforcement Learning (RL) with HoneyHTTPD, a customizable web server used to create a deceptive web environment. The goal is to create a system that not only deceives attackers but also learns from their tactics, thereby improving the defensive strategy over time. Honeypots simulate services or systems to deceive attackers, collecting information on the methods and purposes of the attack. This not only helps protect the actual system but also provides valuable data to enhance understanding of cybersecurity threats.

### **1.3. Research Methodology**

Our research methodology combines theoretical study and practical application. We start by investigating the current state of cybersecurity, identifying gaps in traditional defensive methods, and understanding the principles behind RL and deception techniques. The project includes the design and implementation of the framework, followed by rigorous testing under simulated attack scenarios. This allows us to assess the effectiveness of the defensive framework in a controlled environment and gather data for further refinement of our approach



## **Chapter 2. Theoretical Background and Related Studies**

### **2.1. Theoretical Basis**

#### **2.1.1. Cybersecurity: Ensuring Integrity, Confidentiality, and Availability in Cyberspace**

Cybersecurity is a broad field encompassing various technologies and methods aimed at protecting web services and users from cyber threats. It involves securing data transmission through encryption protocols such as SSL/TLS, ensuring data integrity and confidentiality, and system availability. Cybersecurity also extends to robust authentication and authorization methods, playing a critical role in verifying user identities and controlling access to resources. Moreover, it includes a proactive approach in vulnerability management, focusing on identifying and mitigating potential risks, such as SQL injection and cross-site scripting (XSS).

#### **2.1.2. Deceptive Defence in Cybersecurity: Strategies and Psychological Impact**

Deceptive defence, an innovative approach in cybersecurity, utilises tactics like honeypots and honeytokens - fake systems and data designed to attract attackers. These tools are not only meant to trap but also to study the attacker's methods. Besides the technical structure, the psychological aspect of deception plays a vital role. By creating an environment of uncertainty and misinformation, deceptive defence strategies can effectively disrupt and prevent attacks.

#### **2.1.3. Reinforcement Learning: Adapting and Developing Cybersecurity Defence Systems**

Reinforcement Learning (RL) represents a shift in how cybersecurity systems adapt and evolve. Unlike traditional algorithms, RL allows systems to learn and make decisions through trial and error. This learning process, driven by a system of rewards and punishments, enables RL algorithms to adapt flexibly to new threats. In cybersecurity, deploying RL can make defence mechanisms more resilient, flexible, and intelligent, continuously learning from interactions with potential threats to

enhance their protective capabilities.

RL algorithms, such as Q-learning and Deep Q-Networks, help these systems effectively evaluate the impact of actions and adjust responses accordingly. This means that, unlike static defence mechanisms, systems integrated with RL can evolve their strategies, becoming better at identifying and mitigating emerging threats. This adaptability is crucial in the ever-changing cybersecurity landscape, where attackers continually develop new methods to exploit vulnerabilities.

<b>Criteria</b>	<b>Traditional Deception</b>	<b>Deception using Reinforcement Learning</b>
<b>Flexibility</b>	Balanced, does not change over time	Adapts and learns continuously
<b>Response to threats</b>	Reacts based on pre-established configurations	Reacts based on learning and data
<b>Customizability</b>	Limited, requires manual configuration	High, automatically adjusts based on data
<b>Efficiency against new attacks</b>	Balanced, does not detect new attacks	Recognises and adapts to new attacks
<b>Deceptive Nature</b>	Based on balanced scenarios	Dynamic, changes based on learned scenarios
<b>Management and Maintenance</b>	Requires regular intervention	Automated, requires less intervention

Table 2: Comparison between traditional deception methods and deception using Reinforcement Learning

Integrating RL with defensive techniques like HoneyHTTPD opens new horizons in the field of cybersecurity. This approach enables the creation of security systems that are more flexible, responsive, and intelligent. These systems do not merely react to threats; they learn from them, continually evolving to offer more effective protection against the complex and ever-changing landscape of cyber threats.

#### **2.1.4. Overview of HoneyHTTPD**

HoneyHTTPD is an HTTP server designed to simulate vulnerable websites and services, acting as a honeypot. It offers high customisability, allowing users to configure fake webpages and record the behaviour of attackers. HoneyHTTPD's strength lies in its ability to mimic real websites, thereby attracting attackers while capturing essential information for analysing and improving security systems.

#### **2.1.5. Integrating Reinforcement Learning with HoneyHTTPD**

The Reinforcement Learning (RL) model is integrated into HoneyHTTPD to enhance its ability to detect and counter threats. The RL model learns from each attack, automatically adjusting its strategy to optimize protection.

The integration process involves developing and training the RL model, as well as establishing an interface between the model and HoneyHTTPD to ensure continuous and accurate information exchange between the two systems.

#### **2.1.6. Simulating the DVWA Login Page**

HoneyHTTPD is configured to simulate the login page of the Damn Vulnerable Web Application (DVWA), a web application prone to attacks. The goal is to create an attractive and realistic environment to lure attackers. Details on configuring, designing the login page, and how HoneyHTTPD handles incoming requests and its responses.

