Gary Campbell
INFOST 481 (U) Web Application Penetration Testing

MARTIN BISHOP & ASSOCIATES

Penetration Testing and Security Solutions

A Subsidiary of **SPY PHANTOM IO**

Web Application Penetration Test Report for HAGS

Client: Hippos Are Great Systems (HAGS)
Tester: Gary Campbell

Date: October 22nd, 2024
Project: Midterm Web Application Penetration Test

MARTIN BISHOP & ASSOCIATES

Penetration Testing and Security Solutions

A Subsidiary of **SPY PHANTOM IO**

Table of Contents

MARTIN BISHOP & ASSOCIATES

Penetration Testing and Security Solutions

A Subsidiary of **SPY PHANTOM IO**

1. Executive Summary

This penetration test successfully identified and exploited 9 hidden flags in the Hippos Are Great Systems (HAGS) web application. Additionally, the extra credit flag was discovered using further social engineering techniques and decoding methods. The findings reveal several vulnerabilities, such as weak password policies, insufficient access control, and client-side validation issues, which are detailed in the following sections.

2. Methodology

Tools such as Burp Suite and browser Developer Tools were extensively used to intercept traffic, modify requests, and analyze the web application's behavior. Burp Suite was configured for brute-force attacks using the Intruder module, which allowed testing of numerous login credentials and cookie manipulation. Browser Developer Tools were employed to inspect HTML and JavaScript to reveal hidden form fields, security questions, and encoded data. Additionally, ChatGPT was utilized to assist in decoding encoded strings such as Base64 and hex-encoded values. The methodology is outlined below:

1. Reconnaissance

- **Objective**: To gather information about the target system without directly interacting with it in a way that might alert the system to an attacker's presence.
- **Tools Used**: Browser Developer Tools, manual browsing.
- **Techniques**:
    - **Manual Browsing**: Explored the website's visible and hidden pages to identify potential points of interest such as login forms and admin directories.

- **HTML and JavaScript Inspection**: Used browser Developer Tools to inspect page elements and scripts, revealing hidden form fields, admin areas, and JavaScript functions that could be exploited.

2. Vulnerability Identification

- **Objective**: To identify potential vulnerabilities in the application that could be exploited.
- **Tools Used**: Burp Suite, Browser Developer Tools, ChatGPT (for decoding encoded data).
- **Techniques**:
  - **Brute Force and Directory Fuzzing**: Used Burp Suite's Intruder tool to automate the discovery of directories and files that were not accessible through normal browsing.
  - **Cookie Inspection and Manipulation**: Inspected and modified cookies using Burp Suite and browser Developer Tools, identifying privileges stored in session cookies.
  - **Decoding**: Used ChatGPT as a tool for decoding Base64 and hexadecimal-encoded strings found in cookies, parameters, and JavaScript. This enabled the discovery of hidden information such as session data and security answers.

3. Exploitation

- **Objective**: To actively exploit identified vulnerabilities in order to capture the hidden flags.
- **Tools Used**: Burp Suite, Browser Developer Tools, ChatGPT (as a decoding tool).
- **Techniques**:
  - **Dictionary and Brute Force Attacks**: Utilized Burp Suite's Intruder module to perform dictionary attacks on login forms and brute-force directories.

- **Cookie Manipulation**: Modified cookies (such as the _USER cookie) to escalate privileges and gain access to admin-only areas.
- **JavaScript Manipulation**: Bypassed front-end security checks by inspecting and altering JavaScript validation functions.
- **Form Manipulation**: Altered form inputs (e.g., length restrictions) to bypass client-side validation and submit longer strings or hidden parameters.

4. Documentation and Flag Capture

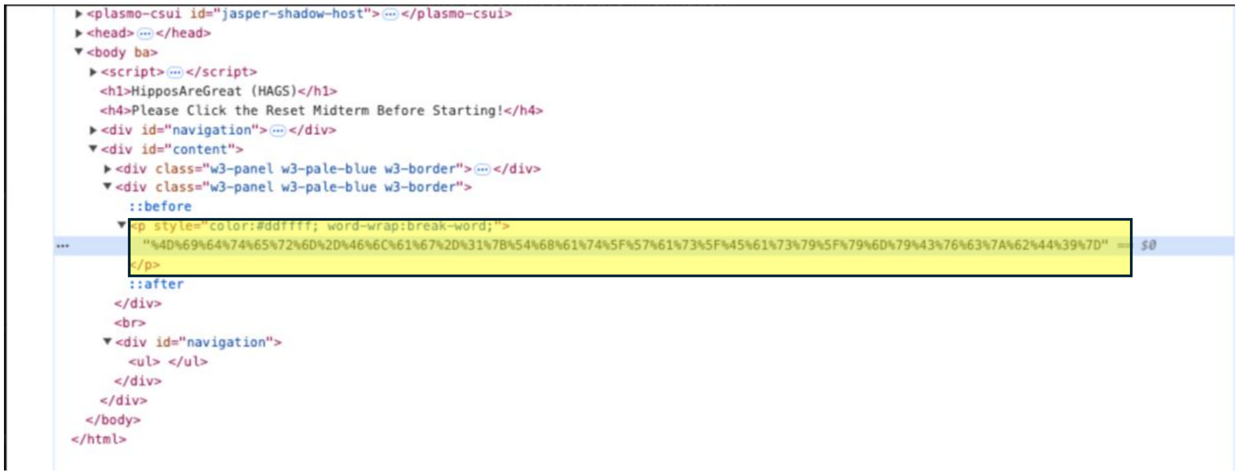
- **Objective**: To document each flag's discovery process, ensuring it could be replicated.
- **Tools Used**: Burp Suite, Browser Developer Tools, ChatGPT (for report structuring).
- **Techniques**:
    - For each flag, a step-by-step reproduction was created, including all steps taken, tools used, and detailed explanations of the vulnerabilities exploited.
    - Screenshots were taken for each significant action, and ChatGPT assisted in structuring the report to ensure clarity and ease of understanding.

5 .Types of Attacks Performed

- **Brute-Force Attack**: Automated testing of various username and password combinations to gain access to restricted areas.
- **Directory Traversal**: Discovered hidden directories by fuzzing URLs and testing common directory names (e.g., /admin_area/).
- **Parameter Tampering**: Modified hidden form fields, URL parameters, and cookies to bypass validation checks and escalate privileges.
- **Decoding**: Decoded encoded data (Base64 and hexadecimal) using ChatGPT to reveal hidden strings, passwords, and session details stored in cookies or JavaScript.

3. Findings

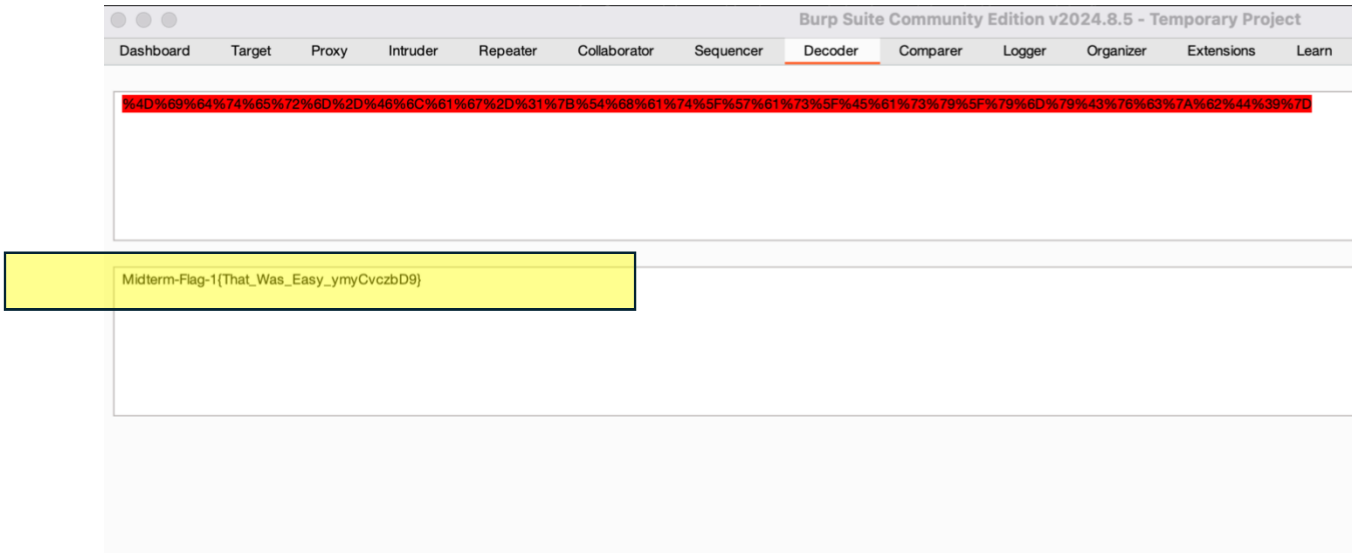
Flag 1 Discovery



```
▶<plasmo-csui id="jasper-shadow-host">⋯</plasmo-csui>
▶<head>⋯</head>
▼<body ba>
  ▶<script>⋯</script>
   <h1>HipposAreGreat (HAGS)</h1>
   <h4>Please Click the Reset Midterm Before Starting!</h4>
  ▶<div id="navigation">⋯</div>
  ▼<div id="content">
    ▶<div class="w3-panel w3-pale-blue w3-border">⋯</div>
    ▼<div class="w3-panel w3-pale-blue w3-border">
       ::before
      ▼<p style="color:#ddffff; word-wrap:break-word;">
...      "%4D%69%64%74%65%72%6D%2D%46%6C%61%67%2D%31%7B%54%68%61%74%5F%57%61%73%5F%45%61%73%79%5F%79%6D%79%43%76%63%7A%62%44%39%7D" == $0
       </p>
       ::after
     </div>
     <br>
    ▼<div id="navigation">
       <ul> </ul>
     </div>
   </div>
 </body>
</html>
```

Screenshot 1- Source Code Inspection finding URL encoded string in the code

- ***Description:*** *By inspecting the page source code, a URL-encoded string was found in a p element styled to be hidden. This string appeared suspicious and was likely encoded to conceal a flag.*
- ***Action Taken:*** *Copied the URL-encoded string for decoding. This type of encoding often contains valuable information, especially in the context of penetration testing.*



```
%4D%69%64%74%65%72%6D%2D%46%6C%61%67%2D%31%7B%54%68%61%74%5F%57%61%73%5F%45%61%73%79%5F%79%6D%79%43%76%63%7A%62%44%39%7D
```

```
Midterm-Flag-1{That_Was_Easy_ymyCvczbD9}
```

Screenshot 2 - Decoding with Burp Suite the URL encoded String

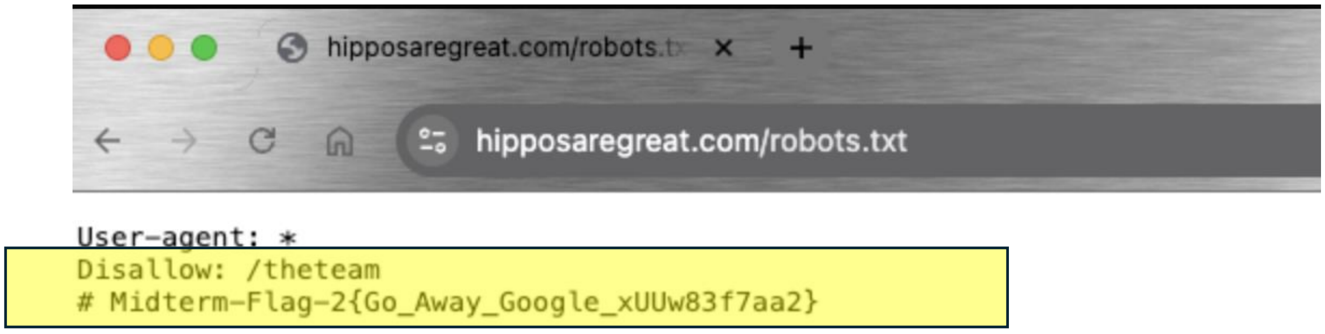
- ***Description:*** *The URL-encoded string %4D%69%64%74%65%72%6D%2D%46%6C%61%67%2D%31%7B%54%68%61%74%5F%57%61%73%5F%45%61%73%79%5F%79%6D%79%43%76%63%7A%62%44%39%7Dwas input into Burp Suite's Decoder tool, where it was successfully converted to reveal the hidden flag.*
- ***Flag Found:*** *Midterm-Flag-1{That_Was_Easy_ymyCvczbD9}*
- ***Steps to Reproduce:***
    1. *Opened the browser's developer tools and inspected the page source.*
    2. *Located a hidden, URL-encoded string within the HTML.*
    3. *Copied the encoded string and pasted it into Burp Suite's Decoder tool.*
    4. *Decoded the string, revealing the flag text.*
- ***Analysis:*** *Finding encoded strings within page sources is common in penetration tests. Such strings are often hidden but can be easily decoded with tools like Burp Suite, leading to the discovery of sensitive information or flags.*

MARTIN BISHOP & ASSOCIATES

Penetration Testing and Security Solutions

A Subsidiary of **SPY PHANTOM IO**

Flag 2 Discovery



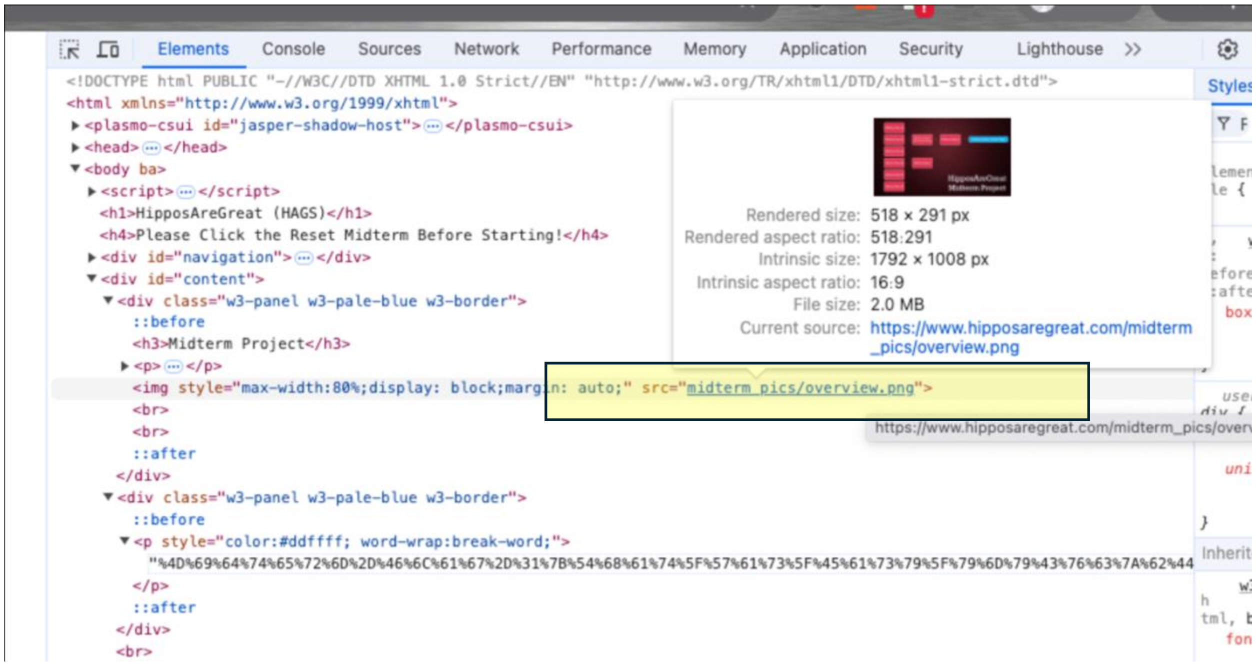
```
User-agent: *
Disallow: /theteam
# Midterm-Flag-2{Go_Away_Google_xUUw83f7aa2}
```

Screenshot 3 - Robots.txt File access to Midterm Flag 2 and a hidden directory

- ***Description:*** *The robots.txt file at https://hipposaregreat.com/robots.txt was accessed to explore directories that might be hidden from search engines but could contain important information. Upon inspection, the file disallows user agents from accessing the /theteam directory, and it also contains a comment revealing Flag 2.*
- ***Flag Found:*** *Midterm-Flag-2{Go_Away_Google_xUWw83f7aa2}*
- ***Steps to Reproduce:***
    1. *Navigated to https://hipposaregreat.com/robots.txt in the browser.*
    2. *Examined the content of the file, which specifies directives for web crawlers. The disallow directive hints at areas possibly containing hidden content.*
    3. *Identified the flag hidden as a comment within the file.*

- ***Analysis:*** *Accessing robots.txt files can often reveal sensitive paths or content meant to be hidden from public view. This is a common tactic in penetration testing to uncover accessible but overlooked information.*

