Internet Security – COS80013 Lab - 6 Report

Student ID: 104837257 **Student Name:** Arun Ragavendhar Arunachalam Palaniyappan **Lab Name:** COS80013 Lab 6 – Authentication Bypass Techniques
Lab Date: 11 /04/2025 **Tutor:** Yasas Akurudda Liyanage Don

Title and Introduction

This lab was about learning how attackers exploit weak authentication mechanisms on web platforms. Using the HackThisSite.org platform, a range of real-world attack techniques were explored across multiple challenge levels. These included inspecting source code, manipulating hidden fields, injecting malicious inputs, and exploiting server-side logic flaws. The idea was to understand how insecure coding and poor validation can allow unauthorised access, and how developers should guard against such flaws.

Methodology

Basic 1–2: HTML source code was inspected for hidden comments and vulnerable PHP logic. A blank password submission in Level 2 exploited PHP's loose type comparisons.

Basic 3–5: The DOM Inspector tool was used to view and modify hidden input fields, revealing file names and changing email addresses for password retrieval.

Basic 6: Encrypted text was reverse-engineered using pattern-based cipher analysis (Caesar cipher), showing the risks of using weak encryption.

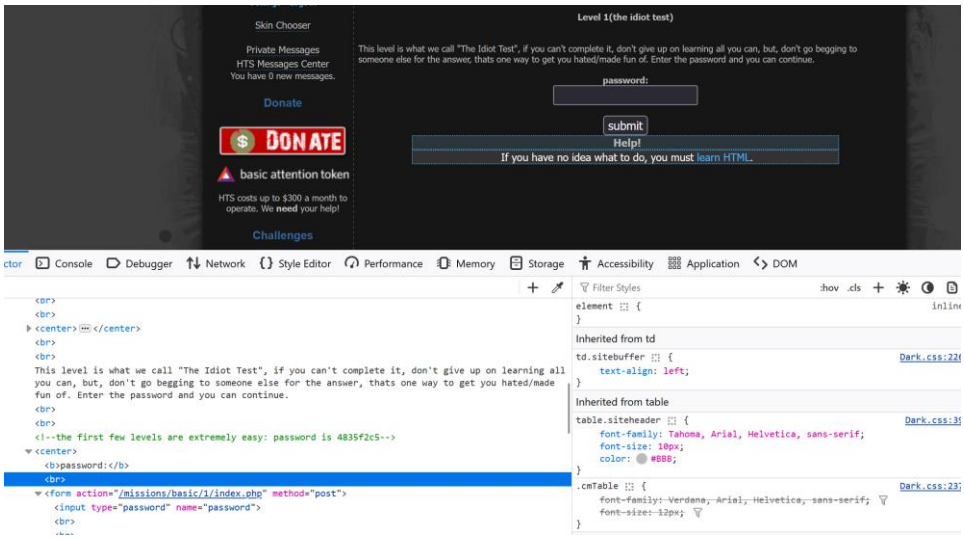
Basic 7–9: Command injection and directory traversal were exploited using crafted inputs like; ls and path traversal (../) to find and open password file.

Basic 10: Browser cookies were edited using Developer Tools.