## **2.2. Comparing My Research with Other Authors**

The comparison between the research using HoneyHTTPD combined with Reinforcement Learning (RL), and the honeypot deception system presented in the 2018 study by A. El-Kosairy and M. A. Azer, "A New Framework for Web Deception Systems Integrated with Reinforcement Learning." Your project's use of RL along with HoneyHTTPD allows for more dynamic and adaptable responses to cyber threats compared to the traditional honeypot method discussed by El-Kosairy and Azer.

**Learning and Integration:** The RL component in the study enables the system to learn from interactions with attackers and evolve its deception strategies, offering a level of adaptability not found in traditional honeypots.

**Deception Techniques:** While both methods utilise deception, integrating RL into HoneyHTTPD can provide more complex and less predictable deception techniques compared to standard honeypot systems.

## **Chapter 3. System Design Analysis**

### **3.1. Problem Definition**

The main challenge our project addresses is the increasing complexity of web-based attacks and the inadequacy of traditional security measures in actively deceiving and minimizing these threats. The issue lies in detecting and responding to malicious activities in a manner that not only protects the system but also gathers valuable intelligence about attack methods.

### **3.2. Implementation Method**

To address this, we have developed an advanced technology framework for web deception defenses using Reinforcement Learning (RL), integrated with HoneyHTTPD.

This method involves creating a simulated deceptive environment of real web services to attract potential attackers while using RL to flexibly adapt deception strategies based on attackers' behaviors. The aim is to deceive attackers, gather

intelligence, and enhance defensive mechanisms.

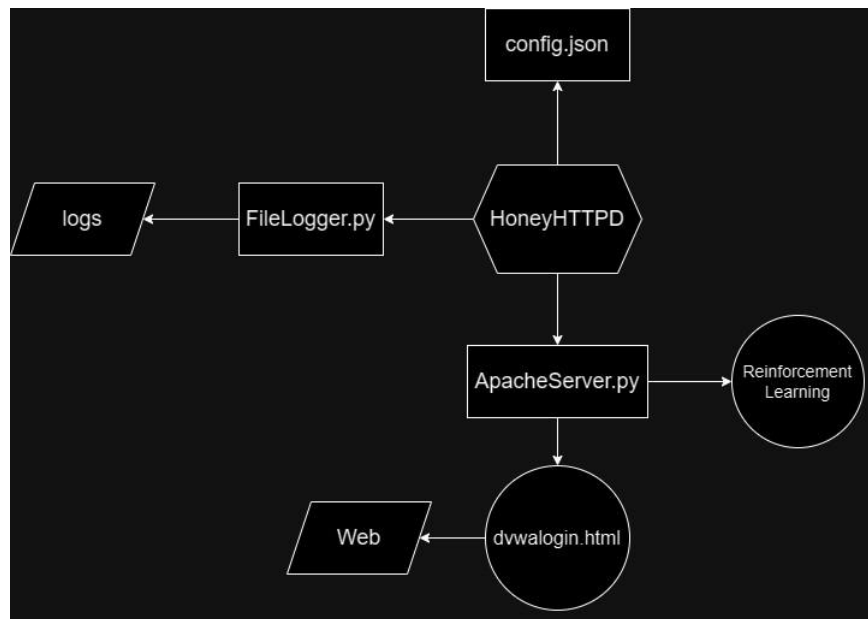
### 3.3. Overall Architecture

The system architecture is designed in a multi-layered framework:

**Deception Layer (HoneyHTTPD):** This layer acts as the first line of defense, creating a deceptive façade to lure potential attackers. It simulates various web services and vulnerabilities to attract attackers.

**Learning and Adaptation Layer (Integrated RL):** Here, RL algorithms analyze interactions of attackers with the deception layer, learning from their behaviors and patterns. This information is used to flexibly adjust deception strategies.

**Control and Command Layer:** This central layer monitors all activities, coordinates between the deception layer and the learning layer, and manages responses based on gathered intelligence.



## Overall Architecture of HoneyHTTPD Model

### 3.4. Architecture of Each

#### **HoneyHTTPD Component:**

Designed to create customizable, realistic web services. Capable of logging detailed information about attacker activities. Key files within HoneyHTTPD include:

The config.json file defines configurations for the HoneyHTTPD server. Key elements include: Loggers (Configurations for logging, with FileLogger.py specifying logging to standard output.), Servers (An array of server configurations, including settings like handler type, mode, port, domain, and timeout, where ApacheServer.py is the main server module), User/Group (Specifies user and group permissions.)

HoneyHTTPD will use the dvwlogin.html file as the simulated login page of DVWA. After HoneyHTTPD receives login data from attackers, it will be logged into a file.

#### **Reinforcement Learning Module:**

Uses a set of algorithms (such as Q-learning or Deep Q-Networks) to process data from HoneyHTTPD. Responsible for making decisions on how to adjust deception strategies. Integrating Reinforcement Learning (RL) into HoneyHTTPD requires a structured approach, where RL influences decision-making in handling HTTP requests and server responses. Integration should focus on learning from interactions and improving server responses over time.