Flag 3 Discovery



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 ▶<plasmo-csui id="jasper-shadow-host">⋯</plasmo-csui>
 ▶<head>⋯</head>
 ▼<body ba>
   ▶<script>⋯</script>
    <h1>HipposAreGreat (HAGS)</h1>
    <h4>Please Click the Reset Midterm Before Starting!</h4>
   ▶<div id="navigation">⋯</div>
   ▼<div id="content">
     ▼<div class="w3-panel w3-pale-blue w3-border">
        ::before
        <h3>Midterm Project</h3>
       ▶<p>⋯</p>
        <img style="max-width:80%;display: block;margin: auto;" src="midterm_pics/overview.png">
        <br>
        <br>
        ::after
      </div>
     ▼<div class="w3-panel w3-pale-blue w3-border">
        ::before
       ▼<p style="color:#ddffff; word-wrap:break-word;">
          "%4D%69%64%74%65%72%6D%2D%46%6C%61%67%2D%31%7B%54%68%61%74%5F%57%61%73%5F%45%61%73%79%5F%79%6D%79%43%76%63%7A%62%44
        </p>
        ::after
      </div>
      <br>
```

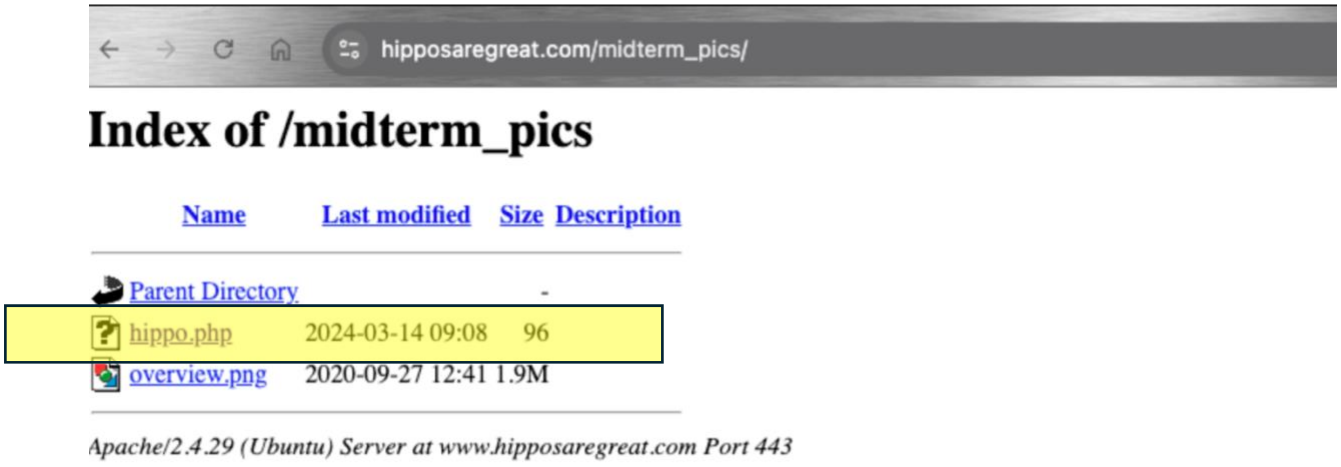
```
Rendered size: 518 × 291 px
Rendered aspect ratio: 518∶291
Intrinsic size: 1792 × 1008 px
Intrinsic aspect ratio: 16∶9
File size: 2.0 MB
Current source: https://www.hipposaregreat.com/midterm_pics/overview.png
```

https://www.hipposaregreat.com/midterm_pics/overv

Screenshot 4 - /midterm_pics Directory Listing for possible flag discovery

- ***Description:*** *Navigated to the /midterm_pics/ directory, where a directory listing was enabled. Two files were visible: hippo.php and overview.png. The presence of a .php file in a folder meant for images raised suspicion, as it could contain executable code or hidden information.*

- ***Action Taken:*** *Clicked on hippo.php to investigate further, as PHP files often contain server-side scripts that may reveal sensitive information or lead to hidden flags.*



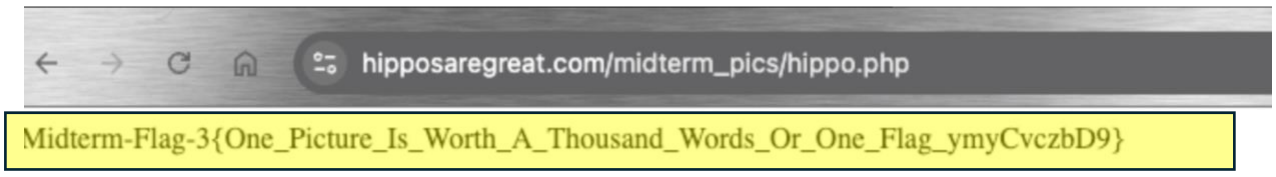
Index of /midterm_pics

Name	Last modified	Size	Description
Parent Directory		-	
hippo.php	2024-03-14 09:08	96	
overview.png	2020-09-27 12:41	1.9M	

Apache/2.4.29 (Ubuntu) Server at www.hipposaregreat.com Port 443

Screenshot 5 - Inspecting the Image Source Path, hippo.php indicating a undiscovered page

- ***Description:*** *In the HTML source of the main page, the image linked to https://hipposaregreat.com/midterm_pics/overview.png. This hinted at the /midterm_pics/ directory, leading to the discovery of the directory listing.*
- ***Steps to Reproduce:***
    1. *Inspected the HTML source code of the main page to locate the path of the image being displayed.*
    2. *Noted that the image resided in the /midterm_pics/ directory.*
    3. *Navigated directly to https://hipposaregreat.com/midterm_pics/ in the browser, revealing the directory listing.*



hipposaregreat.com/midterm_pics/hippo.php

Midterm-Flag-3{One_Picture_Is_Worth_A_Thousand_Words_Or_One_Flag_ymyCvczbD9}

Screenshot 6 - Flag 3 Revealed in hippo.php

- ***Description:*** *Accessed https://hipposaregreat.com/midterm_pics/hippo.php, which directly displayed Flag 3 in the browser. The flag was output as plain text, confirming its presence within this PHP file.*
- ***Flag Found:*** *Midterm-Flag-3{One_Picture_Is_Worth_A_Thousand_Words_Or_One_Flag_ymyCvczbD9}*
- ***Steps to Reproduce:***
  1. *After identifying the /midterm_pics/ directory from the main page's image path, navigated to https://hipposaregreat.com/midterm_pics/.*
  2. *Observed the file listing, including hippo.php.*
  3. *Opened hippo.php, which displayed the flag directly on the page.*
- ***Analysis:*** *PHP files within public directories often reveal information when accessed directly, as they may execute server-side code that outputs sensitive data. In this case, a direct visit to hippo.php exposed the flag as plaintext output.*

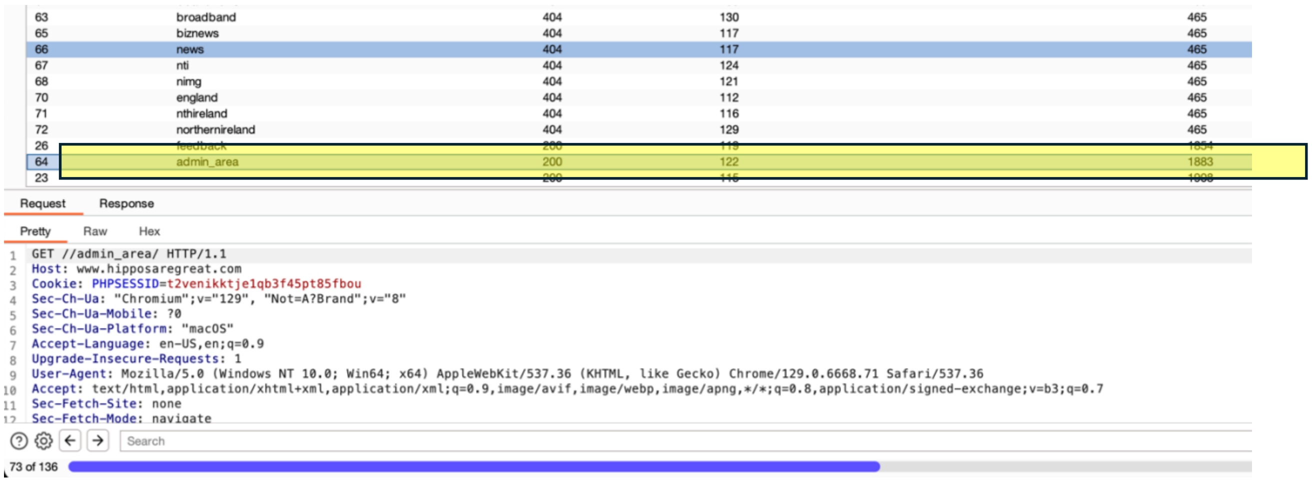
Flag 4 Discovery

Description:

*This flag was discovered by performing a brute-force directory search on the web application. By using a directory scanner tool, various paths were tested until /admin_area/ returned a successful response. This hidden page displayed **Flag 4** directly on the page, granting access to the flag content.*

Steps to Reproduce:

1. *Used a directory brute-forcing tool in Burp Suite to explore potential hidden directories within the web application.*
2. *Observed that the directory /admin_area/ responded with a 200 status code, indicating a valid path.*
3. *Navigated to https://hipposaregreat.com/admin_area/ in a web browser to inspect the page's contents.*
4. *Found **Flag 4** displayed on the page, which read: Midterm-Flag-4{I_Found_You_efjduM4fa2}*



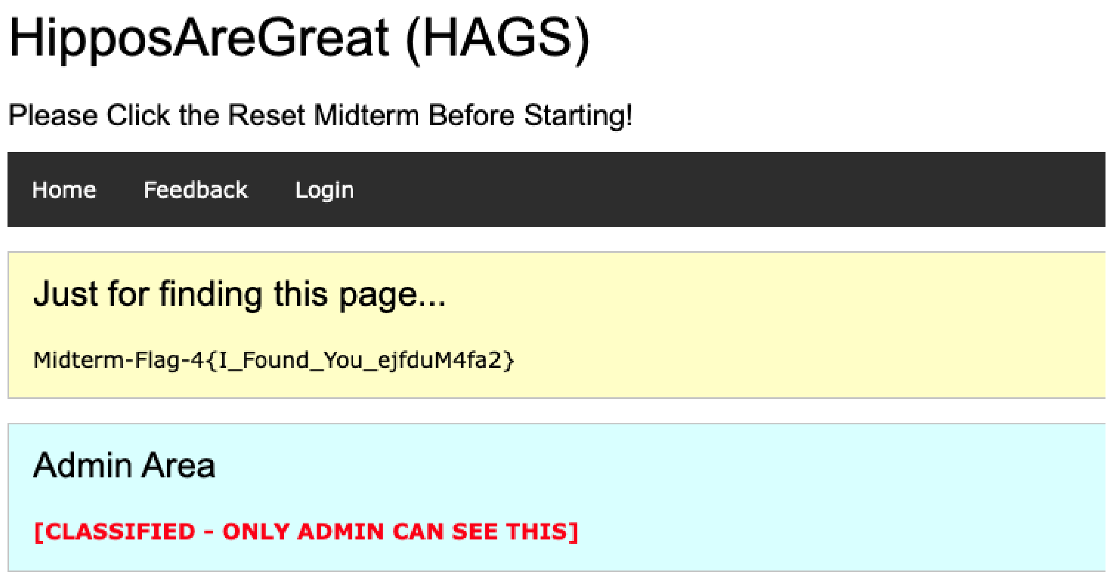
63	broadband	404	130	465
65	biznews	404	117	465
66	news	404	117	465
67	nti	404	124	465
68	nimg	404	121	465
70	england	404	112	465
71	nthireland	404	116	465
72	northernireland	404	129	465
26	feedback	200	119	1854
64	admin_area	200	122	1883
23		200	115	1908

Request Response

Pretty Raw Hex

```
GET //admin_area/ HTTP/1.1
Host: www.hipposaregreat.com
Cookie: PHPSESSID=t2venikktje1qb3f45pt85fbou
Sec-Ch-Ua: "Chromium";v="129", "Not=A?Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.6668.71 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
```

Screenshot 7 - Directory brute-force tool output showing /admin_area/ as a valid path with a 200 status code.


HipposAreGreat (HAGS)

Please Click the Reset Midterm Before Starting!

Just for finding this page...

Midterm-Flag-4{I_Found_You_ejfduM4fa2}

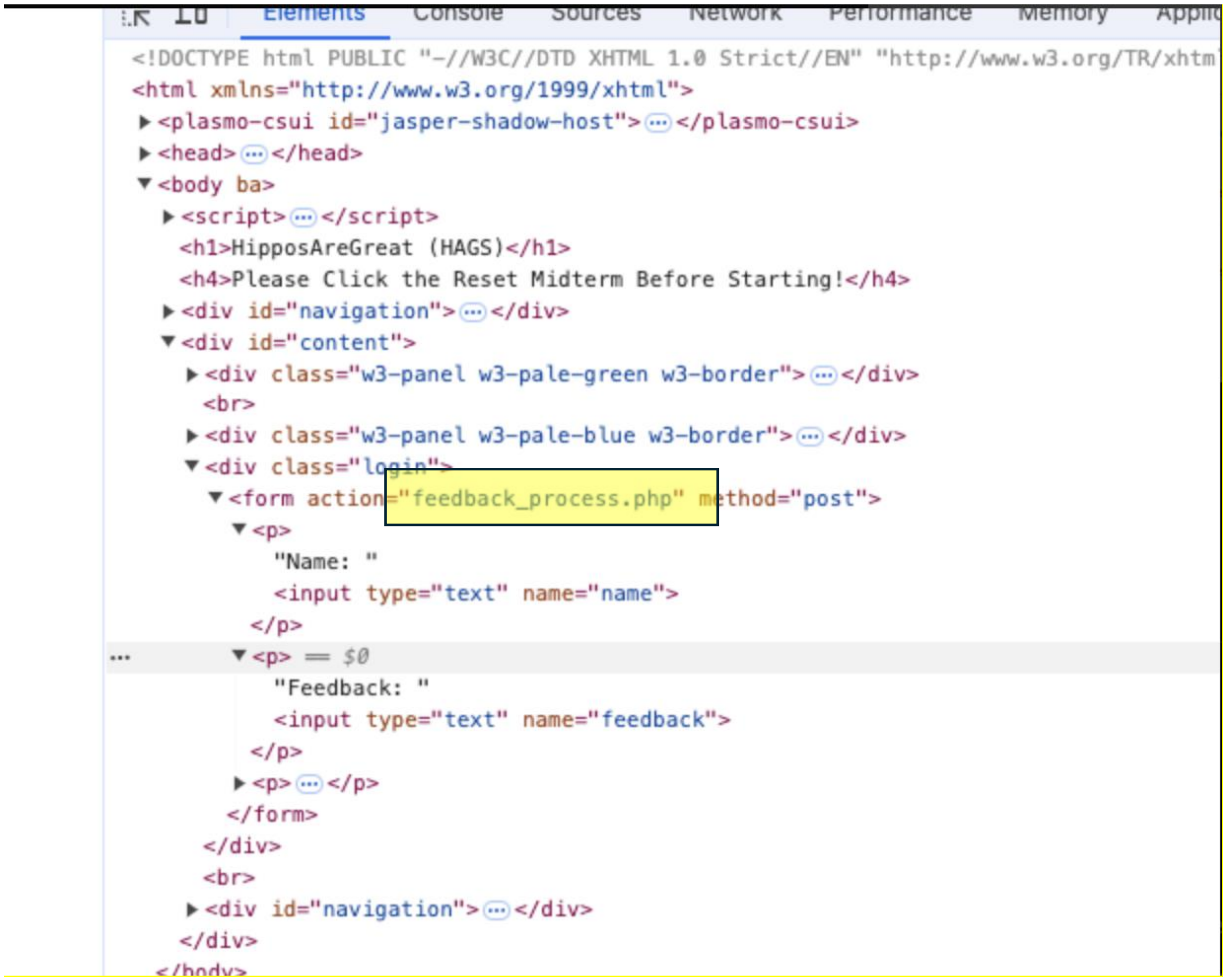
Admin Area

[CLASSIFIED - ONLY ADMIN CAN SEE THIS]

Screenshot 8 - The /admin_area/ page displaying Flag 4 on the website.

Analysis:
This exercise demonstrates the importance of securing hidden directories on web applications. An attacker could exploit these hidden paths to access sensitive information. Regular audits and restricted access policies are recommended to safeguard these areas from unauthorized access.

Flag 5 Discovery

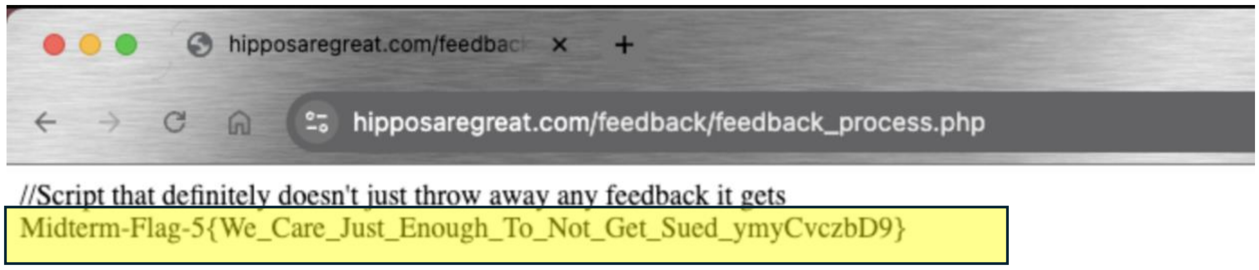