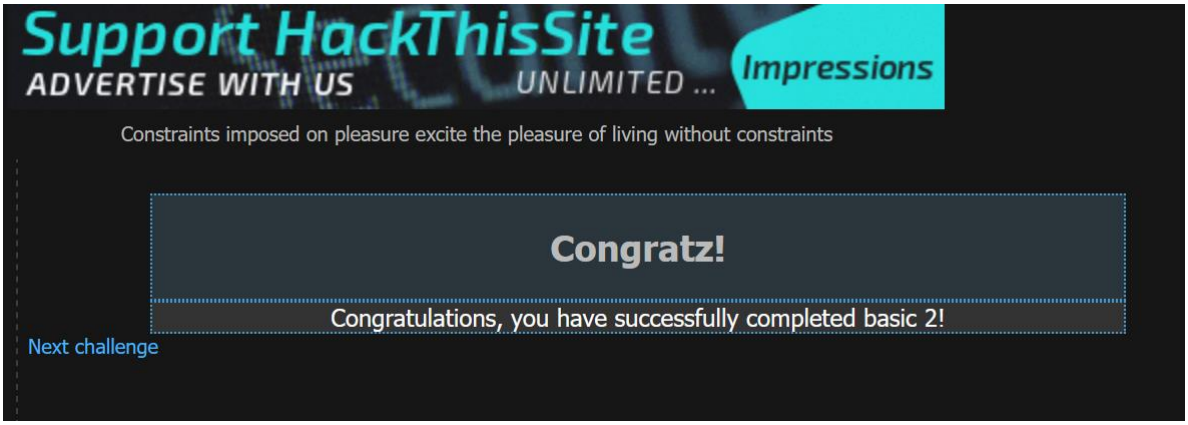
Basic 11: A brute-force approach was used to guess hidden directories and reveal a final clue.

Data Recording and Screenshots

Page source analysis – Found passwords in comments and hidden fields

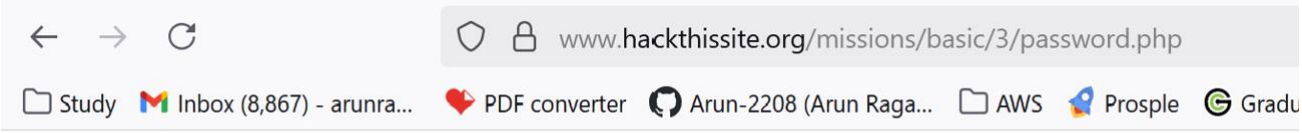


Blank Password accepted due to flaw in PHP code logic.



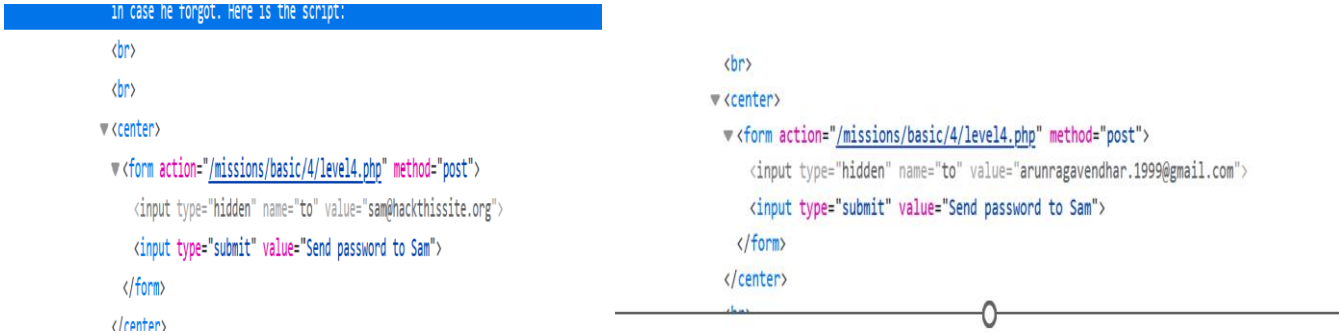
Password file path stored in a variable, found on inspecting the page, making the passwords file accessible.

```
<br>
▼<form action="/missions/basic/3/index.php" method="post">
  <input type="hidden" name="file" value="password.php">
  <input type="password" name="password">
  <br>
  <br>
  <input type="submit" value="submit">
```

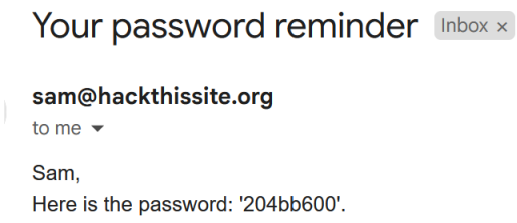


7ca2332d

DOM editing – Modified email fields and hidden inputs to redirect password.

