Executive Summary

This report documents the methodologies, findings, and recommendations derived from the completion of the HackThisSite Basic Challenges. The challenges are designed to enhance basic web security skills with hands-on experience in finding and exploiting typical vulnerabilities in web applications. The most important takeaways include proper server configuration, robust input validation, secure authentication mechanisms, and effective security testing. It now amalgamates important recommendations for mitigating similar vulnerabilities within real-world applications, so that security in web development and server management is taken foremost.

Introduction

The HackThisSite Basic Challenges provide a hands-on platform for learning and testing the basic skills in web security. With every challenge comes a view into how situations in real life are, thus exposing a participant to a variety of common web application vulnerabilities.f/v This report outlines a structured methodologyM to approach each challenge, the specific vulnerabilitiesf/v that were identified and exploited, and the general lessons learned from it.

Methodology

For each challenge, the following approach was taken:

1. Analyze the challenge description and requirements
2. Inspect the web page and its source code
3. Identify potential vulnerabilities or information leaks
4. Exploit any discovered vulnerabilities
5. Document the process and findings

Findings and vulnerability(f/v)

#basic-1-challenge

Challenge 1: The Idiot Test

Vulnerability: Exposed sensitive information in HTML comments
Severity: High

Description: The password required to pass this level was found in the HTML source code of the page, left as a comment.

Challenge Details:

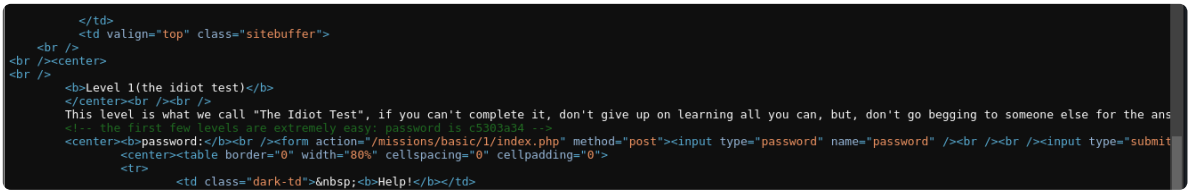
- Title: "Basic 1"
- Description: "This level is what we call 'The Idiot Test', if you can't complete it, don't give up on learning all you can, but, don't go begging to someone else for the answer, that's one way to get you hated/made fun of."
- Requirements: "HTML"

Vulnerability Analysis:

1. The web page contains sensitive information (password) in its HTML source code.
2. The password is easily accessible to anyone who views the page source.
3. This represents a fundamental failure in protecting authentication credentials.

Exploitation:

1. Accessed the challenge page.
2. Viewed the page source code (typically using Ctrl+U or right-click and "View Page Source").
3. Located a comment in the HTML containing the password.



4. Entered the discovered password to complete the challenge.

Impact:
An attacker could easily bypass authentication by viewing the page source, compromising the security of the system. This could lead to unauthorized access to protected areas or information.

Root Cause:

1. Lack of understanding of basic web security principles.
2. Poor development practices, leaving sensitive information in comments.
3. Absence of code review processes to catch such obvious security flaws.

Recommendations:

1. Remove all sensitive information from source code, including comments.
2. Implement proper server-side authentication mechanisms.
3. Store passwords securely using cryptographic hashing algorithms.
4. Conduct regular code reviews with a focus on security.
5. Provide developer training on secure coding practices.
6. Implement a secure development lifecycle (SDL) process.

Lessons Learned:
This challenge demonstrates the importance of:

- Never storing sensitive information in client-side code.
- Understanding that "security through obscurity" is not a valid security measure.
- The need for proper authentication mechanisms.
- The importance of developer education in basic security practices.

Additional Notes:
This vulnerability could be classified as "Exposure of Sensitive Information" (CWE-200) and relates to the broader issue of "Use of Hard-coded Credentials" (CWE-798).

basic-2-challenge

Challenge 2: Incomplete Password Script

Vulnerability: Improper implementation of authentication mechanism
Severity: High
Description: The challenge involves a password protection script that loads the real password from an unencrypted text file. However, the password file was not uploaded, leaving the authentication mechanism vulnerable.

Challenge Details:

- Title: "Basic 2"
- Description: "A slightly more difficult challenge, involving an incomplete password script."
- Requirements: "Common sense"
- Context: Set up by "Network Security Sam"

Vulnerability Analysis:

1. The password is supposed to be loaded from an external file.
2. The password file is not present on the server.
3. The script likely fails to handle the case where the password file is missing.