```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtm
<html xmlns="http://www.w3.org/1999/xhtml">
 ▶<plasmo-csui id="jasper-shadow-host">⋯</plasmo-csui>
 ▶<head>⋯</head>
 ▼<body ba>
   ▶<script>⋯</script>
    <h1>HipposAreGreat (HAGS)</h1>
    <h4>Please Click the Reset Midterm Before Starting!</h4>
   ▶<div id="navigation">⋯</div>
   ▼<div id="content">
     ▶<div class="w3-panel w3-pale-green w3-border">⋯</div>
      <br>
     ▶<div class="w3-panel w3-pale-blue w3-border">⋯</div>
     ▼<div class="login">
       ▼<form action="feedback_process.php" method="post">
         ▼<p>
            "Name: "
            <input type="text" name="name">
          </p>
         ▼<p> == $0
            "Feedback: "
            <input type="text" name="feedback">
          </p>
         ▶<p>⋯</p>
        </form>
      </div>
      <br>
     ▶<div id="navigation">⋯</div>
    </div>
  </body>
```

Screenshot 9 - Feedback Form HTML Inspection indicating a feedback_process.php that can be visited

- ***Description:*** *By inspecting the HTML of the feedback form on the site, the action attribute was found pointing to feedback_process.php. This indicates the script that processes form submissions and could potentially reveal additional information if accessed directly.*
- ***Steps to Reproduce:***
    1. *Opened the browser's developer tools and navigated to the "Elements" tab.*

2. *Located the feedback form and examined the form action attribute, which directed submissions to feedback_process.php.*


hipposaregreat.com/feedback
hipposaregreat.com/feedback/feedback_process.php

//Script that definitely doesn't just throw away any feedback it gets
Midterm-Flag-5{We_Care_Just_Enough_To_Not_Get_Sued_ymyCvczbD9}

Screenshot 10 - Accessing feedback_process.php indicating Midterm Flag 5

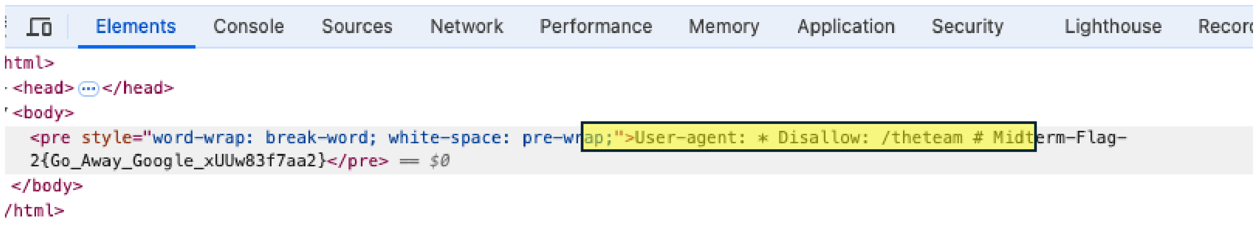
- ***Description:*** *Navigated to https://hipposaregreat.com/feedback/feedback_process.php based on the form action. The page displayed a comment about handling feedback, followed by the hidden flag.*
- ***Flag Found:*** *Midterm-Flag-5{We_Care_Just_Enough_To_Not_Get_Sued_ymyCvczbD9}*
- ***Analysis:*** *This demonstrates how direct access to form processing scripts can expose sensitive data. In this case, feedback_process.php was unprotected and openly displayed the flag, which could be a common misconfiguration in web applications that lack access controls on backend scripts.*

MARTIN BISHOP & ASSOCIATES

Penetration Testing and Security Solutions

A Subsidiary of **SPY PHANTOM IO**

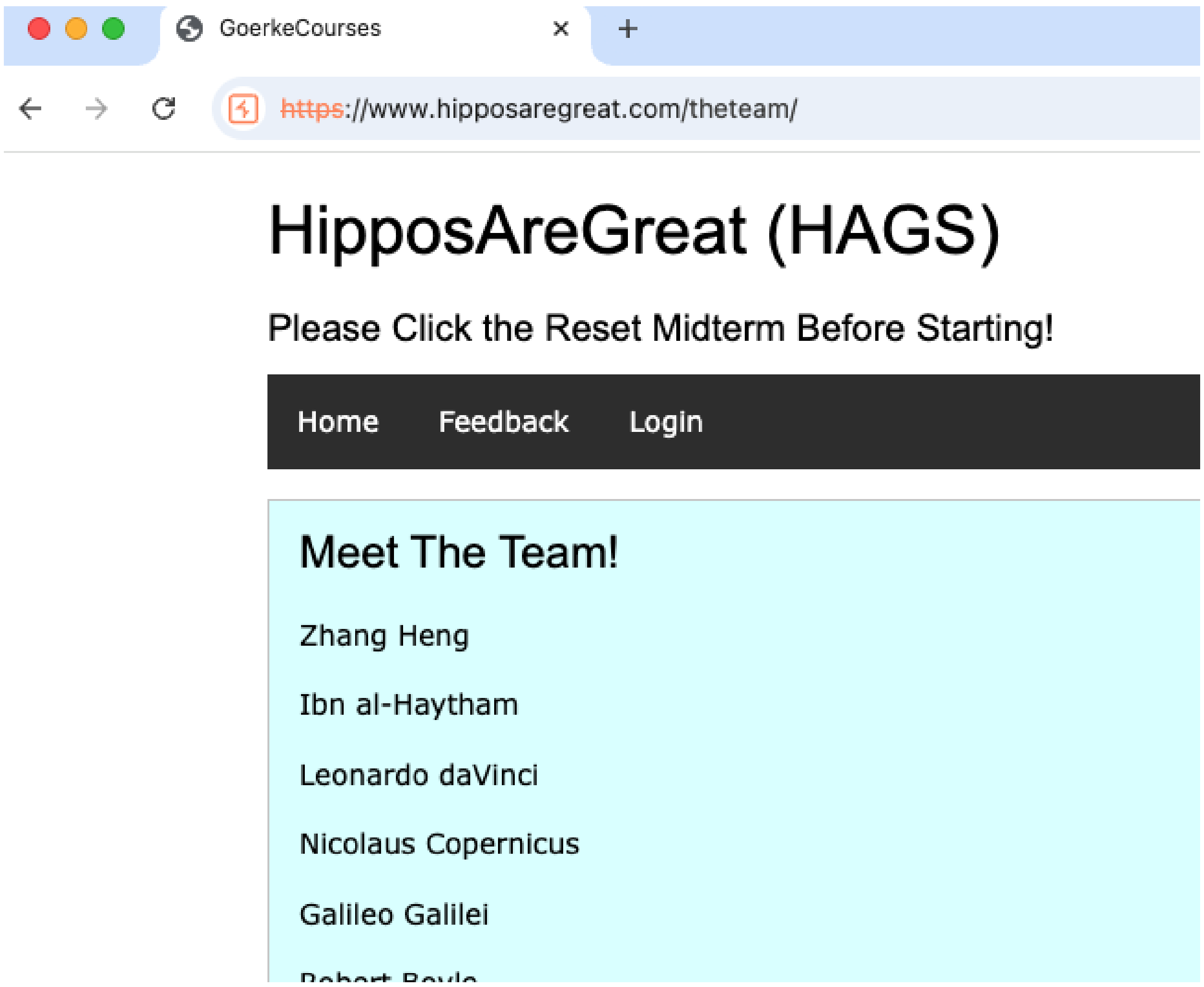
Flag 6 - User Discovery (Non-Flag)



```
html>
<head>⋯</head>
<body>
  <pre style="word-wrap: break-word; white-space: pre-wrap;">User-agent: * Disallow: /theteam # Midterm-Flag-
  2{Go_Away_Google_xUUw83f7aa2}</pre> == $0
</body>
/html>
```

Screenshot 11 - Inspection of robots.txt for Clues – Showing a hidden director /theteam and the Midterm Flag 2

- ***Description:*** *By inspecting robots.txt on the website, a clue was found about a directory /theteam that was restricted from web crawlers. This directory potentially contains information about the team members or user data that could be useful for further testing.*
- ***Steps to Reproduce:***
  1. *Accessed https://hipposaregreat.com/robots.txt.*
  2. *Noted the restriction on /theteam, which warranted further exploration.*



HipposAreGreat (HAGS)

Please Click the Reset Midterm Before Starting!

Meet The Team!

Zhang Heng

Ibn al-Haytham

Leonardo daVinci

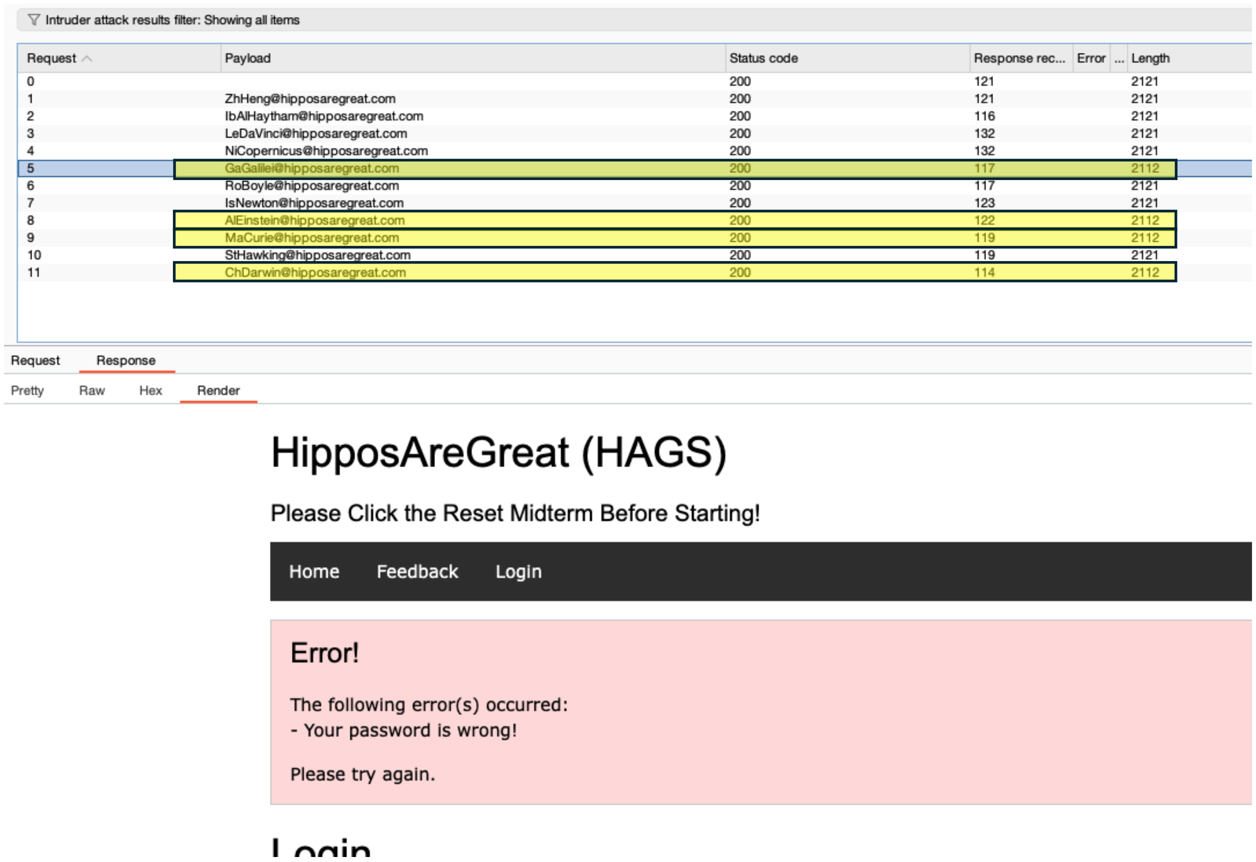
Nicolaus Copernicus

Galileo Galilei

Screenshot 12 - Exploring the /theteam Directory List of protentional targets for entry into the system

- ***Description:*** *Navigated to https://hipposaregreat.com/theteam, where a list of team members was displayed. This list included names of historical figures, which suggested these names could be potential usernames or email addresses.*
- ***Action Taken:*** *Compiled a list of team member names for testing against the login page.*
- *After gathering a list of validated emails, Burp Suite's Intruder tool was used to test combinations of emails and common passwords. The 'Cluster Bomb' attack method allowed the testing of each combination, and responses from the server (such as*

differing status codes and response lengths) revealed whether the login credentials were valid or not. The tool automatically tracked which combinations returned an abnormal response, indicating a correct username and password


Intruder attack results filter: Showing all items

Request	Payload	Status code	Response rec...	Error	...	Length
0		200	121			2121
1	ZhHeng@hipposaregreat.com	200	121			2121
2	IbAlHaytham@hipposaregreat.com	200	116			2121
3	LeDaVinci@hipposaregreat.com	200	132			2121
4	NiCopernicus@hipposaregreat.com	200	132			2121
5	GaGalilei@hipposaregreat.com	200	117			2112
6	RoBoyle@hipposaregreat.com	200	117			2121
7	IsNewton@hipposaregreat.com	200	123			2121
8	AlEinstein@hipposaregreat.com	200	122			2112
9	MaCurie@hipposaregreat.com	200	119			2112
10	StHawking@hipposaregreat.com	200	119			2121
11	ChDarwin@hipposaregreat.com	200	114			2112

HipposAreGreat (HAGS)

Please Click the Reset Midterm Before Starting!

Error!

The following error(s) occurred:
- Your password is wrong!

Please try again.

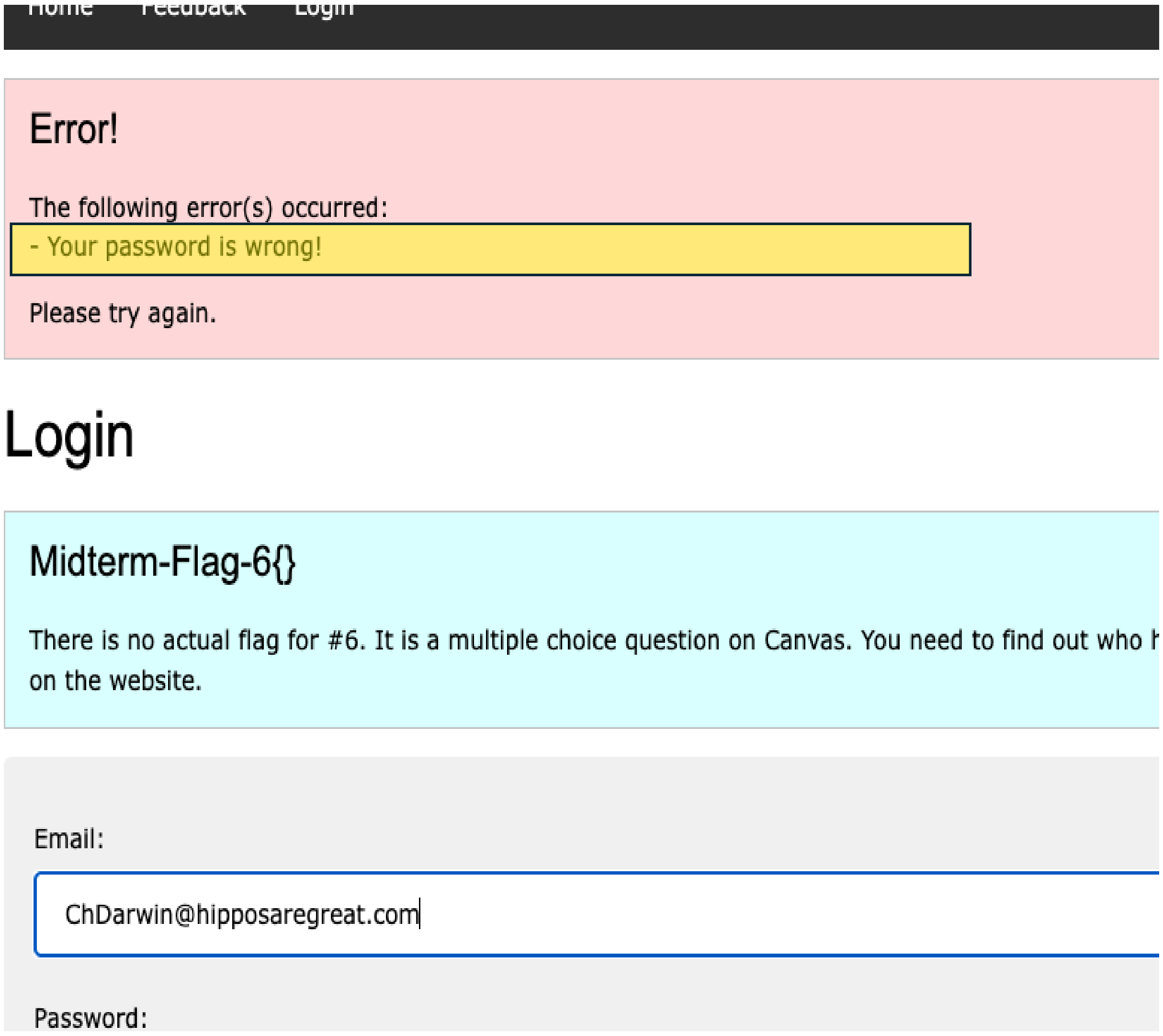
Login

Screenshot 13 - Compiling and Testing Email Addresses with Burp Suite Intruder showing all valid users with a length of 2112

- ***Description:*** *Based on the team names, created a set of email addresses using the format [Name]@hipposaregreat.com. These email addresses were then used in Burp Suite's Intruder tool to test for valid user accounts by analyzing response differences.*
- ***Steps to Reproduce:***
    1. *Created a list of email addresses based on the team member names.*
    2. *Configured Burp Suite's Intruder to test each email against the login page.*

3. *Analyzed response codes and response sizes to identify potential valid users.*

- ***Result:*** *The response for ChDarwin@hipposaregreat.com indicated a "wrong password" error, suggesting it is a valid username. This aligns with the hint provided that Flag 6 is tied to a multiple-choice question rather than a flag in the conventional sense.*



Error!

The following error(s) occurred:
- Your password is wrong!

Please try again.

Login

Midterm-Flag-6{}

There is no actual flag for #6. It is a multiple choice question on Canvas. You need to find out who h
on the website.

Email:

ChDarwin@hipposaregreat.com

Password:

Screenshot 14 - Login Page with Valid User showing that the user is in the system (correct) by indicating password is wrong

- ***Description:*** *Attempted to log in with ChDarwin@hipposaregreat.com, GaGalilei@hipposaregreat.com, MaCurie@hipposaregreat.com,* *AlEinstein@hipposaregreat.com, resulting in a "wrong password" error message. This response is consistent with an existing account but incorrect password, indicating a valid username for Flag 6's associated multiple-choice question.*

- ***Analysis:** Using Burp Suite's Intruder function to brute force potential usernames based on visible team data revealed the likely user account ChDarwin@hipposaregreat.com, GaGalilei@hipposaregreat.com, MaCurie@hipposaregreat.com, AlEinstein@hipposaregreat.com, . While Flag 6 itself is not directly a flag, this process helps with understanding how usernames can be identified through HTTP response analysis.*

Flag 7 Discovery

***Description**: Flag 7 was discovered on the "Are You Worthy?" page by interacting with the JavaScript validation function. The flag could be obtained through two different approaches: decoding a hex string found in the function or modifying the input field's maxlength attribute to submit the correct secret code.*

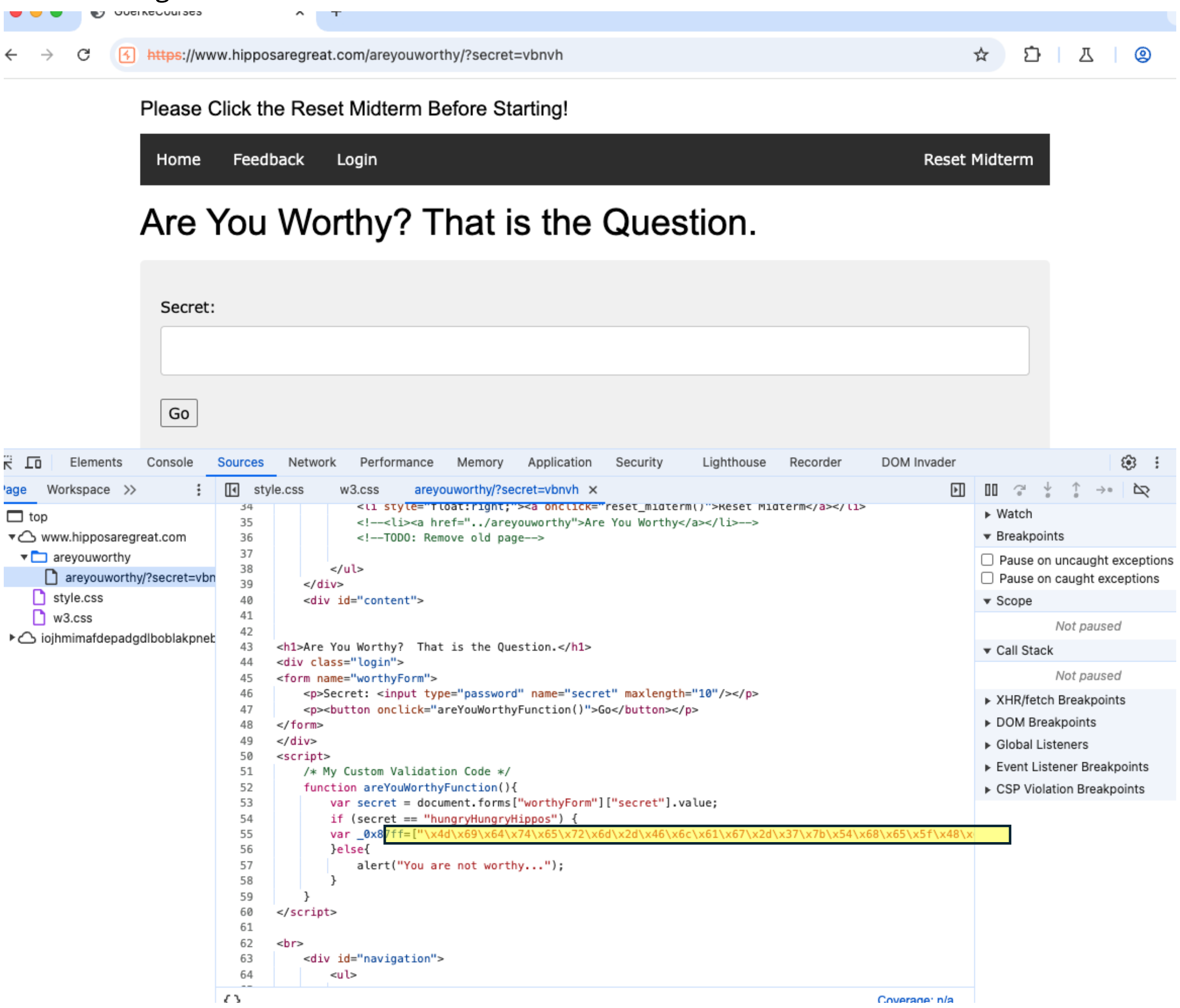
***Steps to Reproduce**:*

1. ***Method 1: Decoding the Hexadecimal Flag**:*
   - ***Step 1**: Navigated to the "Are You Worthy?" page, where the JavaScript validation function areYouWorthyFunction() was visible in the page source.*
   - ***Step 2**: Inspected the function and identified the hex-encoded string \x4d\x69\x64\x74\x65\x72\x6d\x2d\x46\x6c\x61\x67\x2d\x37\x7b\x54\x68\x65\x5f\x48\x69\x70\x70\x6f\x73\x5f\x41\x72\x65\x5f\x41\x6c\x77\x61\x79\x73\x5f\x57\x6f\x72\x74\x68\x79\x5f\x79\x6d\x79\x43\x76\x63\x7a\x62\x44\x39\x7d.*
   - ***Step 3**: Decoded the hex string, revealing the flag: Midterm-Flag-7{The_Hippos_Are_Always_Worthy_ymyCvczbD9}.*
2. ***Method 2: Modifying Input Length**:*

- ***Step 1**: On the "Are You Worthy?" page, located the input field for the "Secret" with a maxlength attribute set to 10, which limited the input length.*
- ***Step 2**: Opened Developer Tools and modified the maxlength attribute to 30.*
- ***Step 3**: Entered "hungryHungryHippos" (the expected code based on the JavaScript) into the input field and submitted it.*
- ***Step 4**: The page then displayed the flag: Midterm-Flag-7{The_Hippos_Are_Always_Worthy_ymyCvczbD9}.*

Description: Displaying the areYouWorthyFunction() JavaScript code, highlighting the hex-

encoded flag.



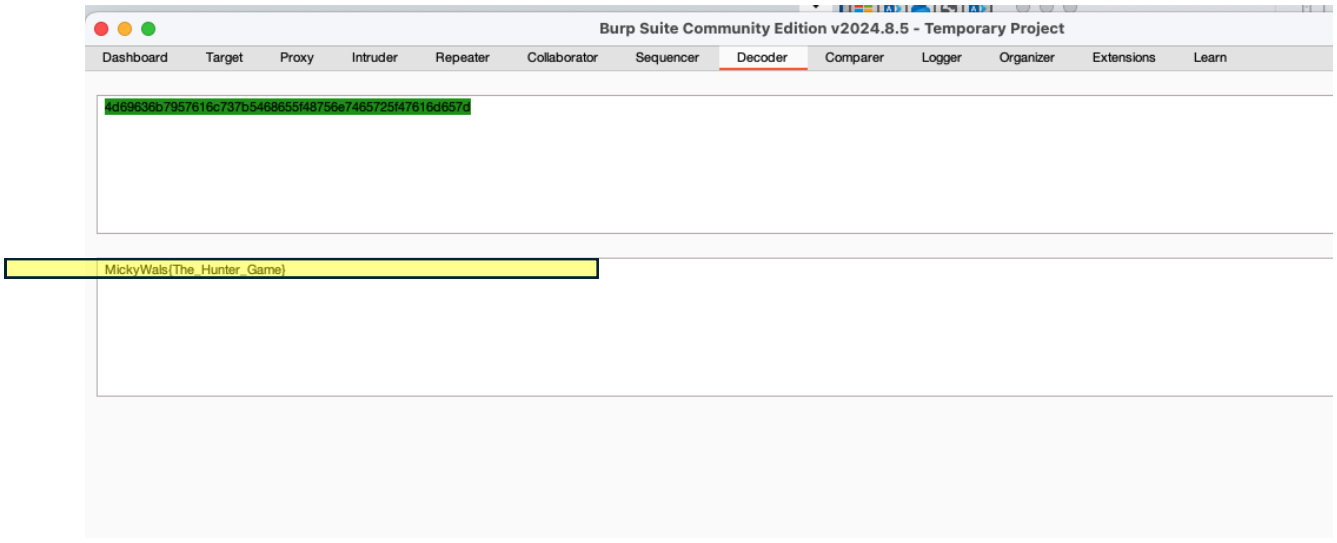
https://www.hipposaregreat.com/areyouworthy/?secret=vbnvh

Please Click the Reset Midterm Before Starting!

Are You Worthy? That is the Question.