Considerations for RL Model:

**State Representation:** Determine the state space to represent features of relevant HTTP requests for decision-making.

**Action Space:** Define actions that the server can take to respond to different states (e.g., responding with different HTTP status codes, redirecting, disconnecting).

**Reward Mechanism:** Design a reward system to reinforce desired outcomes (e.g., detecting malicious behavior, providing accurate content).

**Integration within Server Layer:**

```
# Implement RL decision making in the GET request handler
def on_GET(self, path, headers):
    state = self.create_state_from_request(path, headers)
    action = self.rl_model.predict_action(state)

    if action == 'dvwa_login.html':
        return self.serve_login_page()
    else:
        return super().on_GET(path, headers)

# Implement RL decision making in the POST request handler
def on_POST(self, path, headers, post_data):
    state = self.create_state_from_request(path, headers, post_data)
    action = self.rl_model.predict_action(state)

    if action == 'HANDLE_LOGIN':
        return self.handle_login_attempt(post_data)
    else:
        return super().on_POST(path, headers, post_data)
```

In the server layer, integrate the RL model with request processing, particularly for GET requests to serve the login page and POST requests to submit forms.

```
# Add to the Server class
def log(self, client, request, response, extra={}):
    extra['rl_state'] = self.create_state_from_request(...)
    extra['rl_action'] = self.rl_model.last_action
    super().log(client, request, response, extra)
```

Modify logging mechanisms to include data about states and actions chosen by the RL model.

**Management Dashboard:**

Provide a user interface for monitoring and control. Allow for manual overrides and customization of deception scenarios.

## Chapter 4. System Implementation

System Configuration Setup Steps:

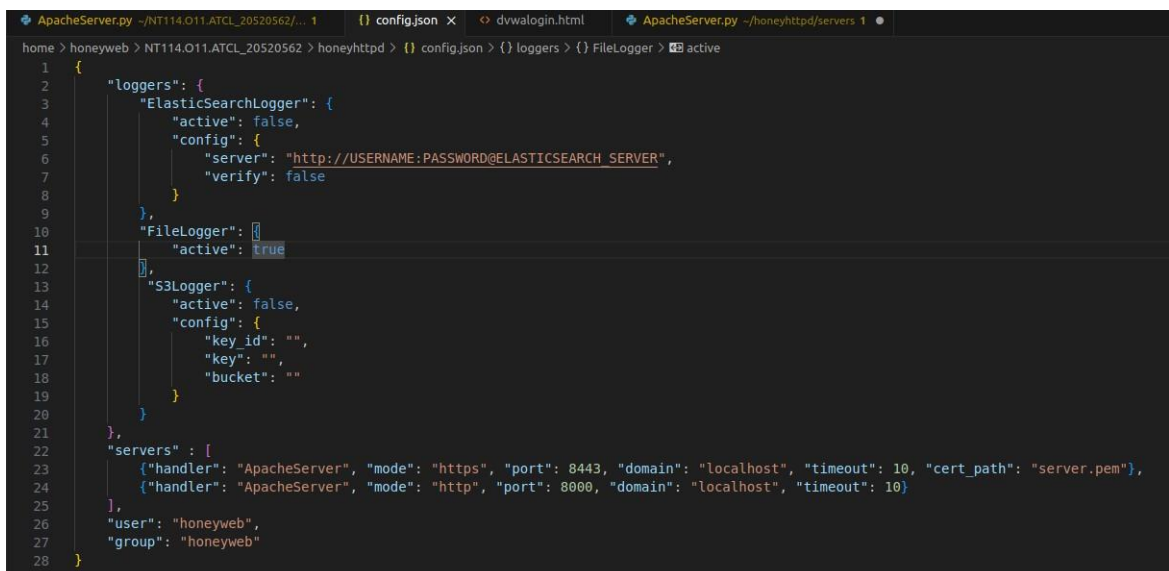
Firstly, we need to edit the config.json file, one of the crucial files when running HoneyHTTPD.

Parameter	Description	Example Value
handler	The type of server used	ApacheServer
mode	Network protocol (HTTP/HTTPS)	https
port	Network port that the server listens to	8443 (HTTPS), 8000 (HTTP)
domain	Domain name used by the server	localhost
timeout	Maximum waiting time for each request	10 seconds
cert_path (HTTPS only)	Path to the SSL certificate (for HTTPS)	server.pem
loggers	The loggers have been activated and configured	FileLogger, ElasticSearchLogger, S3Logger



active (trong loggers)	The status of logger activation	true/false
config (trong loggers)	Detailed configuration for each type of logger	Server address, authentication details, etc.
user	Người dùng hệ thống dưới đó máy chủ được chạy	honeyweb
group	Nhóm hệ thống dưới đó máy chủ được chạy	honeyweb