Exploitation:

1. Attempted to submit an empty password.
2. The system accepted the empty password, granting access.
3. Confirmed successful exploitation with the message: "You have already completed this level!"

Impact:
An attacker can bypass authentication entirely by submitting an empty password, potentially gaining unauthorized access to protected resources.

Root Cause:

1. Failure to implement proper error handling for missing password file.
2. Lack of default denial in case of system errors.
3. Absence of secure coding practices and thorough testing.

Recommendations:

1. Implement proper error handling for all file operations.
2. Use a default "deny all" approach for authentication failures.
3. Store passwords securely using cryptographic hashing, not in plain text files.
4. Implement multi-factor authentication for enhanced security.
5. Conduct thorough security testing, including edge cases like missing files.
6. Use environment variables or secure vaults for storing sensitive information, not files.

Lessons Learned:
This challenge demonstrates the importance of:

- Proper error handling in security-critical code
- Secure storage of authentication credentials
- Thorough testing of authentication mechanisms
- The principle of "fail-secure" in security implementations

Additional Notes:
This vulnerability could be classified as an "Improper Error Handling" issue (CWE-390) combined with "Use of Hard-coded Credentials" (CWE-798).

basic-3-challenge

Challenge 3: Exposed Password File

Vulnerability: Information disclosure through exposed password file path
Severity: Critical
Description: The challenge involves a password protection mechanism where the path to the password file is directly exposed in the HTML source code of the page.

Challenge Details:

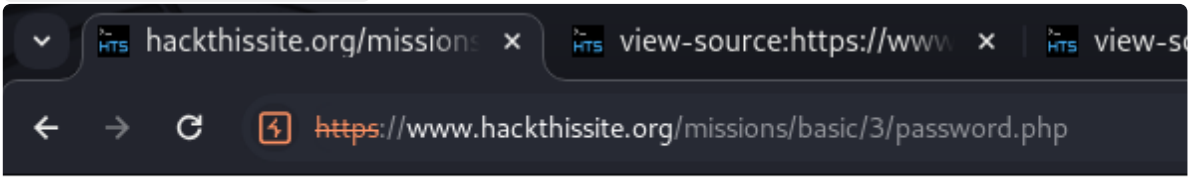
- Title: "Basic 3"
- Description: "This time Network Security Sam remembered to upload the password file, but there were deeper problems than that."
- Requirements: "Some intuition is needed to find the location of the hidden password file. Basic HTML knowledge."

Vulnerability Analysis:

1. The path to the password file ("password.php") is clearly visible in the HTML source code.
2. The form action reveals the challenge structure: "/missions/basic/3/index.php".
3. A hidden input field exposes the filename of the password file.

Exploitation:

1. Inspected the HTML source code of the challenge page.
2. Identified the hidden input field: `<input type="hidden" name="file" value="password.php" />`



1e78cc6d

4. Accessed "password.php" directly through the web browser by navigating to "/missions/basic/3/password.php".
5. Retrieved the password from the contents of "password.php".
6. Used the obtained password to successfully complete the challenge.

Impact:
An attacker can easily obtain the password by directly accessing the exposed password file, completely compromising the authentication mechanism.

Root Cause:

1. Improper security design exposing sensitive file paths in client-side code.
2. Failure to implement server-side security measures to protect sensitive files.
3. Reliance on security through obscurity, assuming users won't inspect the source code.

Recommendations:

1. Remove all references to sensitive files from client-side code.
2. Implement server-side authentication that doesn't rely on accessible password files.
3. Store passwords using secure hashing algorithms in a database, not in plain text files.
4. Use proper access controls to prevent direct access to sensitive files and directories.
5. Implement input validation and sanitization to prevent path traversal attacks.
6. Use environment variables or secure vaults for storing sensitive information.
7. Conduct regular security audits of both client-side and server-side code.

Lessons Learned:
This challenge demonstrates the importance of:

- Keeping sensitive information out of client-side code
- Implementing proper server-side security measures
- Not relying on obscurity for security
- Understanding that all client-side code is accessible to users

Additional Notes:
This vulnerability combines "Information Exposure Through Source Code" (CWE-540) with "Exposure of Sensitive Information to an Unauthorized Actor" (CWE-200).

basic-4-challenge

Challenge 4: Insecure Password Recovery Mechanism

Vulnerability: Exploitable password recovery script with manipulable email recipient
Severity: Critical
Description: The challenge involves a password recovery mechanism where an email script has been set up to send the password to the administrator. The email recipient can be easily manipulated, allowing an attacker to receive the password.