```
            <!--<li><a href="../areyouworthy">Are You Worthy</a></li>-->
            <!--TODO: Remove old page-->

        </ul>
    </div>
    <div id="content">


<h1>Are You Worthy?  That is the Question.</h1>
<div class="login">
<form name="worthyForm">
    <p>Secret: <input type="password" name="secret" maxlength="10"/></p>
    <p><button onclick="areYouWorthyFunction()">Go</button></p>
</form>
</div>
<script>
    /* My Custom Validation Code */
    function areYouWorthyFunction(){
        var secret = document.forms["worthyForm"]["secret"].value;
        if (secret == "hungryHungryHippos") {
        var _0x87ff=["\x4d\x69\x64\x74\x65\x72\x6d\x2d\x46\x6c\x61\x67\x2d\x37\x7b\x54\x68\x65\x5f\x48\x
        }else{
            alert("You are not worthy...");
        }
    }
</script>

<br>
    <div id="navigation">
        <ul>
```

Screenshot 15 - JavaScript Code Inspection indicating a hex encoded string in the script

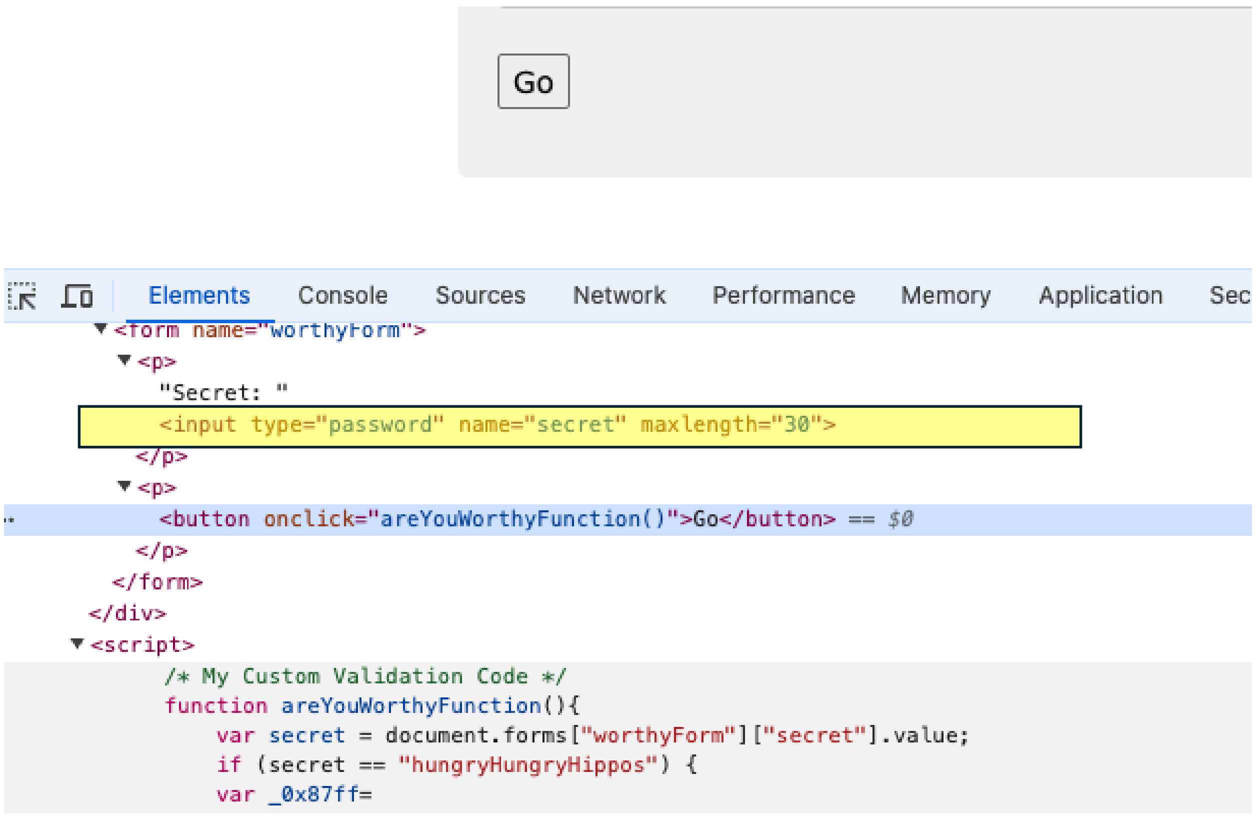


```
4d69636b7957616c737b5468655f48756e7465725f47616d657d
```

```
MickyWals{The_Hunter_Game}
```

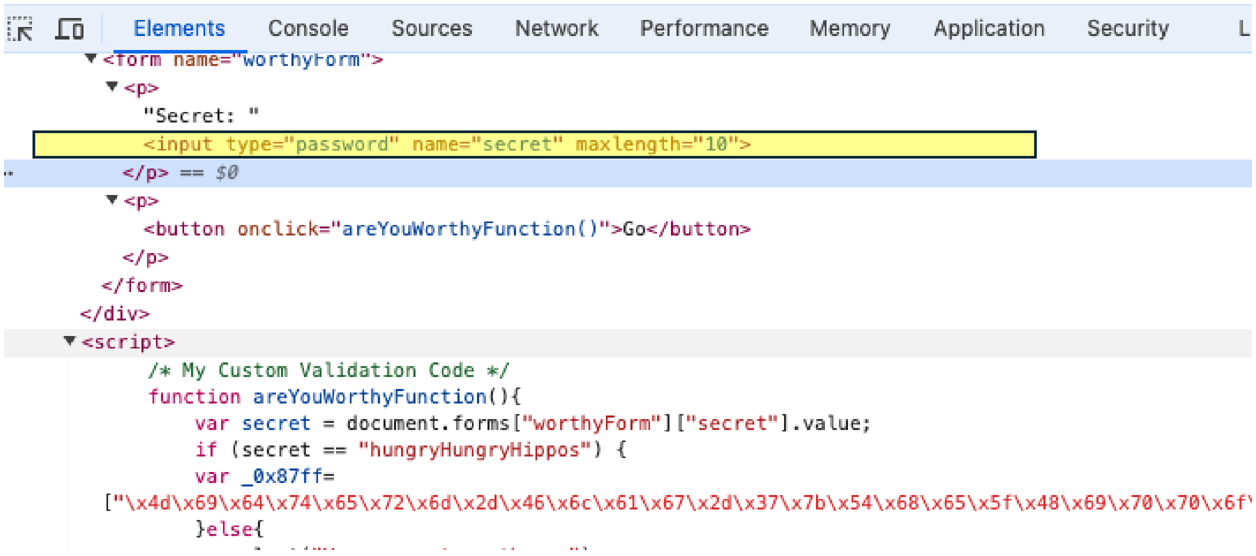
Screenshot 16 - Decoding the Hexadecimal String decoded indicates a Flag

Description: Decoding the string to obtain the flag directly from the JavaScript code.