Table 2: Detailed parameters and configurations of the config.json file in HoneyHTTPD



```

1  {
2    "loggers": {
3      "ElasticSearchLogger": {
4        "active": false,
5        "config": {
6          "server": "http://USERNAME:PASSWORD@ELASTICSEARCH_SERVER",
7          "verify": false
8        }
9      },
10     "FileLogger": {
11       "active": true
12     },
13     "S3Logger": {
14       "active": false,
15       "config": {
16         "key_id": "",
17         "key": "",
18         "bucket": ""
19       }
20     }
21   },
22   "servers": [
23     {"handler": "ApacheServer", "mode": "https", "port": 8443, "domain": "localhost", "timeout": 10, "cert_path": "server.pem"},
24     {"handler": "ApacheServer", "mode": "http", "port": 8000, "domain": "localhost", "timeout": 10}
25   ],
26   "user": "honeyweb",
27   "group": "honeyweb"
28 }

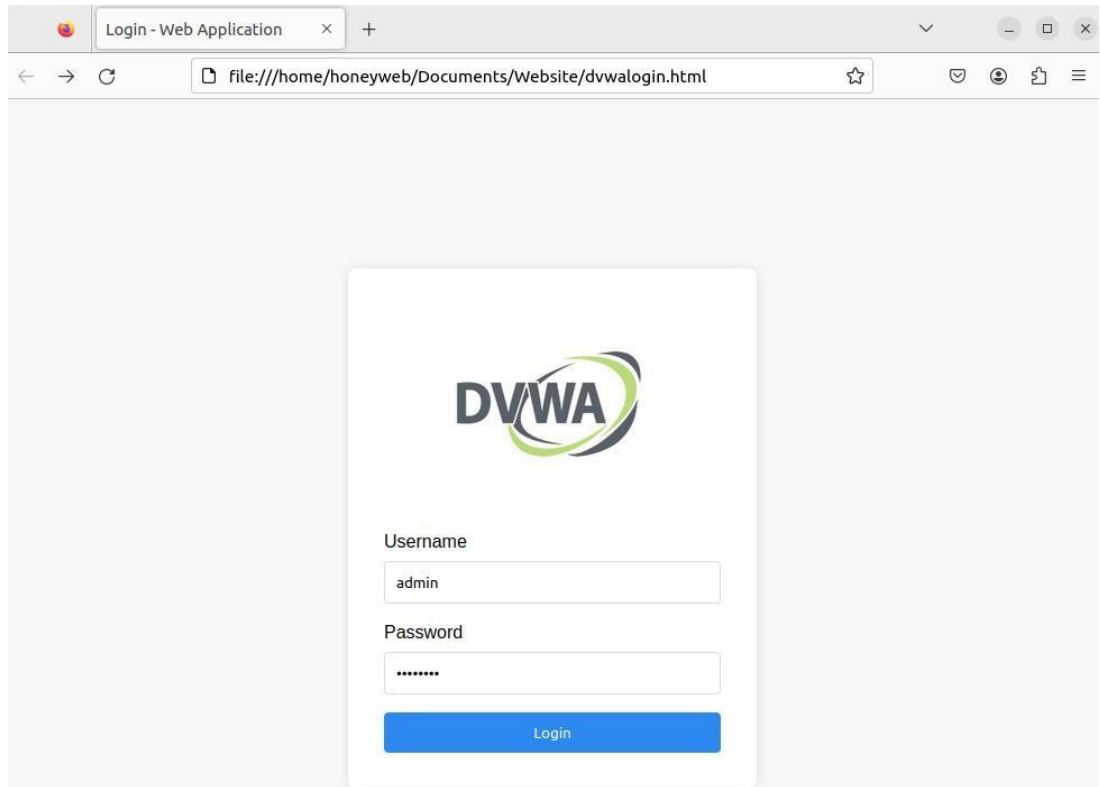
```

### Code snippet of the config.json configuration file

The config.json file configures HoneyHTTPD to use both HTTP and HTTPS, with an ApacheServer module handler set up for each protocol, corresponding to ports 8000 and 8443. It also specifies loggers, with FileLogger active, and the default domain as localhost.

To use HTTPS mode, we need to initialize an SSL certificate, a matter of creating an SSL certificate for the web server, helping encrypt communication between the server and clients. SSL certificates are also known as HTTPS certificates, and they help protect users' privacy and security.