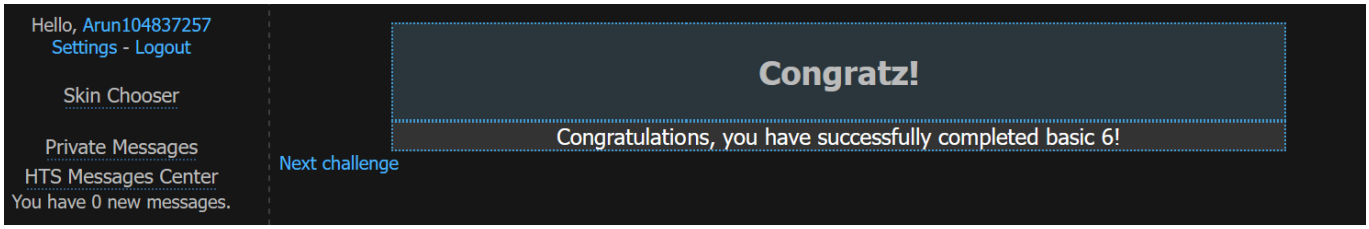


Edited to my own email and password received in my mail



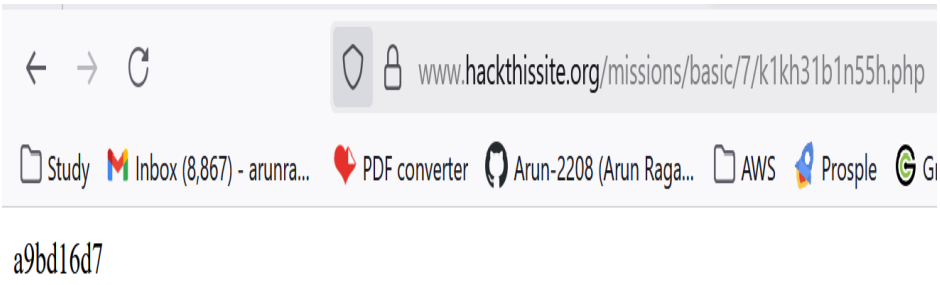
Cipher decoding – aaaa456 → encrypted as abcd8:<, which means it's a **progressive Caesar cipher,** where the shift increases by +1 for every position

The decryption for bff49579 -> bed15012.



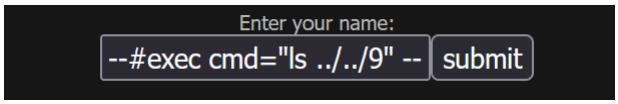
Command injection – Used; ls to list server files

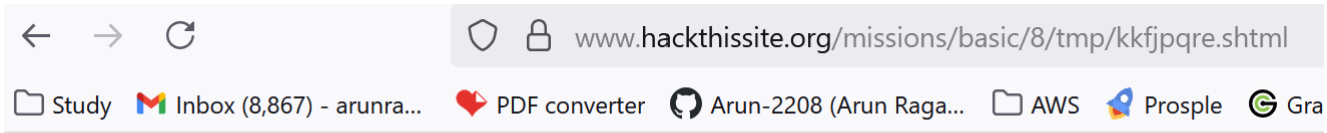
```
      April 2025
Mon Tue Wed Thu Fri Sat Sun
      1   2   3   4   5   6
  7   8   9  10  11  12  13
 14  15  16  17  18  19  20
 21  22  23  24  25  26  27
 28  29  30


index.php
level7.php
cal.pl
.
..
k1kh31b1n55h.php
```