```
▼<form name="worthyForm">
  ▼<p>
     "Secret: "
     <input type="password" name="secret" maxlength="30">
   </p>
  ▼<p>
     <button onclick="areYouWorthyFunction()">Go</button> == $0
   </p>
 </form>
</div>
▼<script>
        /* My Custom Validation Code */
        function areYouWorthyFunction(){
            var secret = document.forms["worthyForm"]["secret"].value;
            if (secret == "hungryHungryHippos") {
            var _0x87ff=
```

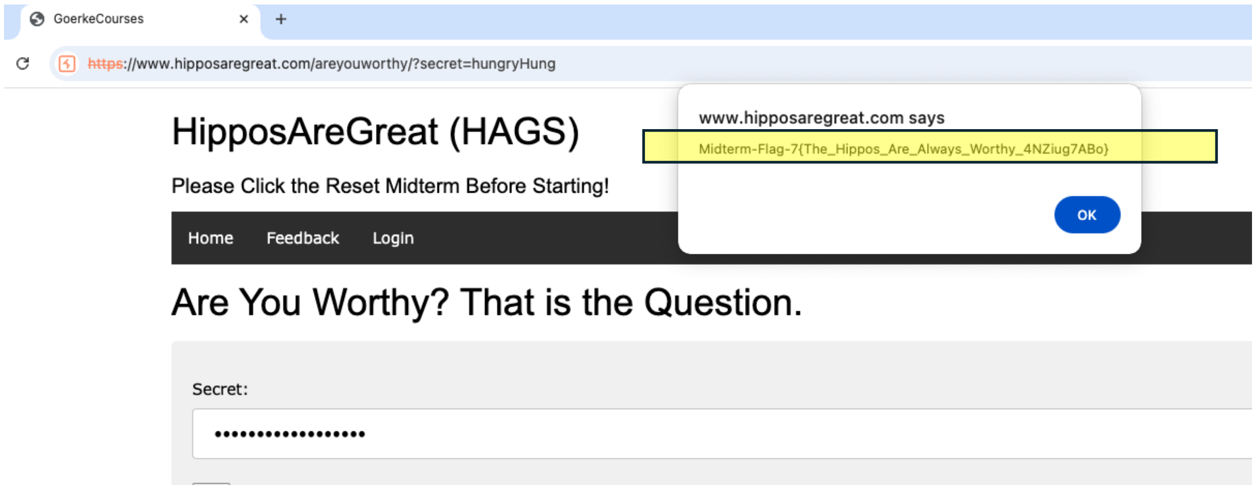
Screenshot 17 - Modifying Input Length 30 to allow for full password to be entered



```
▼<form name="worthyForm">
  ▼<p>
     "Secret: "
     <input type="password" name="secret" maxlength="10">
   </p> == $0
  ▼<p>
     <button onclick="areYouWorthyFunction()">Go</button>
   </p>
 </form>
</div>
▼<script>
       /* My Custom Validation Code */
       function areYouWorthyFunction(){
           var secret = document.forms["worthyForm"]["secret"].value;
           if (secret == "hungryHungryHippos") {
           var _0x87ff=
   ["\x4d\x69\x64\x74\x65\x72\x6d\x2d\x46\x6c\x61\x67\x2d\x37\x7b\x54\x68\x65\x5f\x48\x69\x70\x70\x6f
           }else{
```

Screenshot 18 - Modifying Input Length 10 from current to longer to accommodate full password

Description: Adjusting the maxlength attribute on the "Secret" input field to allow the full "hungryHungryHippos"code entry.


GoerkeCourses
https://www.hipposaregreat.com/areyouworthy/?secret=hungryHung
HipposAreGreat (HAGS)
Please Click the Reset Midterm Before Starting!
Home
Feedback
Login
Are You Worthy? That is the Question.
Secret:
OK

www.hipposaregreat.com says
Midterm-Flag-7{The_Hippos_Are_Always_Worthy_4NZiug7ABo}

Screenshot 19 - Flag Reveal for Midterm Flag 7

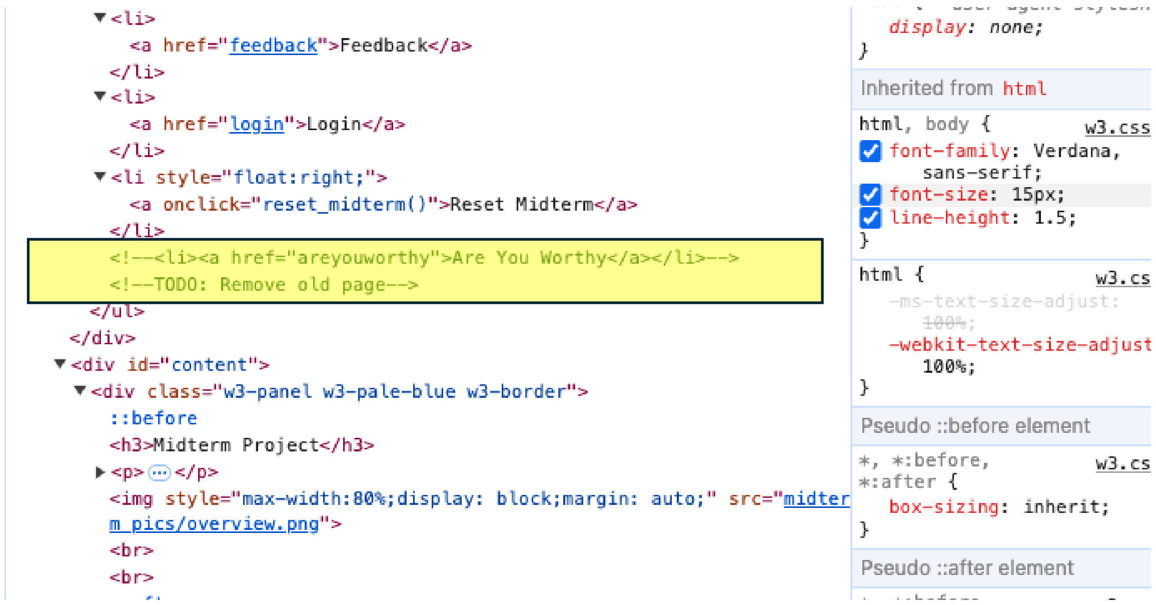
- *Description: After submitting the code with the modified input field, the page displays the flag.*

***Flag Found**:*
Midterm-Flag-7{The_Hippos_Are_Always_Worthy_ymyCvczbD9}

***Analysis**:*
This flag demonstrated two approaches: decoding a hex-encoded string directly from the JavaScript code and adjusting the HTML input constraints to allow for a complete code entry. The first method underscored the importance of recognizing encoded data in scripts, while the second method illustrated how input limitations could be bypassed by modifying HTML attributes. Both techniques highlight the need for robust server-side validation to prevent bypassing front-end constraints.

Flag 7 Discovery

Exploring the HTML Source for Clues



```
<li>
  <a href="feedback">Feedback</a>
</li>
<li>
  <a href="login">Login</a>
</li>
<li style="float:right;">
  <a onclick="reset_midterm()">Reset Midterm</a>
</li>
<!--<li><a href="areyouworthy">Are You Worthy</a></li>-->
<!--TODO: Remove old page-->
</ul>
</div>
<div id="content">
<div class="w3-panel w3-pale-blue w3-border">
  ::before
  <h3>Midterm Project</h3>
  <p>…</p>
  <img style="max-width:80%;display: block;margin: auto;" src="midterm_pics/overview.png">
  <br>
  <br>
```

```
  display: none;
}
Inherited from html
html, body {                w3.css
  font-family: Verdana,
      sans-serif;
  font-size: 15px;
  line-height: 1.5;
}
html {                      w3.cs
  -ms-text-size-adjust:
      100%;
  -webkit-text-size-adjust
      100%;
}
Pseudo ::before element
*, *:before,                w3.cs
*:after {
  box-sizing: inherit;
}
Pseudo ::after element
```

Screenshot 20 - HTML Source Code Exploration for reference to other hidden pages

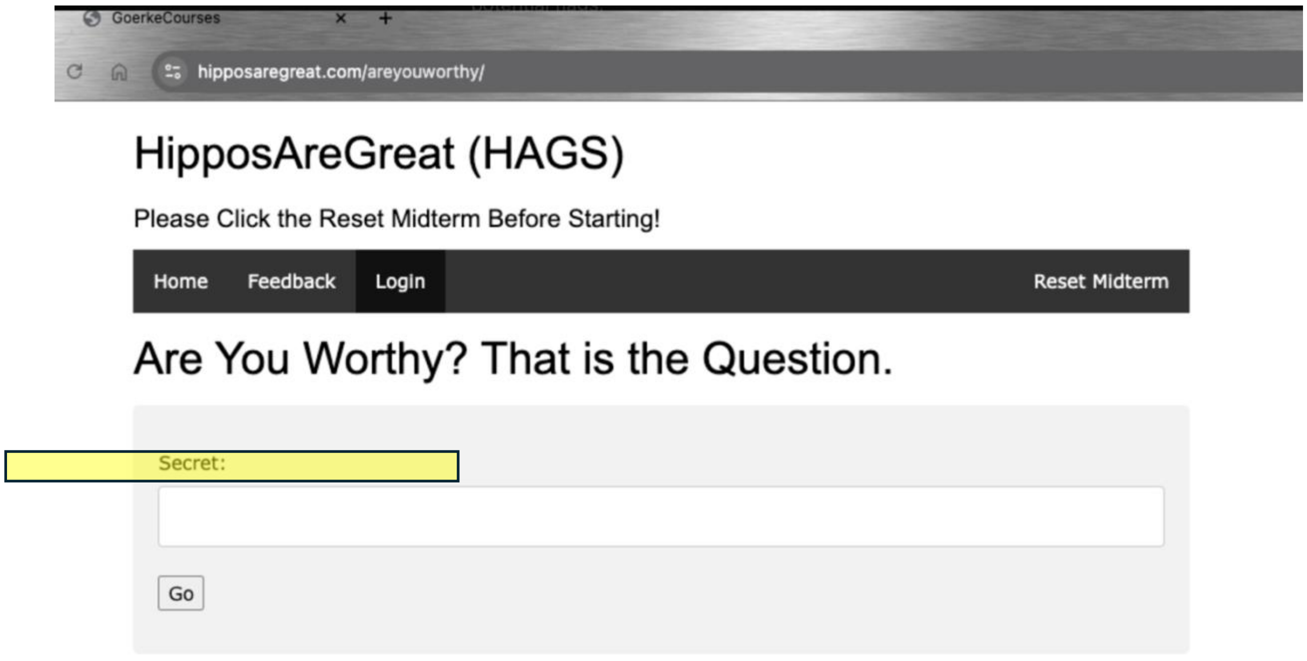
- **Description:** While inspecting the HTML source code of the main page, a hidden hyperlink (areyouworthy) was found in a commented section. The comment indicates that there may be an old page titled "Are You Worthy," which could potentially contain another hidden flag.

- **Clue Found:** Commented code pointing to areyouworthy with a message: "TODO: Remove old page."
- **Steps to Reproduce:**
  1. Opened the browser's developer tools and navigated to the "Elements" tab.
  2. Inspected the HTML code of the main page.
  3. Found a hidden link commented out in the navigation section, suggesting a possible page named "Are You Worthy."
- **Next Action:** Access the areyouworthy page by entering https://hipposaregreat.com/areyouworthy in the browser to explore for further clues or potential flags.
- **Analysis:** Hidden comments in HTML code can reveal links to legacy pages or areas under development. Checking these pages can often lead to the discovery of sensitive information or hidden flags in web applications.

MARTIN BISHOP & ASSOCIATES

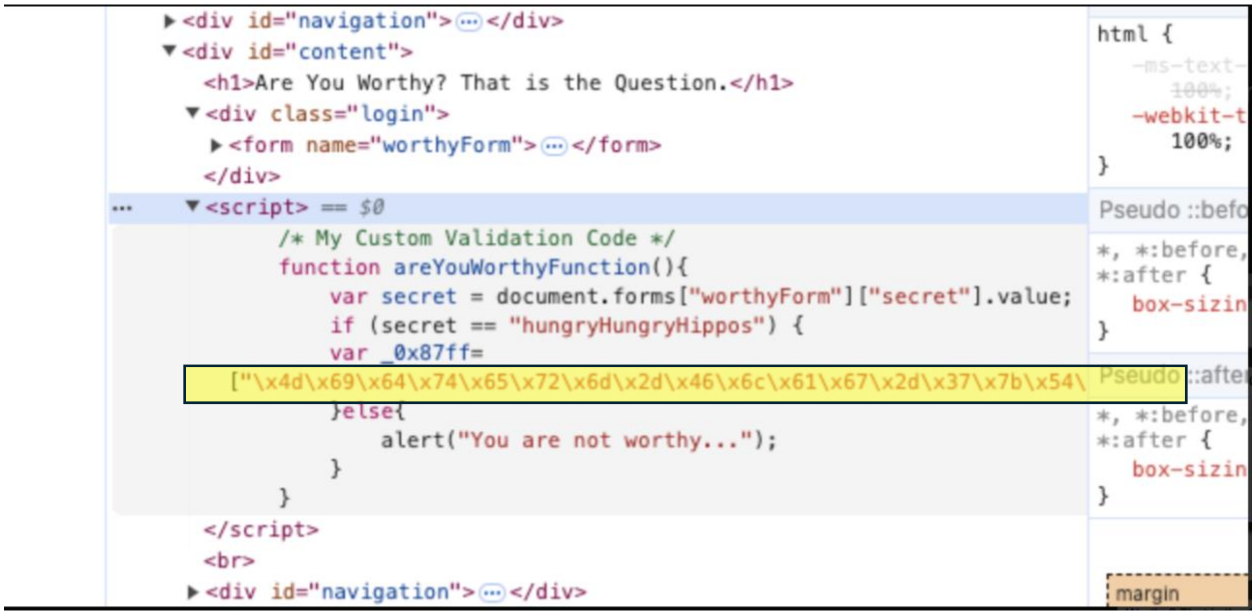
Penetration Testing and Security Solutions

A Subsidiary of **SPY PHANTOM IO**


GoerkeCourses
hipposaregreat.com/areyouworthy/
HipposAreGreat (HAGS)
Please Click the Reset Midterm Before Starting!
Home
Feedback
Login
Reset Midterm
Are You Worthy? That is the Question.
Secret:
Go

Screenshot 21 - HTML Source Code Exploration indicates something is possibly hidden in this page

- **Description:** After following the link discovered in the HTML source code, a page titled "Are You Worthy?" was found. This page includes a form labeled "Secret," prompting the user to input a code or password.
- **Action Taken:** Examined the page for any visible hints and prepared to inspect the source code for any embedded clues.

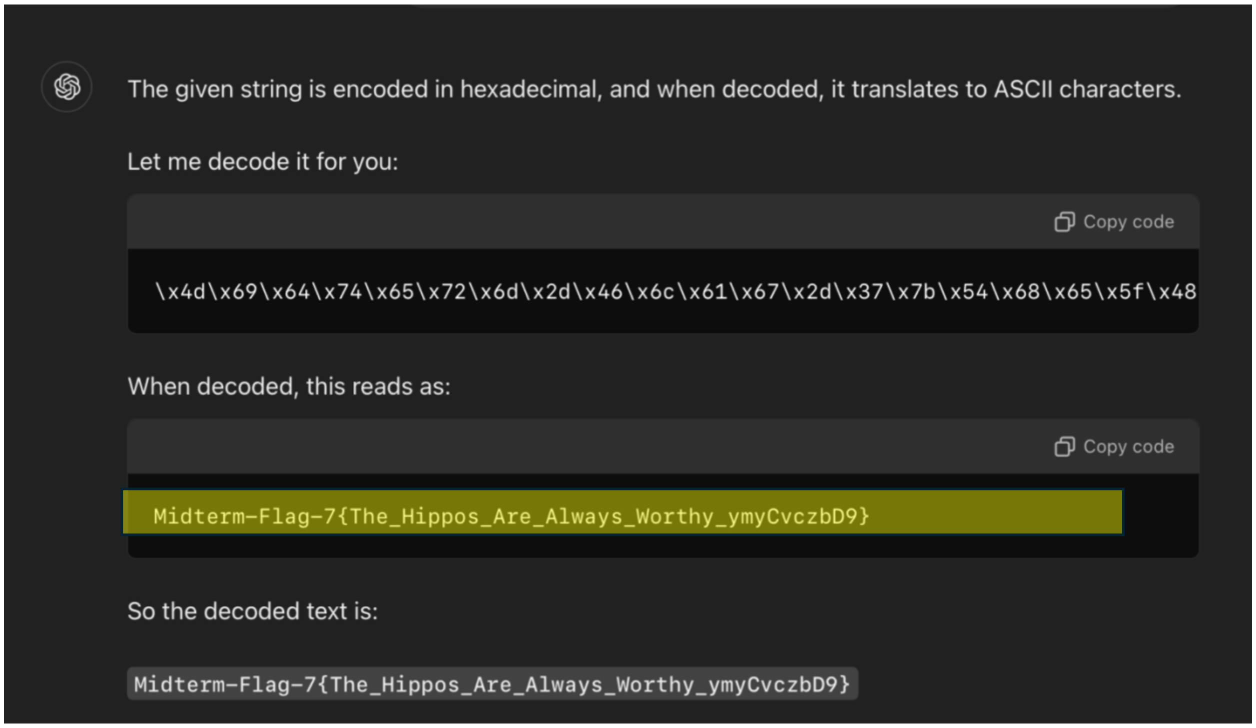


```
▶<div id="navigation">⋯</div>
▼<div id="content">
   <h1>Are You Worthy? That is the Question.</h1>
  ▼<div class="login">
    ▶<form name="worthyForm">⋯</form>
   </div>
  ▼<script> == $0
        /* My Custom Validation Code */
        function areYouWorthyFunction(){
            var secret = document.forms["worthyForm"]["secret"].value;
            if (secret == "hungryHungryHippos") {
            var _0x87ff=
      ["\x4d\x69\x64\x74\x65\x72\x6d\x2d\x46\x6c\x61\x67\x2d\x37\x7b\x54\
            }else{
                alert("You are not worthy...");
            }
        }
   </script>
   <br>
  ▶<div id="navigation">⋯</div>
```

```
html {
   -ms-text-
   100%;
   -webkit-t
   100%;
}
Pseudo ::befo
*, *:before,
*:after {
   box-sizin
}
Pseudo ::after
*, *:before,
*:after {
   box-sizin
}
margin
```

Screenshot 22 - JavaScript Code Inspection hex encoded message in the script

- **Description:** Upon inspecting the page source code, a JavaScript function areYouWorthyFunction() was found. This function validates the input against a secret code (hungryHungryHippos) and, if correct, returns a hex-encoded string.
- **Action Taken:** Identified hungryHungryHippos as the required input and noted the hex-encoded string. Decoding this string was necessary to reveal the flag.



The given string is encoded in hexadecimal, and when decoded, it translates to ASCII characters.

Let me decode it for you:

```
\x4d\x69\x64\x74\x65\x72\x6d\x2d\x46\x6c\x61\x67\x2d\x37\x7b\x54\x68\x65\x5f\x48
```

When decoded, this reads as:

```
Midterm-Flag-7{The_Hippos_Are_Always_Worthy_ymyCvczbD9}
```

So the decoded text is:

`Midterm-Flag-7{The_Hippos_Are_Always_Worthy_ymyCvczbD9}`

Screenshot 23 - Decoding the Hexadecimal Flag with ChatGPT 4.0

- **Description:** The hex-encoded string \x4d\x69\x64\x74\x65\x72\x6d\x2d\x46\x6c\x61\x67\x2d\x37\x7b\x54\x68\x65\x5f\x48\x69\x70\x70\x6f\x73\x5f\x41\x72\x65\x5f\x41\x6c\x77\x61\x79\x73\x5f\x57\x6f\x72\x74\x68\x79\x5f\x79\x6d\x79\x43\x76\x63\x7a\x62\x44\x39\x7d was decoded, resulting in the flag.
- **Flag Found:** Midterm-Flag-7{The_Hippos_Are_Always_Worthy_ymyCvczbD9}
- **Steps to Reproduce:**
    1. Navigated to https://hipposaregreat.com/areyouworthy.
    2. Inspected the JavaScript on the page to find the validation function and the required secret phrase.
    3. Captured the resulting hex-encoded string and decoded it to reveal the flag.

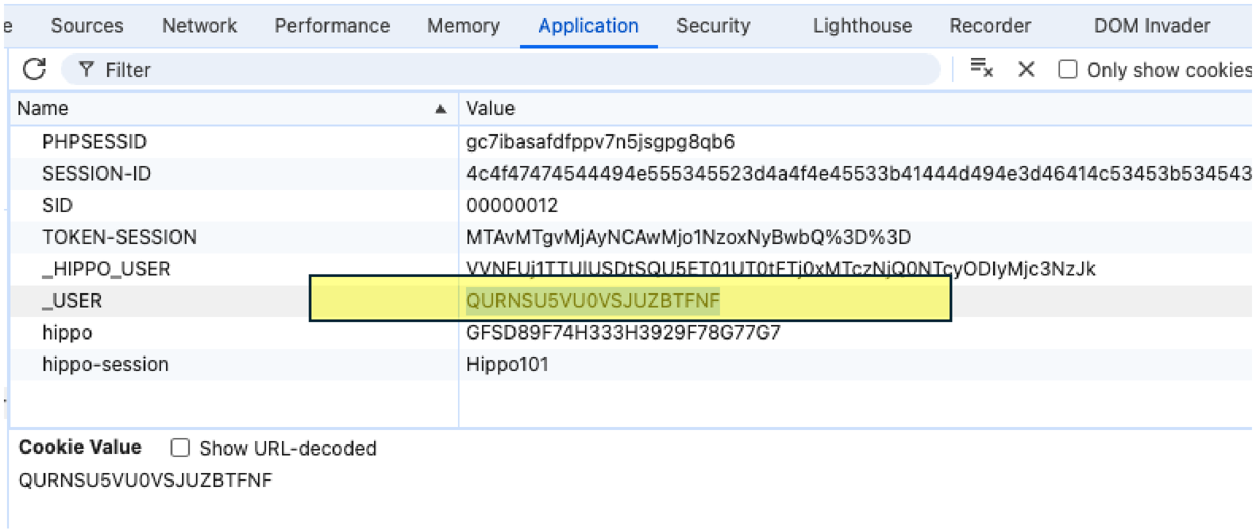
- **Analysis:** Using developer tools to inspect JavaScript can expose critical validation details and hidden data, as demonstrated here. This allowed for both bypassing front-end security and decoding sensitive information.

Flag 8 Discovery

Description: While analyzing the cookies of the web application, a cookie labeled _USER was found to be set to false for the admin privileges. Decoding the value revealed that the user did not have admin rights. By altering the cookie value and setting the admin access to True, I was able to access the restricted admin area and reveal the flag.

Steps to Reproduce:

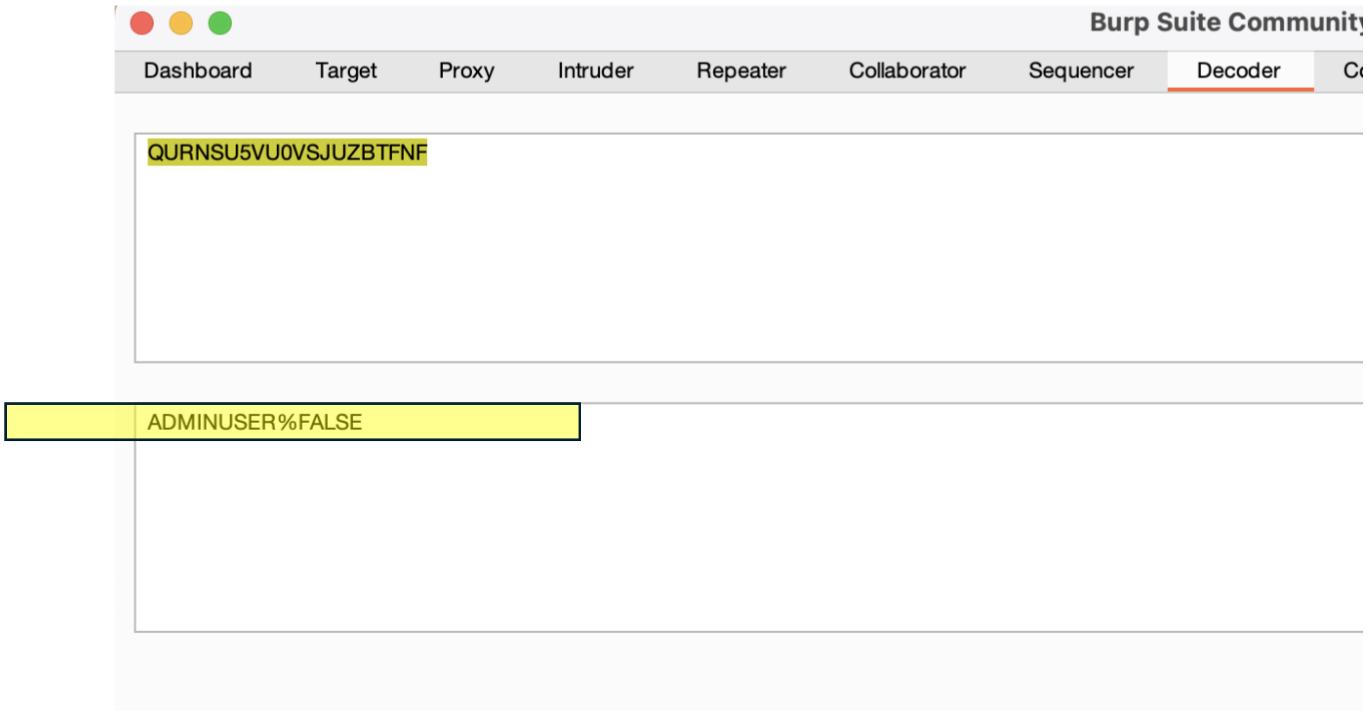
1. Open the developer tools on the web page, navigate to the Application tab, and locate the cookies.
2. Inspect the _USER cookie, which holds the key information about admin status.
3. Decode the cookie value using a base64 decoder.
4. Modify the cookie to change the ADMINUSER%FALSE value to ADMINUSER%TRUE.
5. Re-encode the modified value and set the altered cookie in the browser.
6. Refresh the page to access the admin area, revealing **Midterm Flag 8**.



Name	Value
PHPSESSID	gc7ibasafdfppv7n5jsgpg8qb6
SESSION-ID	4c4f47474544494e555345523d4a4f4e45533b41444d494e3d46414c53453b534543
SID	00000012
TOKEN-SESSION	MTAvMTgvMjAyNCAwMjo1NzoxNyBwbQ%3D%3D
_HIPPO_USER	VVNFUj1TTUlUSDtSQU5ET01UT0tFTj0xMTczNjQ0NTcyODIyMjc3NzJk
_USER	QURNSU5VU0VSJUZBTFNF
hippo	GFSD89F74H333H3929F78G77G7
hippo-session	Hippo101

Cookie Value ☐ Show URL-decoded
QURNSU5VU0VSJUZBTFNF

Screenshot 24 - Displaying the application cookies and the specific cookie holding the admin privilege information

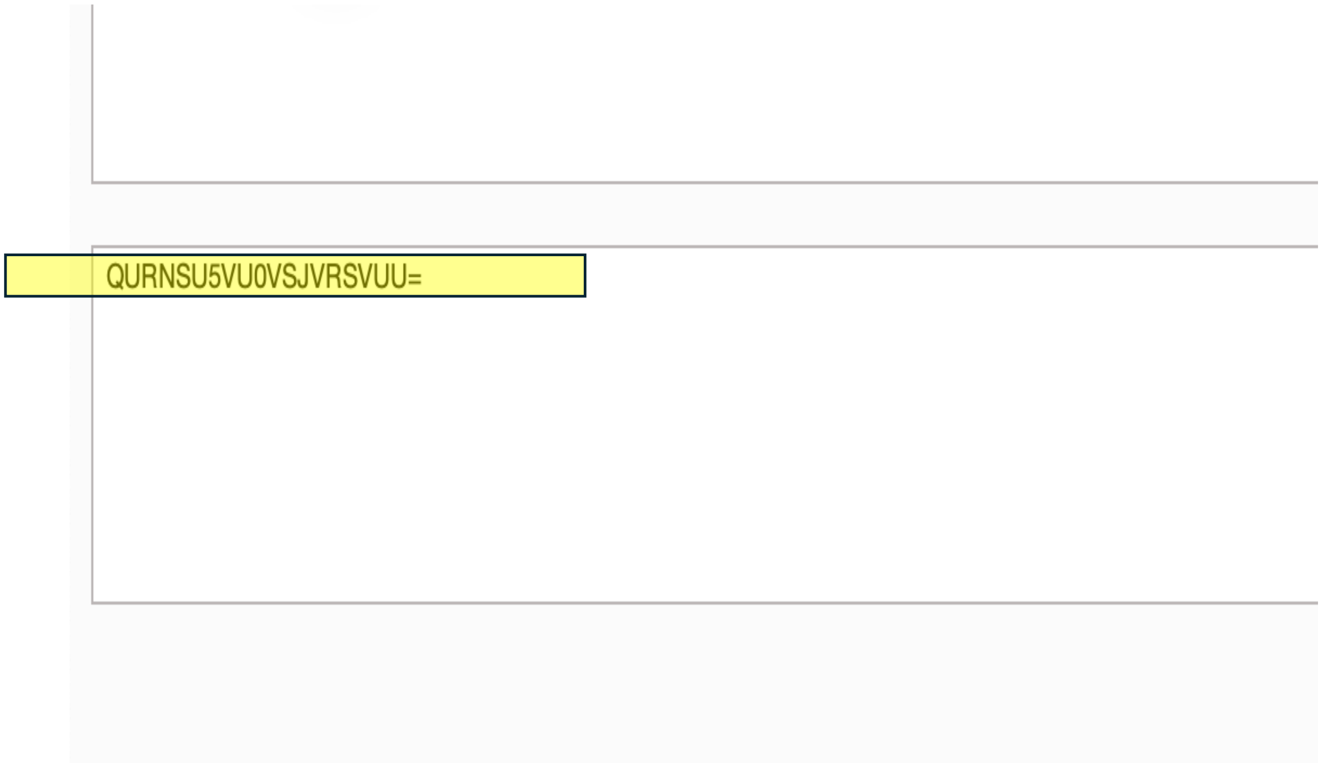


QURNSU5VU0VSJUZBTFNF

ADMINUSER%FALSE

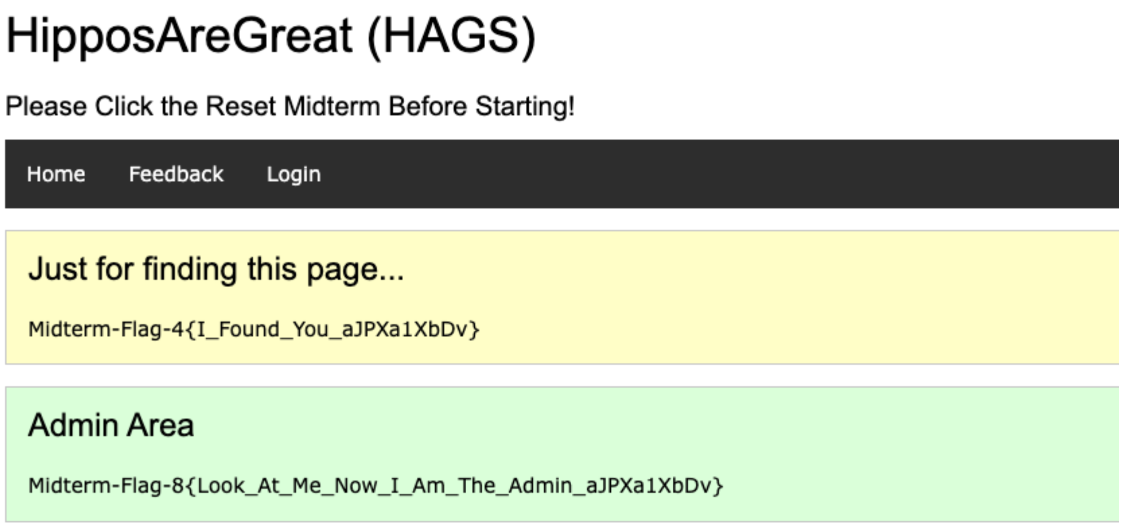
Screenshot 25 - Decoding the _USER cookie to identify the ADMINUSER%FALSE value

After modifying the cookie value to ADMINUSER%TRUE and refreshing the page, the admin area becomes accessible, revealing **Flag 8**.


QURNSU5VU0VSJVRSVUU=

Screenshot 26 - After modifying the cookie value to ADMINUSER%TRUE and refreshing the page, the admin area becomes accessible, revealing Flag 8

The final page showing **Flag 8**.


HipposAreGreat (HAGS)

Please Click the Reset Midterm Before Starting!

Just for finding this page...

Midterm-Flag-4{I_Found_You_aJPXa1XbDv}

Admin Area

Midterm-Flag-8{Look_At_Me_Now_I_Am_The_Admin_aJPXa1XbDv}

Screenshot 27 - The final page showing Flag

Flag Found: Midterm-Flag-8{Look_At_Me_Now_I_Am_The_Admin_aJPXa1XbDv}

Analysis: Cookie manipulation is a common way to bypass access restrictions in poorly secured web applications. By understanding how cookies store information and knowing how to decode and re-encode their values, attackers can often escalate privileges or bypass restrictions to access hidden content.

Flag 9 Discovery

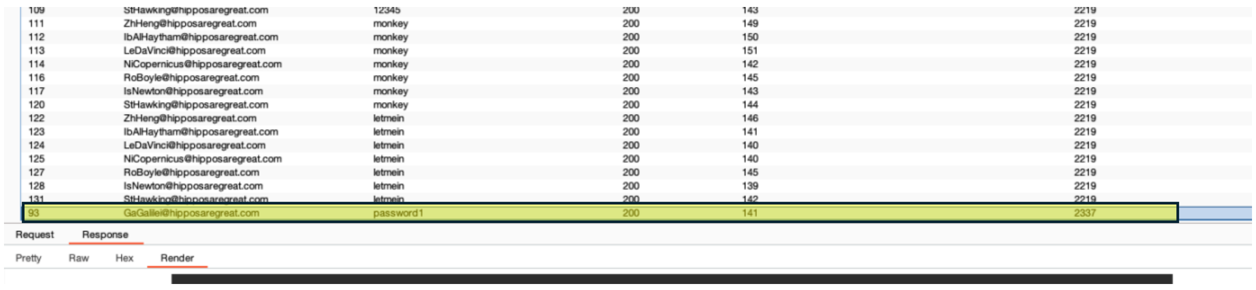
Description: After gathering a validated list of emails from the target system and having tried other approaches, a brute force method using Burp Suite's Intruder tool was employed to attack the login functionality. The goal was to identify a correct email and password combination that would log in a user and reveal the next flag.

Steps to Reproduce:

1. Gather a list of validated email addresses from previous recon efforts on the site.
2. Use a simple password list containing commonly used passwords.
3. Launch Burp Suite's Intruder tool and use the "Cluster Bomb" attack type to test combinations of the validated emails and password list.
4. Observe the server's responses, looking for a different status code or response length that indicates a successful login.
5. Upon finding a valid email and password combination, log in to the target account and capture the flag.

Burp Suite Intruder Setup

Description: Cluster Bomb attack setup with email and password payloads loaded in Burp Suite's Intruder tool.



109	StHawking@hipposaregreat.com	12345	200	143	2219
111	ZhHeng@hipposaregreat.com	monkey	200	149	2219
112	IbAlHaytham@hipposaregreat.com	monkey	200	150	2219
113	LeDaVinci@hipposaregreat.com	monkey	200	151	2219
114	NiCopernicus@hipposaregreat.com	monkey	200	142	2219
116	RoBoyle@hipposaregreat.com	monkey	200	145	2219
117	IsNewton@hipposaregreat.com	monkey	200	143	2219
120	StHawking@hipposaregreat.com	monkey	200	144	2219
122	ZhHeng@hipposaregreat.com	letmein	200	146	2219
123	IbAlHaytham@hipposaregreat.com	letmein	200	141	2219
124	LeDaVinci@hipposaregreat.com	letmein	200	140	2219
125	NiCopernicus@hipposaregreat.com	letmein	200	140	2219
127	RoBoyle@hipposaregreat.com	letmein	200	145	2219
128	IsNewton@hipposaregreat.com	letmein	200	139	2219
131	StHawking@hipposaregreat.com	letmein	200	142	2219
93	GaGalilei@hipposaregreat.com	password1	200	141	2337

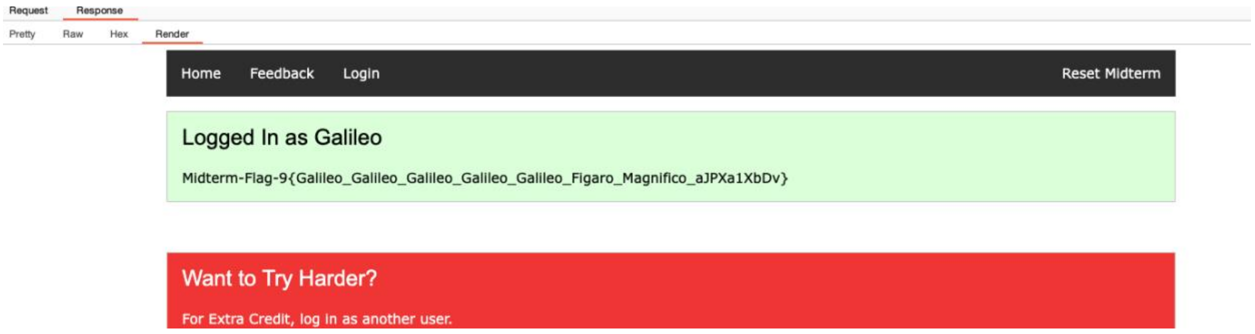
[redacted]

Screenshot 28 - Burp Suite Intruder GaGalileo@hipposaregreat.com password1 with a result 2337 indicting a strong change for success.

The brute-force attack was performed by using Burp Suite's Cluster Bomb feature, which tested combinations of a validated email list and a simple password list. The payloads were set to loop through each possible combination, and the response codes were analyzed to determine when a correct username/password pair was found. Specifically, the response code 200 indicated a successful login, and the credentials that triggered this response were used to capture Flag 9.

Successful Login and Flag 9 Discovery
Description: After the successful login as Galileo (GaGalileo@hipposaregreat.com), the flag Midterm-Flag-9 was revealed on the page.

Flag Found:
Midterm-Flag-9{Galileo_Galileo_Galileo_Galileo_Figaro_Magnifico_aJPXa1XbDv}



Logged In as Galileo

Midterm-Flag-9{Galileo_Galileo_Galileo_Galileo_Galileo_Figaro_Magnifico_aJPXa1XbDv}

Want to Try Harder?

For Extra Credit, log in as another user.

Screenshot 29 - Successful Login and Flag 9 Discovery

Analysis:
By using a brute force technique that involved testing multiple combinations of emails and passwords, the login functionality was compromised. This flag demonstrates the effectiveness of combining social engineering (gathering valid emails) with brute force attacks to exploit weak password policies. The simplicity of the password "password1" reinforces the importance of enforcing stronger password requirements to prevent similar attacks.

MARTIN BISHOP & ASSOCIATES

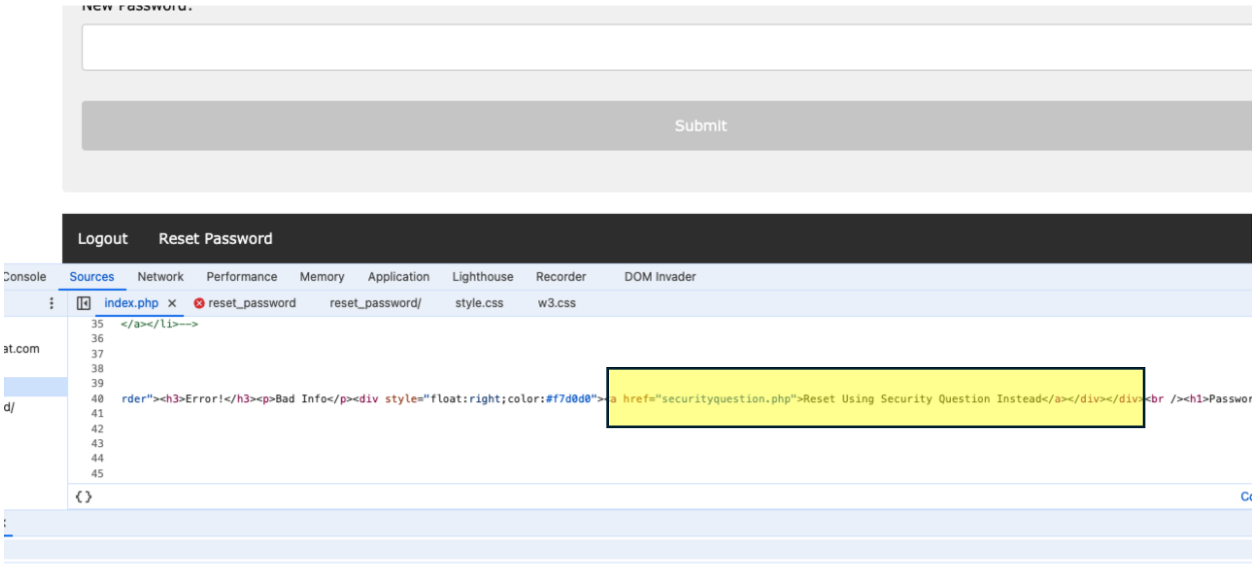
Penetration Testing and Security Solutions

A Subsidiary of **SPY PHANTOM IO**

5. Appendix

Extra Credit Flag Discovery

"Password Reset Page Inspection"

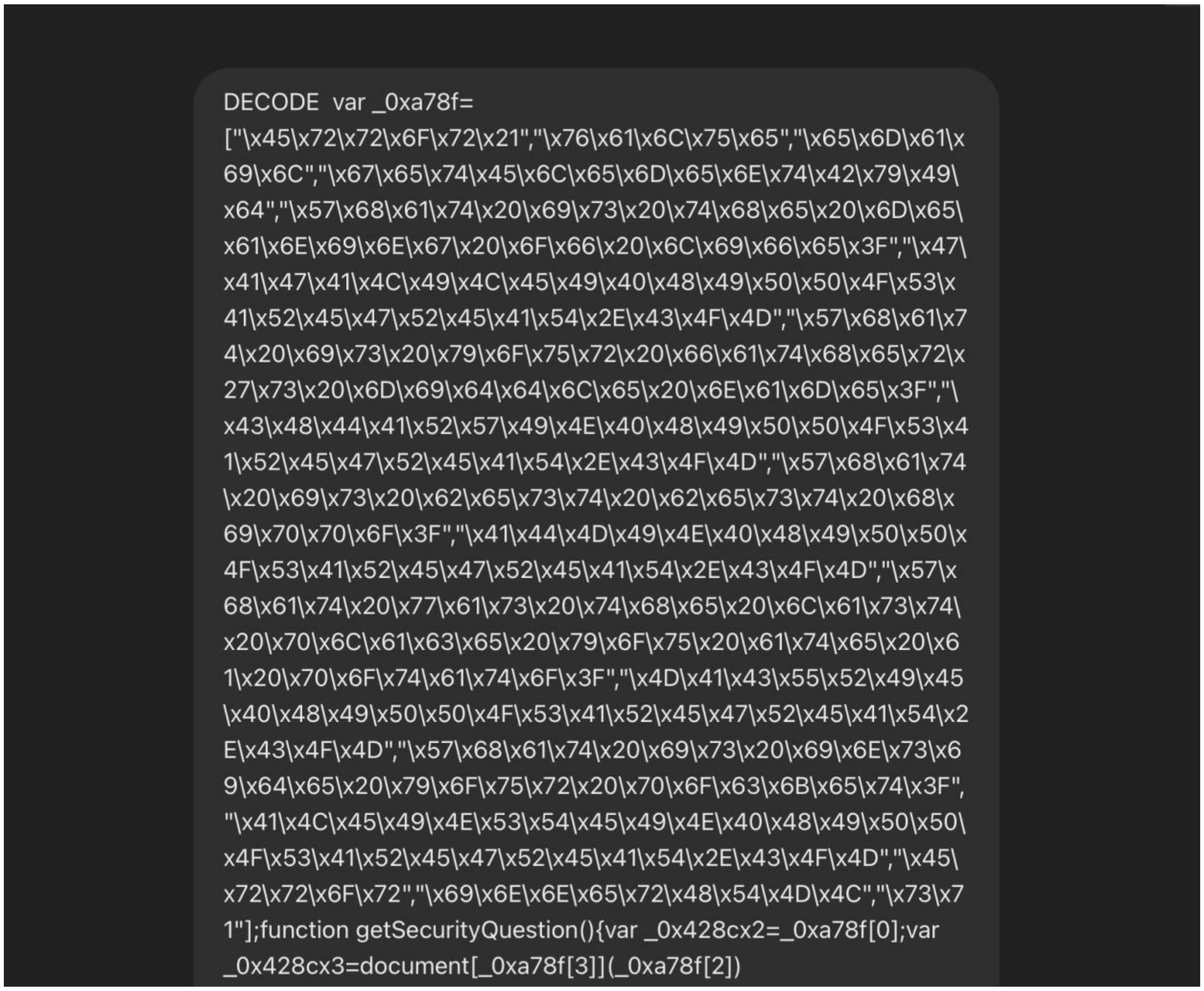