Mock login page interface of DVWA

dvwalogin.html

x

home

>

honeyweb

>

Documents

>

Website

>

dvwalogin.html

>

html

>

head

>

style

>

.login-form label

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Login - Web Application</title>
7      <style>
8          body {
9              font-family: 'Arial', sans-serif;
10             background: #f7f7f7;
11             display: flex;
12             justify-content: center;
13             align-items: center;
14             height: 100vh;
15             margin: 0;
16         }
17         .login-container {
18             padding: 2rem;
19             background: #ffffff;
20             box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
21             border-radius: 8px;
22             width: 300px;
23             text-align: center;
24         }
25         .login-container img {
26             width: 80%;
27             margin-bottom: 1rem;
28         }
29         .login-form {
30             display: flex;
31             flex-direction: column;
32         }
33         .login-form label {
34             text-align: left;
35             margin-bottom: 0.5rem;
36         }

```

```

37     .login-form input[type='text'],
38     .login-form input[type='password'] {
39         padding: 10px;
40         margin-bottom: 1rem;
41         border: 1px solid #ddd;
42         border-radius: 4px;
43         box-sizing: border-box;
44     }
45     .login-form input[type='submit'] {
46         padding: 10px;
47         border: none;
48         border-radius: 4px;
49         background-color: #2d89ef;
50         color: white;
51         cursor: pointer;
52         transition: background-color 0.2s;
53     }
54     .login-form input[type='submit']:hover {
55         background-color: #2b5797;
56     }
57 </style>
58 </head>
59 <body>
60     <div class="login-container">
61         <!-- Placeholder for logo-->
62         
63         <form class="login-form" action="/post_login" method="post">
64             <label for="username">Username</label>
65             <input type="text" id="username" name="username">
66             <label for="password">Password</label>
67             <input type="password" id="password" name="password">
68             <input type="submit" value="Login">
69         </form>
70     </div>
71 </body>
72 </html>

```

Figure 2.2: Code snippet for the mock login page interface of DVWA

```

# Called on GET requests. Return ERROR_CODE, HEADERS, EXTRA
def on_GET(self, path, headers):
    if path == "/" or path == "/login":
        with open('/home/honeyweb/Documents/Website/dvwa/login.html', 'r') as file:
            html_content = file.read()
            return 200, [("Content-Type", "text/html; charset=utf-8")], html_content

    else:
        return 404, [("Content-Type", "text/html; charset=utf-8")], "<html><body>Not Found</body></html>"

```

The `on_GET` method in the `ApacheServer` class serves HTML content when the login page URL is requested.

```

def on_POST(self, path, headers, post_data):
    if path == "/post_login":
        # Process the login credentials
        username, password = self.parse_credentials(post_data)
        # Simulate an authentication process
        if self.authenticate(username, password):
            return 200, [("Content-Type", "text/html; charset=utf-8")], "Login successful."
        else:
            return 401, [("Content-Type", "text/html; charset=utf-8")], "Login failed."
    else:
        # Handle other POST requests normally
        return super().on_POST(path, headers, post_data)

def parse_credentials(self, post_data):
    # Assuming post_data is a byte string of form-encoded data
    parsed_data = parse_qs(post_data.decode('utf-8'))
    # The parse_qs function returns a dictionary with the form field names as keys
    # and lists of values as values. We'll get the first value in the list for each key.
    username = parsed_data.get("username", [None])[0]
    password = parsed_data.get("password", [None])[0]
    return username, password

def authenticate(self, username, password):
    # Placeholder for actual authentication logic
    # For simulation purposes, check if the credentials match a hardcoded pair.
    return username == "admin" and password == "password"

```

In the `on_POST` method of the server class handling the login URL, the team will simulate the login authentication process. When the attacker logs in with the correct username and password, HoneyHTTPD will respond with "Login successful" to let the attacker know they have successfully accessed the website.

## Chapter 5. Experimentation and Evaluation

```
honeyweb@honeyweb-virtual-machine: ~/honeyhttpd/logs
honeyweb@honeyweb-virtual-machine:~/honeyhttpd/logs$ curl -k https://localhost:8443/
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login - Web Application</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background: #f7f7f7;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }
    .login-container {
      padding: 2rem;
      background: #ffffff;
      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
      border-radius: 8px;
      width: 300px;
      text-align: center;
    }
    .login-container img {
      width: 80%;
      margin-bottom: 1rem;
    }
    .login-form {
      display: flex;
      flex-direction: column;
    }
    .login-form label {
      text-align: left;
      margin-bottom: 0.5rem;
    }
    .login-form input[type='text'],
    .login-form input[type='password'] {
      padding: 10px;
      margin-bottom: 1rem;
      border: 1px solid #ddd;
      border-radius: 4px;
      box-sizing: border-box;
    }
    .login-form input[type='submit'] {
      padding: 10px;
      border: none;
      border-radius: 4px;
      background-color: #2d89ef;
      color: white;
      cursor: pointer;
      transition: background-color 0.2s;
    }
    .login-form input[type='submit']:hover {
      background-color: #2b5797;
    }
  </style>
</head>
<body>
  <div class="login-container">
    <!-- Placeholder for logo-->
    