a9bd16d7

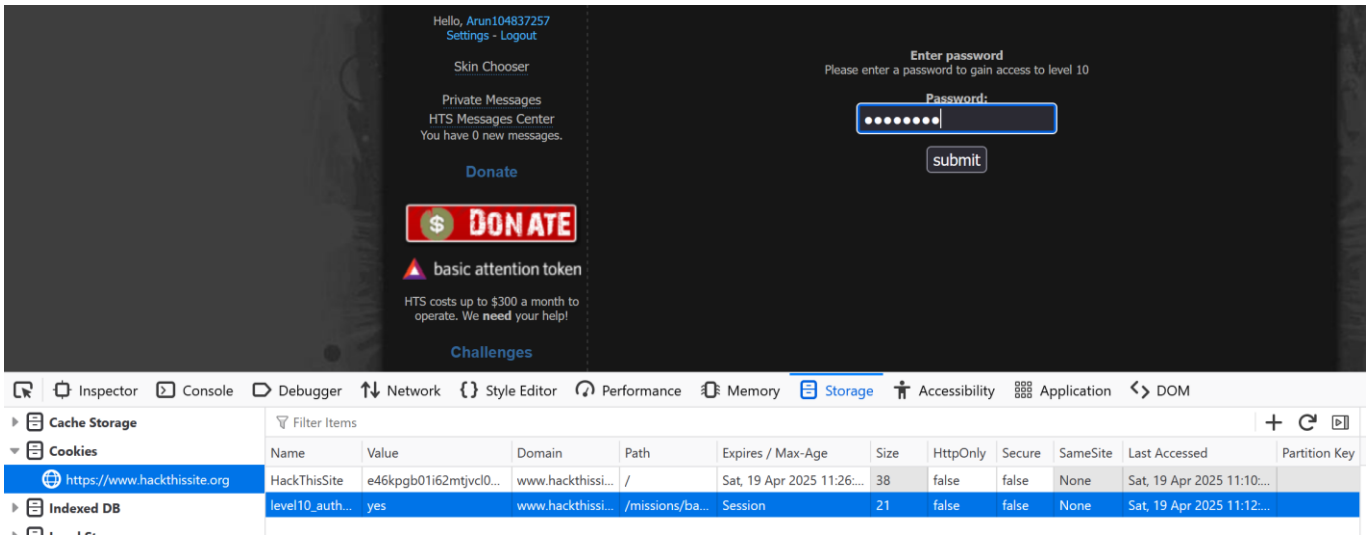
Directory traversal – Navigated between folders to find .php password file





Hi, index.php p91e283zc3.php! Your name contains 24 characters.

Cookie manipulation – Changed level_10_authorized to yes



Discussion and Learnings

Learning 1

Viewing the page source showed that developers had left important information—like passwords—hidden in HTML comments or fields. This shows how attackers can find sensitive data just by right-clicking and choosing "View Source."

Real-World Link: In the real world, this is one of the first things attackers' check. If developers accidentally leave passwords or file paths in the code, they're giving away access for free. Security starts with not exposing anything in the front-end.

Learning 2

Some challenges exposed how PHP scripts can behave strangely when comparing data. For example, a blank password or a missing file can still pass the logic check if the code isn't strict about what it expects.

Real-World Link: PHP is powerful but has its issues. Developers often forget to enforce strict type checks or handle file errors properly. These small mistakes can let attackers bypass checks and gain access.

Learning 3

Modern browsers allow you to open the developer tools and change anything in the page—like hidden fields, cookies, or even script logic—before submitting a form. This made it easy to trick the website into thinking I was authorised.

Real-World Link: If a system only checks things on the user's side (the browser), it can be tricked easily. That's why real security must always be done on the server side, where users can't change anything.

Learning 4

In some levels, typing special input—like a shell command or SSI directive—into a form field caused the website to run system-level commands. This gave access to hidden files and folder structures.

Real-World Link: If web applications allow users to interact directly with system commands without filtering, attackers can take control of the server. These injection attacks are common in real breaches and must be blocked through proper input validation and sandboxing.

Learning 5

One level used a simple form of encryption that could be broken by spotting patterns. By trying out a few inputs, the password could be reverse-engineered using just ASCII values.

Real-World Link: Many older systems or poorly written programs still use weak or custom-made encryption methods. These are not secure. Only trusted, standard encryption methods like AES should be used to protect sensitive information.

Limitations

HackThisSite offers simplified, learning-focused simulations rather than full-scale real systems. While the exercises are useful, real-world websites often use stronger defences like Web Application Firewalls (WAFs). Some methods used here, like SSI, are outdated but still matter when dealing with legacy systems that haven't been properly secured.