```
</a></li>-->




rder"><h3>Error!</h3><p>Bad Info</p><div style="float:right;color:#f7d0d0"><a href="securityquestion.php">Reset Using Security Question Instead</a></div></div><br /><h1>Passwor
```

Screenshot 30 - Password Reset Page Inspection shows hidden page reference securityquestion.php

Description: While attempting to log in as another user, the password reset page was examined. The "Reset Using Security Question" link was found in the HTML source code of the login page, directing the user to the security question reset feature. **Action Taken:** Inspected the page source for any additional hints and accessed the security question reset functionality.

JavaScript Code Inspection for Security Questions

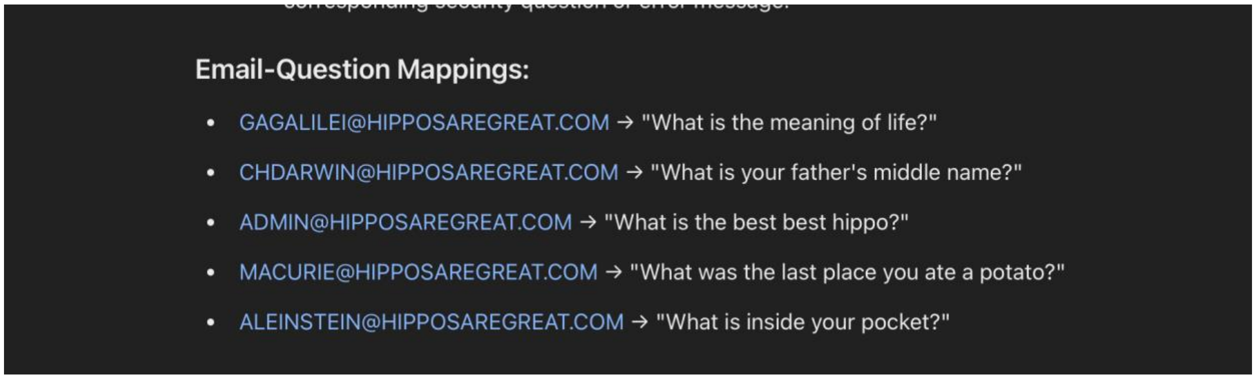