    <form class="login-form" action="/post_login" method="post">
      <label for="username">Username</label>
      <input type="text" id="username" name="username">
      <label for="password">Password</label>
      <input type="password" id="password" name="password">
      <input type="submit" value="Login">
    </form>
  </div>
</body>
</html>
```

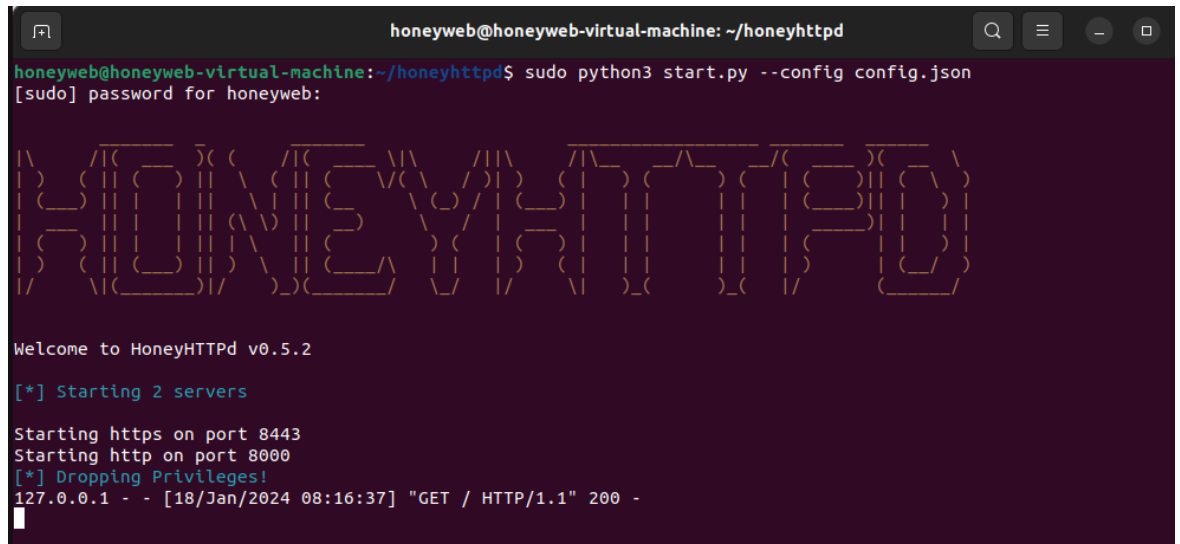
```

    color: white;
    cursor: pointer;
    transition: background-color 0.2s;
  }
  .login-form input[type='submit']:hover {
    background-color: #2b5797;
  }
</style>
</head>
<body>
  <div class="login-container">
    <!-- Placeholder for logo-->
    
    <form class="login-form" action="/post_login" method="post">
      <label for="username">Username</label>
      <input type="text" id="username" name="username">
      <label for="password">Password</label>
      <input type="password" id="password" name="password">
      <input type="submit" value="Login">
    </form>
  </div>
</body>
</html>
```



Figure 3: The curl command is used to request content from the HoneyHTTPD server running on port 8443 with HTTPS security protocol.

The result of the curl command displays an HTML/CSS code snippet, which is the source of the simulated DVWA login page served by HoneyHTTPD.



```
honeyweb@honeyweb-virtual-machine: ~/honeyhttpd
honeyweb@honeyweb-virtual-machine:~/honeyhttpd$ sudo python3 start.py --config config.json
[sudo] password for honeyweb:

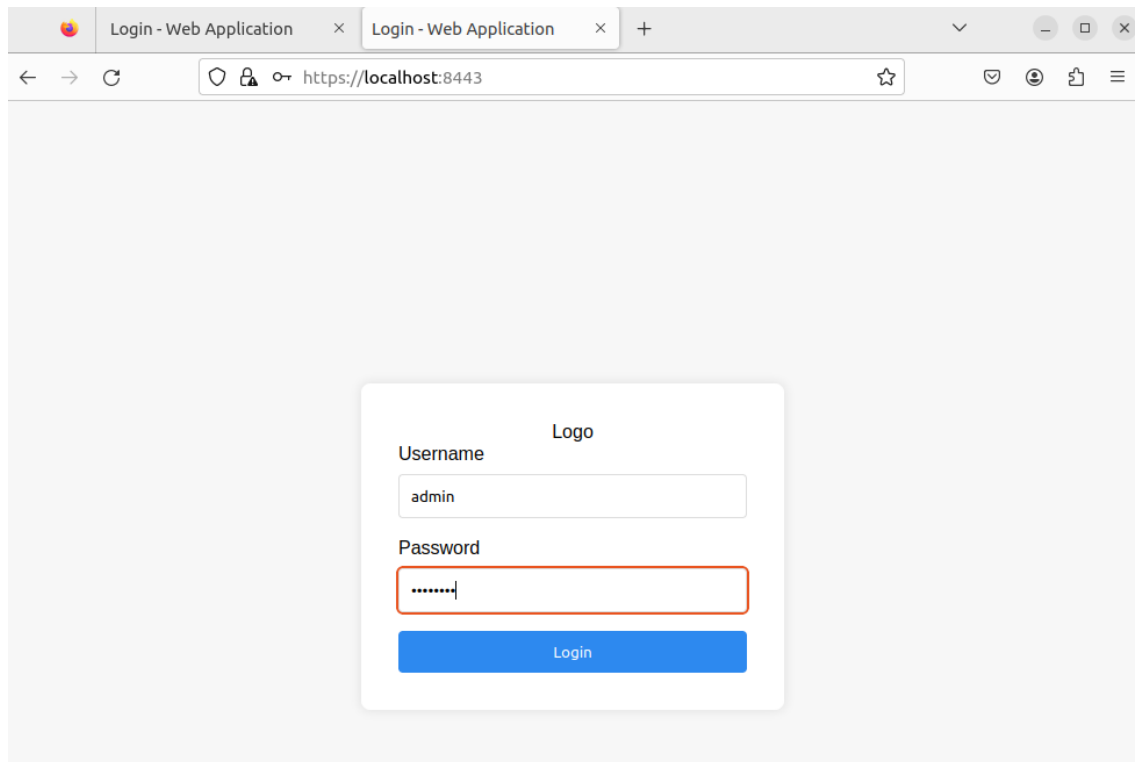
Welcome to HoneyHTTPD v0.5.2

[*] Starting 2 servers
Starting https on port 8443
Starting http on port 8000
[*] Dropping Privileges!
127.0.0.1 - - [18/Jan/2024 08:16:37] "GET / HTTP/1.1" 200 -
```

HoneyHTTPD has received a GET HTTP request and responded with 200 OK.

- The attacker proceeds to access the DVWA login page:

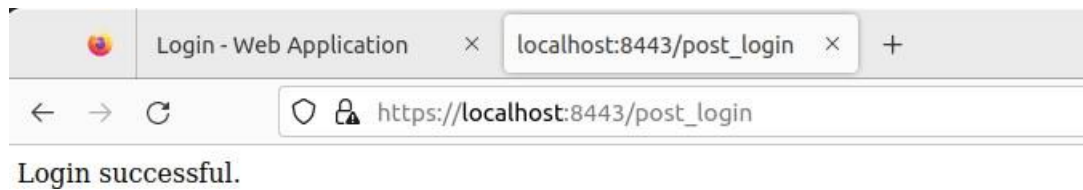




HoneyHTTPD has noticed a GET HTTP request has been sent and proceeded to respond with the simulated DVWA webpage:

[illegible]

When the attacker successfully logs in, HoneyHTTPD will display the text "Login Successful".



- However, HoneyHTTPD has logged this access in a log file for the administrator to observe login attempts:

```
honeyweb@honeyweb-virtual-machine: ~/honeyhttpd
honeyweb@honeyweb-virtual-machine:~/honeyhttpd$ sudo python3 start.py --config config.json
[sudo] password for honeyweb:

Welcome to HoneyHTTPd v0.5.2

[*] Starting 2 servers

Starting https on port 8443
Starting http on port 8000
[*] Dropping Privileges!
127.0.0.1 - - [18/Jan/2024 08:16:37] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2024 08:24:44] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2024 08:24:44] "GET /home/honeyweb/Pictures/dvwa.png HTTP/1.1" 404 -
127.0.0.1 - - [18/Jan/2024 08:27:46] "POST /post_login HTTP/1.1" 200 -
```

- When the administrator checks the log file to view the access flow to the website:

```
honeyweb@honeyweb-virtual-machine: ~/honeyhttp/logs
honeyweb@honeyweb-virtual-machine: ~/honeyhttp/logs$ ls
server-https-8443.log
honeyweb@honeyweb-virtual-machine: ~/honeyhttp/logs$ cat server-https-8443.log

2024-01-18T08:16:37.341459 - From 127.0.0.1:40266:
GET / HTTP/1.1
Host: localhost:8443
User-Agent: curl/7.81.0
Accept: */*

Sent:
b'<!DOCTYPE html>\n<html lang="en">\n<head>\n  <meta charset="UTF-8">\n  <meta name="viewport" content="width=device-width, initial-scale=1.0">\n  <title>Login - Web Application</title>\n  <style>\n    body {\n      font-family: \'Arial\', sans-serif;\n      background: #f7f7f7;\n      display: flex;\n      justify-content: center;\n      align-items: center;\n      height: 100vh;\n      margin: 0;\n    }\n    .login-container {\n      padding: 2rem;\n      background: #ffffff;\n      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);\n      border-radius: 8px;\n      width: 300px;\n      text-align: center;\n    }\n    .login-form {\n      display: flex;\n      flex-direction: column;\n    }\n    .login-form label {\n      text-align: left;\n      margin-bottom: 0.5rem;\n    }\n    .login-form input[type="text"] {\n      padding: 10px;\n      margin-bottom: 1rem;\n      border: none;\n      border-radius: 4px;\n      box-sizing: border-box;\n    }\n    .login-form input[type="password"] {\n      padding: 10px;\n      margin-bottom: 1rem;\n      border: none;\n      border-radius: 4px;\n      background-color: #2d89ef;\n      color: white;\n      cursor: pointer;\n      transition: background-color 0.2s;\n    }\n    .login-form input[type="submit"] {\n      padding: 10px;\n      border: none;\n      border-radius: 4px;\n      background-color: #2d89ef;\n      color: white;\n      cursor: pointer;\n      transition: background-color 0.2s;\n    }\n  </style>\n</head>\n<body>\n  <div class="login-container">\n    <!-- Placeholder for logo-->\n    \n    <form class="login-form" action="/post_login" method="post">\n      <label for="username">Username</label>\n      <input type="text" id="username" name="username" />\n      <label for="password">Password</label>\n      <input type="password" id="password" name="password" />\n      <input type="submit" value="Login" />\n    </form>\n  </div>\n</body>\n</html>'
```