```
DECODE  var _0xa78f=
["\x45\x72\x72\x6F\x72\x21","\x76\x61\x6C\x75\x65","\x65\x6D\x61\x
69\x6C","\x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\
x64","\x57\x68\x61\x74\x20\x69\x73\x20\x74\x68\x65\x20\x6D\x65\
x61\x6E\x69\x6E\x67\x20\x6F\x66\x20\x6C\x69\x66\x65\x3F","\x47\
x41\x47\x41\x4C\x49\x4C\x45\x49\x40\x48\x49\x50\x50\x4F\x53\x
41\x52\x45\x47\x52\x45\x41\x54\x2E\x43\x4F\x4D","\x57\x68\x61\x7
4\x20\x69\x73\x20\x79\x6F\x75\x72\x20\x66\x61\x74\x68\x65\x72\x
27\x73\x20\x6D\x69\x64\x64\x6C\x65\x20\x6E\x61\x6D\x65\x3F","\
x43\x48\x44\x41\x52\x57\x49\x4E\x40\x48\x49\x50\x50\x4F\x53\x4
1\x52\x45\x47\x52\x45\x41\x54\x2E\x43\x4F\x4D","\x57\x68\x61\x74
\x20\x69\x73\x20\x62\x65\x73\x74\x20\x62\x65\x73\x74\x20\x68\x
69\x70\x70\x6F\x3F","\x41\x44\x4D\x49\x4E\x40\x48\x49\x50\x50\x
4F\x53\x41\x52\x45\x47\x52\x45\x41\x54\x2E\x43\x4F\x4D","\x57\x
68\x61\x74\x20\x77\x61\x73\x20\x74\x68\x65\x20\x6C\x61\x73\x74\
x20\x70\x6C\x61\x63\x65\x20\x79\x6F\x75\x20\x61\x74\x65\x20\x6
1\x20\x70\x6F\x74\x61\x74\x6F\x3F","\x4D\x41\x43\x55\x52\x49\x45
\x40\x48\x49\x50\x50\x4F\x53\x41\x52\x45\x47\x52\x45\x41\x54\x2
E\x43\x4F\x4D","\x57\x68\x61\x74\x20\x69\x73\x20\x69\x6E\x73\x6
9\x64\x65\x20\x79\x6F\x75\x72\x20\x70\x6F\x63\x6B\x65\x74\x3F",
"\x41\x4C\x45\x49\x4E\x53\x54\x45\x49\x4E\x40\x48\x49\x50\x50\
x4F\x53\x41\x52\x45\x47\x52\x45\x41\x54\x2E\x43\x4F\x4D","\x45\
x72\x72\x6F\x72","\x69\x6E\x6E\x65\x72\x48\x54\x4D\x4C","\x73\x7
1"];function getSecurityQuestion(){var _0x428cx2=_0xa78f[0];var
_0x428cx3=document[_0xa78f[3]](_0xa78f[2])
```

Screenshot 31- JavaScript Code Inspection for Security Questions decoding the hexadecimal values

- **Description**: While inspecting the page, a custom validation function was discovered in the source code, encoded with hexadecimal values. This script defines the method to verify security questions.
- **Action Taken**: The encoded string was copied for further decoding to reveal the actual security questions.

Decoding the JavaScript Code



corresponding security question or error message.

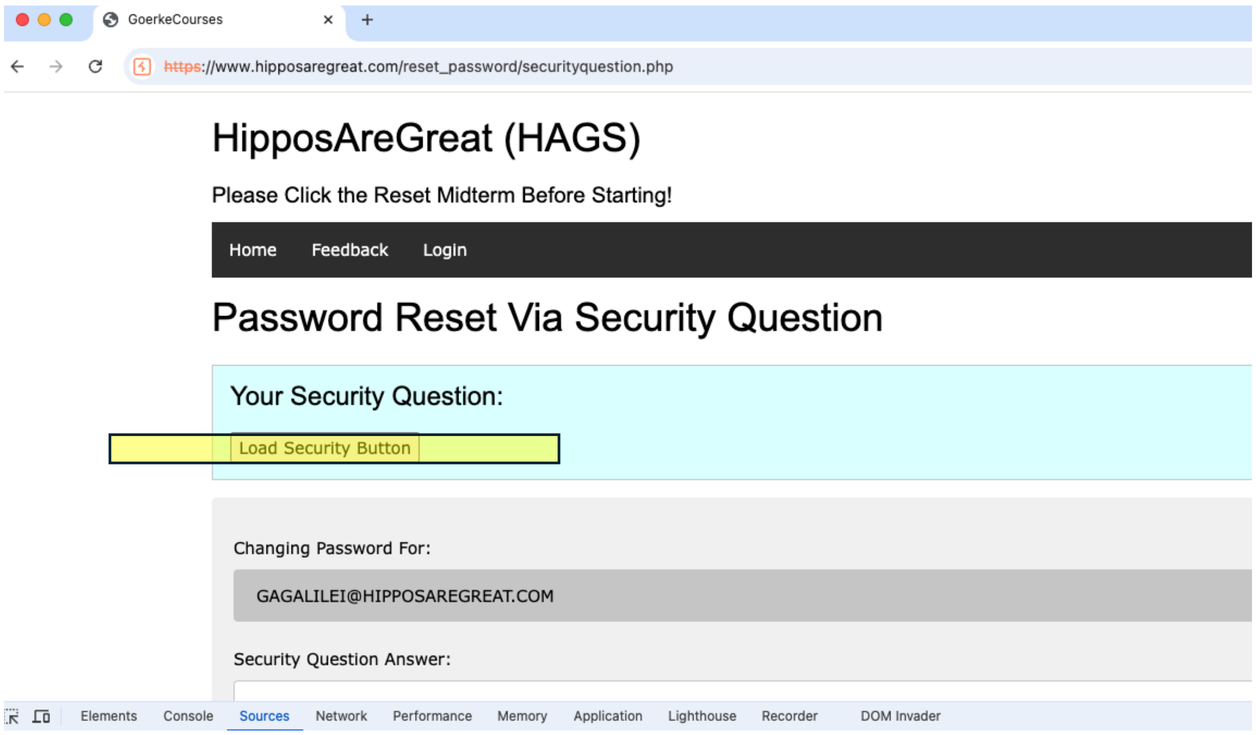
Email-Question Mappings:

- GAGALILEI@HIPPOSAREGREAT.COM → "What is the meaning of life?"
- CHDARWIN@HIPPOSAREGREAT.COM → "What is your father's middle name?"
- ADMIN@HIPPOSAREGREAT.COM → "What is the best best hippo?"
- MACURIE@HIPPOSAREGREAT.COM → "What was the last place you ate a potato?"
- ALEINSTEIN@HIPPOSAREGREAT.COM → "What is inside your pocket?"

Screenshot 32 - Decoding the JavaScript Code to reveal the email password reset questions

- **Description**: Using a decoding tool, the hexadecimal string was decoded, revealing human-readable text including the exact security questions that needed to be answered for password resets.
- **Action Taken**: The decoded security questions were used to gather necessary data for resetting user passwords.

"Security Question Form"



HipposAreGreat (HAGS)

Please Click the Reset Midterm Before Starting!

Home Feedback Login

Password Reset Via Security Question

Your Security Question:

Load Security Button

Changing Password For:

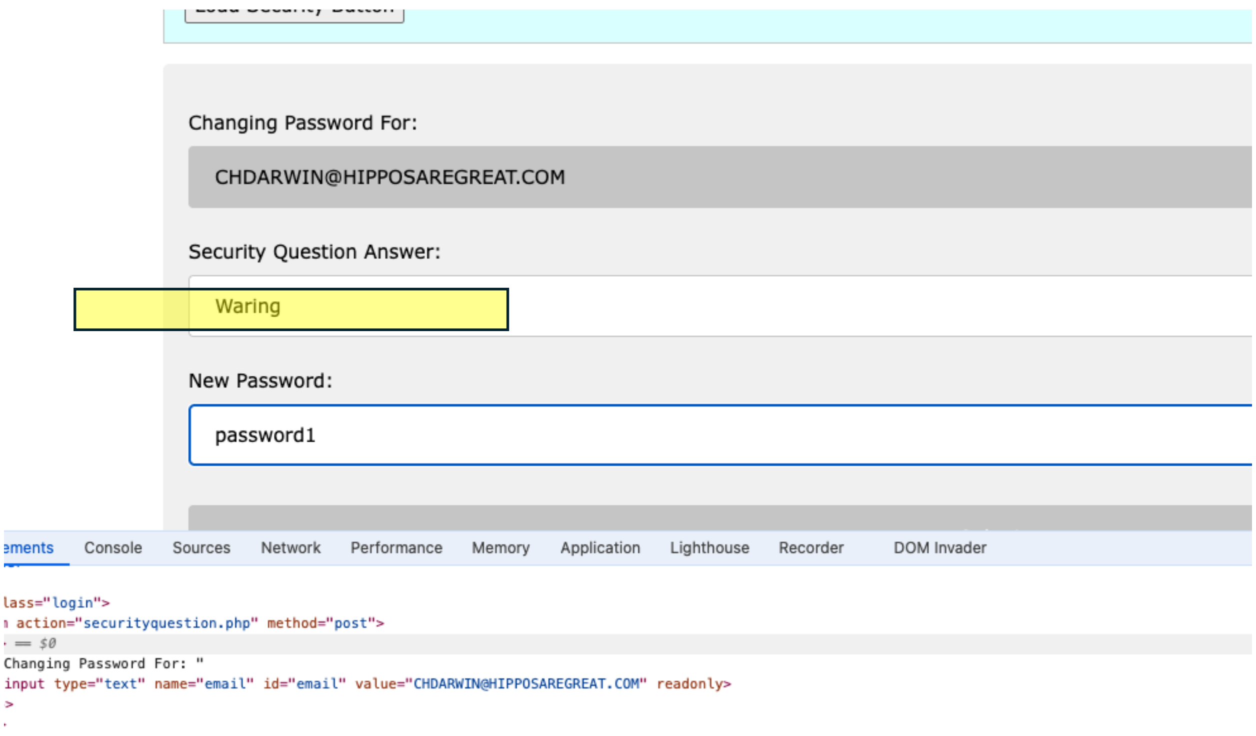
GAGALILEI@HIPPOSAREGREAT.COM

Security Question Answer:

Screenshot 33 - Security Question Form options to load the security question for the changed user

Description: Upon visiting the reset password page for Charles Darwin's account, the form asked for a security question response. Using publicly available information, Charles Darwin's father's middle name (Waring) was identified as the likely answer. **Action Taken:** Input the correct security question answer and prepared to reset the password.

Password Reset



Changing Password For:

CHDARWIN@HIPPOSAREGREAT.COM

Security Question Answer:

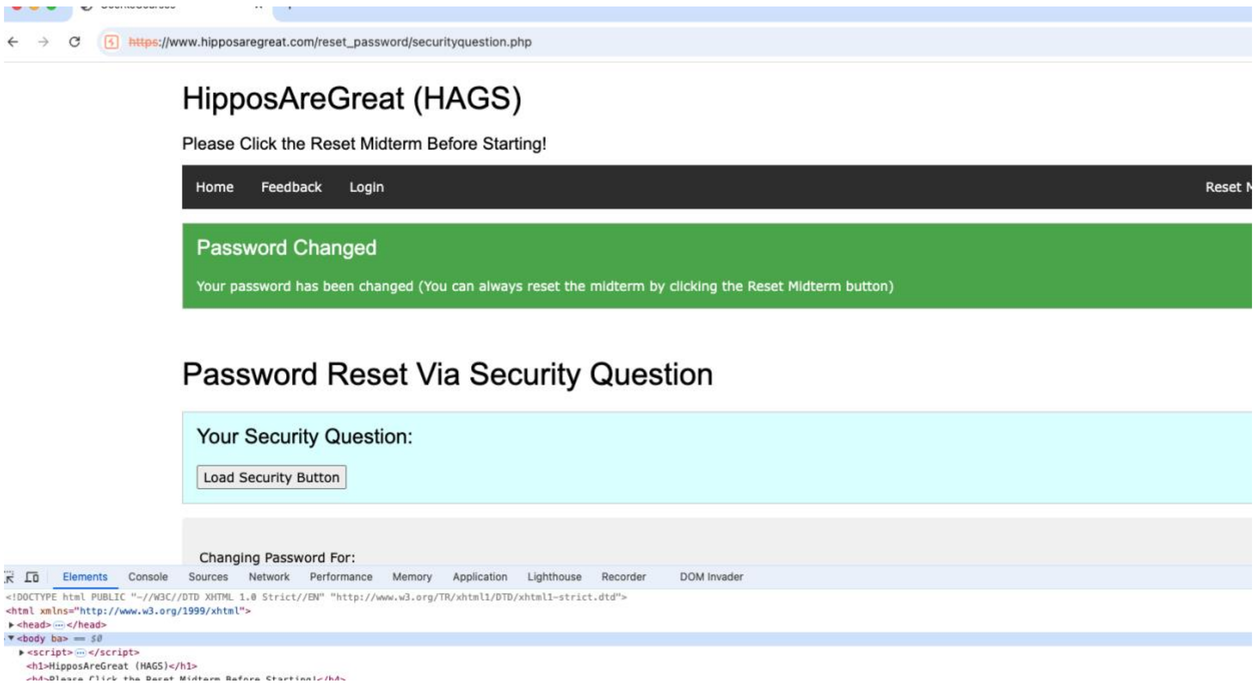
Waring

New Password:

password1

```
lass="login">
n action="securityquestion.php" method="post">
 == $0
Changing Password For: "
input type="text" name="email" id="email" value="CHDARWIN@HIPPOSAREGREAT.COM" readonly>
>
```

Screenshot 34 - Password Reset Attempt with the social engineer discovery of fathers middle name


https://www.hipposaregreat.com/reset_password/securityquestion.php

HipposAreGreat (HAGS)

Please Click the Reset Midterm Before Starting!

Password Changed

Your password has been changed (You can always reset the midterm by clicking the Reset Midterm button)

Password Reset Via Security Question

Your Security Question:

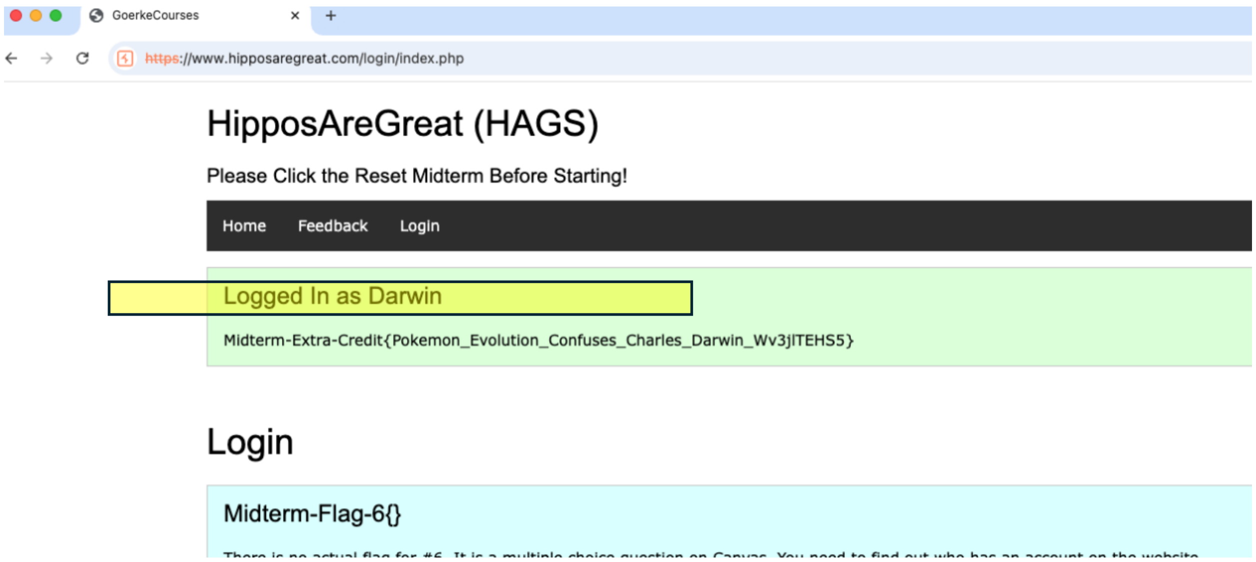
Changing Password For:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 ▸<head>⋯</head>
 ▾<body ba> == $0
   ▸<script>⋯</script>
    <h1>HipposAreGreat (HAGS)</h1>
    <h4>Please Click the Reset Midterm Before Starting!</h4>
```

Screenshot 35 - Successful Password Reset

Description: After answering the security question correctly, the form allowed for the password to be reset to a simple string (e.g., "password1"). This allowed access to Charles Darwin's account. **Action Taken:** Reset the password for Darwin's account and attempted login.

"Logged in as Charles Darwin"



HipposAreGreat (HAGS)

Please Click the Reset Midterm Before Starting!

Logged In as Darwin

Midterm-Extra-Credit{Pokemon_Evolution_Confuses_Charles_Darwin_Wv3jITEHS5}

Login

Midterm-Flag-6{}

Screenshot 36 - Logged in as Charles Darwin and reveal extra credit flag

Description: Successfully logged into Charles Darwin's account after resetting the password. This action revealed the extra credit flag as a reward for the exploit. **Result:** Obtained the Midterm-Extra-Credit flag, which reads: Midterm-Extra-Credit{Pokemon_Evolution_Confuses_Charles_Darwin_Wv3jITEHS5}.

Steps to Reproduce:

1. **Inspect the login page HTML and locate the "Reset Using Security Question" link**:
   - Use browser Developer Tools to inspect the HTML of the login page. Look for hidden or obfuscated links related to password reset functionality, particularly one mentioning "Reset Using Security Question."
2. **Analyze the JavaScript for encoded security question validation**:
   - Once you have located the security question reset page, inspect the associated JavaScript file using Developer Tools. In this script, you will find

the logic that handles security question validation, including encoded security questions.

- **Decode the encoded information**: The questions may be stored in an encoded format (e.g., Base64 or hexadecimal). Use a decoding tool (such as ChatGPT or Python's built-in decoding functions) to convert the encoded string into readable text. This reveals the security questions for the user accounts.

3. **Use the decoded questions to navigate the security question reset page**:
    - Using the decoded security question for Charles Darwin's account, navigate to the password reset page and attempt to answer the question correctly. In this case, the answer to Darwin's security question (his father's middle name) is **Waring**.
4. **Input the correct answer for Charles Darwin's security question (Waring)**:
    - Enter "Waring" as the answer to the security question.
5. **Reset the password and log in with the new password**:
    - Once the correct answer has been submitted, you will be prompted to reset the password. Set a new password for Darwin's account and use it to log in.
6. **Capture the extra credit flag once logged in**:
    - After logging into Charles Darwin's account, the extra credit flag will be displayed. Capture the flag and document the steps in your report.

By inspecting the JavaScript source code of the password reset page, an encoded security question was discovered. The encoded string was in Base64 format and was decoded using ChatGPT to reveal the full security question. Social engineering techniques were used to determine the answer to Charles Darwin's security question, which was 'Waring' (his father's middle name). This allowed the password reset and subsequent capture of the extra credit flag.

Analysis:

This method exposed how a password reset mechanism using security questions can be exploited, particularly when questions are based on publicly accessible information. Social engineering and information gathering played a key role in bypassing account security and obtaining the extra credit flag.

---

5. Final Recommendations for HAGS

1. ### Implement Stronger Password Policies:
   - **Issue**: Weak password policies allowed for brute-force and dictionary attacks to succeed.
   - **Recommendation**: Enforce strong password requirements, such as a minimum length of 12 characters, and require a mix of upper- and lowercase letters, numbers, and special characters. Implement account lockout mechanisms after a certain number of failed login attempts to prevent automated brute-force attacks.
2. ### Disable Directory Browsing and Secure Hidden Directories:
   - **Issue**: Directory traversal attacks revealed hidden directories like /admin_area/, which exposed sensitive information and provided unauthorized access.
   - **Recommendation**: Disable directory browsing on the web server and restrict access to sensitive directories via proper authentication and role-based access controls (RBAC). Additionally, use obfuscation or remove unnecessary directories entirely.
3. ### Strengthen Cookie Security and Validation:
   - **Issue**: Manipulation of session cookies (such as the _USER cookie) allowed privilege escalation to admin roles.

- **Recommendation**: Secure session cookies with proper encryption (e.g., using HttpOnly and Secure flags), and ensure that sensitive information, such as user roles, is validated on the server side rather than relying solely on client-side cookies. Regularly invalidate session tokens.

4. Improve Client-Side and Server-Side Input Validation:
   - **Issue**: Client-side input validation was easily bypassed, allowing for the submission of invalid or malicious data, such as entering longer passwords or hidden field manipulation.
   - **Recommendation**: Implement robust input validation on both the client and server sides. Never rely solely on client-side validation to protect sensitive operations. Use server-side validation to ensure all input is properly sanitized and verified before being processed.
5. Harden JavaScript Security:
   - **Issue**: JavaScript validation functions revealed sensitive information, such as encoded flag data and password validation logic.
   - **Recommendation**: Obfuscate or minimize sensitive client-side code, and ensure that any critical validation or security checks are handled server-side. Avoid embedding sensitive data in JavaScript, and regularly review front-end code for potential security leaks.
6. Implement Rate Limiting and CAPTCHA on Login Forms:
   - **Issue**: Brute-force attacks on the login form were successful due to the lack of rate-limiting mechanisms.
   - **Recommendation**: Implement rate limiting on all login attempts and use CAPTCHA to prevent automated login attempts. Additionally, track IP addresses and alert admins when there is suspicious login behavior, such as a high volume of failed attempts.
7. Conduct Regular Security Audits and Penetration Tests:

- **Issue**: Several vulnerabilities were found that could have been detected earlier through proper security assessments.
- **Recommendation**: Conduct regular security audits and penetration tests, especially after deploying new features or changes to the application. This will ensure that new vulnerabilities are detected and remediated before they can be exploited by malicious actors.

8. Secure Password Reset Functionality:
    - **Issue**: Security questions used for password resets were based on easily researchable public information, leading to unauthorized password changes.
    - **Recommendation**: Use more secure, non-public information for security questions, or implement multi-factor authentication (MFA) for password resets. Additionally, ensure that security answers are properly hashed and stored securely.
9. Encrypt All Sensitive Data at Rest and in Transit:
    - **Issue**: Some sensitive data was found in plaintext in cookies and JavaScript.
    - **Recommendation**: Ensure that all sensitive data (including passwords, session tokens, and personal information) is encrypted at rest and in transit using industry-standard encryption protocols (e.g., TLS 1.2 or higher). Regularly rotate encryption keys and review encryption practices for compliance with best practices.
10. Apply Role-Based Access Control (RBAC):
    - **Issue**: Privilege escalation vulnerabilities allowed regular users to elevate their permissions to admin.
    - **Recommendation**: Apply strict role-based access control (RBAC) to ensure users can only access data and functionality specific to their roles. Regularly audit role assignments and limit admin access to a minimum number of users.

Conclusion

By implementing these recommendations, HAGS can significantly improve the security of their web application. The most critical issues, including weak password policies, insecure cookies, and improper validation mechanisms, expose the system to serious risks. Addressing these vulnerabilities will reduce the attack surface and protect both the application and user data from exploitation.

Acknowledgments

In addition to manual penetration testing techniques, ChatGPT was used throughout the project to assist in multiple capacities. This included:

- Decoding assistance: When faced with encoded strings, ChatGPT was used as a decoder to quickly translate Base64 and hexadecimal strings into readable text, which facilitated further exploration of the application's vulnerabilities.
- Report Structuring: ChatGPT helped refine and structure the documentation, ensuring clarity in the description of findings and the methodology used during the penetration test.
- Concept Clarification: ChatGPT also provided clarification on certain security concepts, which were crucial in forming effective attack strategies during the testing process.

While ChatGPT played a supporting role in these areas, the technical exploitation and penetration testing tasks were carried out using tools like Burp Suite, browser Developer Tools, and manual techniques.