```
Sent:
b'<!DOCTYPE html>\n<html lang="en">\n<head>\n  <meta charset="UTF-8">\n  <meta name="viewport" content="width=device-width, initial-scale=1.0">\n  <title>Login - Web Application</title>\n  <style>\n    body {\n      font-family: \'Arial\', sans-serif;\n      background: #f7f7f7;\n      display: flex;\n      justify-content: center;\n      align-items: center;\n      height: 100vh;\n      margin: 0;\n    }\n    .login-container {\n      padding: 2rem;\n      background: #ffffff;\n      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);\n      border-radius: 8px;\n      width: 300px;\n      text-align: center;\n    }\n    .login-form {\n      display: flex;\n      flex-direction: column;\n    }\n    .login-form label {\n      text-align: left;\n      margin-bottom: 0.5rem;\n    }\n    .login-form input[type="text"] {\n      padding: 10px;\n      margin-bottom: 1rem;\n      border: none;\n      border-radius: 4px;\n      box-sizing: border-box;\n    }\n    .login-form input[type="password"] {\n      padding: 10px;\n      margin-bottom: 1rem;\n      border: none;\n      border-radius: 4px;\n      background-color: #2d89ef;\n      color: white;\n      cursor: pointer;\n      transition: background-color 0.2s;\n    }\n    .login-form input[type="submit"] {\n      padding: 10px;\n      border: none;\n      border-radius: 4px;\n      background-color: #2d89ef;\n      color: white;\n      cursor: pointer;\n      transition: background-color 0.2s;\n    }\n  </style>\n</head>\n<body>\n  <div class="login-container">\n    <!-- Placeholder for logo-->\n    \n    <form class="login-form" action="/post_login" method="post">\n      <label for="username">Username</label>\n      <input type="text" id="username" name="username" />\n      <label for="password">Password</label>\n      <input type="password" id="password" name="password" />\n      <input type="submit" value="Login" />\n    </form>\n  </div>\n</body>\n</html>'
```

```
2024-01-18T08:24:44.751215 - From 127.0.0.1:57276:
GET /home/honeyweb/Pictures/dwa.png HTTP/1.1
Host: localhost:8443
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: https://localhost:8443/
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

```
Sent:
b'<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">\n<html>\n<head>\n<title>404 Not Found</title>\n</head>\n<body>\n<h1>Not Found</h1>\n<p>The requested URL /home/honeyweb/Pictures/dwa.png was not found on this server.<br />\n</p>\n</body>\n</html>'
```

```
2024-01-18T08:27:46.890988 - From 127.0.0.1:44436:
POST /post_login HTTP/1.1
Host: localhost:8443
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Origin: https://localhost:8443
Connection: keep-alive
Referer: https://localhost:8443/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

Sent:
b'Login successful.'
```

It can be seen that HoneyHTTPD has logged access attempts to the simulated DVWA website in the "server-https-8443.log" file. HoneyHTTPD records these website visits and responds in a pre-configured manner. This makes the attacker believe they have successfully accessed the website without being detected.

HoneyHTTPD has successfully simulated a DVWA login webpage and deceived attackers into thinking they have successfully attacked and logged into the official DVWA website. However, HoneyHTTPD still only responds with simple information to illustrate interaction with the attacker, unable to respond with the complete content of the webpage when the attacker successfully logs in.

## **Chapter 6. Conclusion and Future Directions**

### **Conclusion:**

This project aims to integrate Reinforcement Learning (RL) with HoneyHTTPD to create an advanced framework for web deception defenses. The main issue is the lack of RL integration within HoneyHTTPD, which limits its adaptability and responsiveness to various web attacks.

The approach involves deep research into the foundational theory of RL and web deception defenses, followed by systematic system design and implementation. However, the current state of HoneyHTTPD has not yet been integrated with RL. Nevertheless, the groundwork has been laid for future integration, and the project has provided profound insights into the challenges and potential solutions for this task.

### **Future Directions:**

The main direction for future development will be integrating RL into HoneyHTTPD. This will involve developing RL algorithms that can effectively operate with HoneyHTTPD, testing them in various scenarios, and fine-tuning the system based on the results.

Additionally, future work may explore using different RL algorithms, developing more sophisticated deception techniques, and integrating with other defense systems. The ultimate goal is to create a robust and adaptive framework for web deception defenses, capable of effectively protecting web servers against various types of attacks.

## REFERENCES

A. El-Kosairy and M. A. Azer, “A New Web Deception System Framework,” in 2018 1st International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 2018

<https://github.com/bocajsppear1/honeyhttpd.git>