Challenge Details:

- Title: "Basic 4"
- Requirements: HTML knowledge, an email address, and ability to intercept/modify HTTP requests

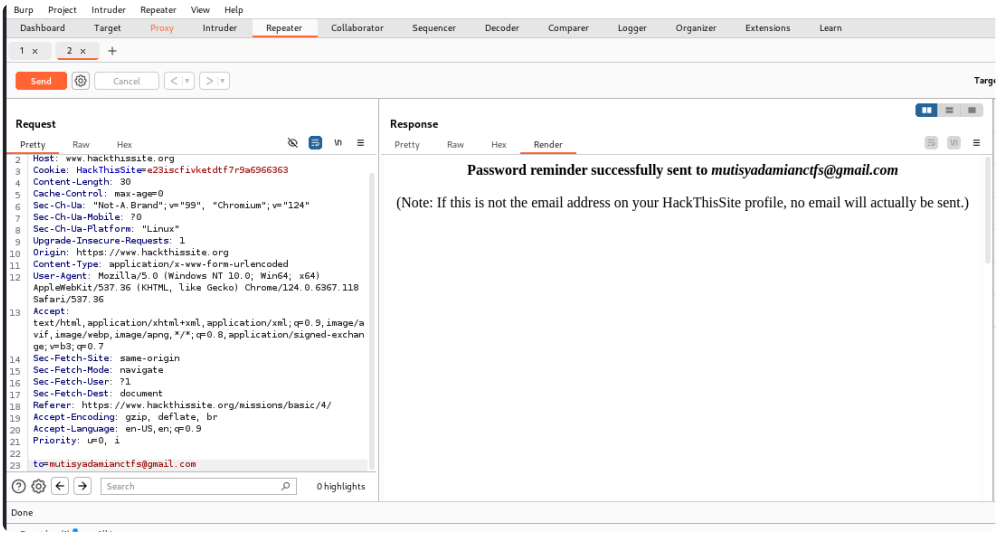
Vulnerability Analysis:

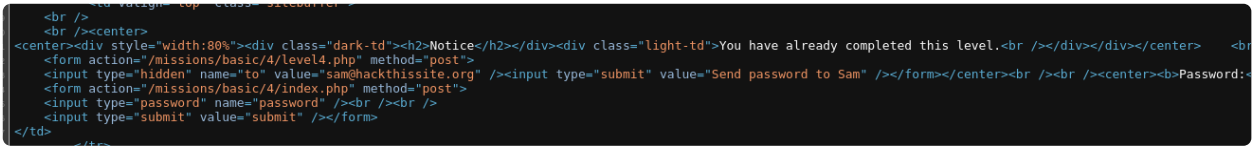
1. The password recovery mechanism emails the password directly.
2. The email recipient address is not properly validated or protected.
3. The form action allows manipulation of the recipient email address.

Exploitation:

1. Examined the HTML source of the challenge page.
2. Identified two forms:
   - One to send the password to "sam@hackthissite.org"
   - Another to submit a password for authentication
3. Used Burp Suite's Repeater tool to intercept and modify the request.
4. Changed the recipient email address from "sam@hackthissite.org" to the attacker's email (partially redacted as "mutisyadamian...").
5. Sent the modified request to trigger the password email.
6. Successfully received the password in the attacker's email inbox.

7. Used the obtained password to complete the challenge.





Impact:
An attacker can easily obtain the administrator's password by exploiting the insecure email recipient in the password recovery mechanism, completely compromising the system's security.

Root Cause:

1. Lack of server-side validation for the email recipient.
2. Directly emailing passwords instead of using a secure reset mechanism.
3. Failure to implement proper authentication before sending sensitive information.

Lessons Learned:
This challenge demonstrates the importance of:

- Thorough input validation, especially for security-critical functions
- Secure design of password recovery mechanisms
- The risks of sending sensitive information via email
- The need for defense in depth in security implementations

Additional Notes:
This vulnerability could be classified as "Improper Input Validation" (CWE-20) combined with "Exposure of Sensitive Information to an Unauthorized Actor" (CWE-200).

basic-5-challenge

Challenge 5: Inadequately Secured Password Recovery Mechanism

Vulnerability: Exploitable password recovery script with insufficient security improvements
Severity: Critical
Description: This challenge is similar to Basic 4, with claimed additional security measures. However, the core vulnerability of manipulable email recipient remains exploitable.

Challenge Details:

- Title: "Basic 5"
- Description: "Similar to the previous challenge, but with some extra security measures in place."
- Requirements: HTML knowledge, JavaScript or Firefox, an email address

Vulnerability Analysis:

1. The challenge claims to have improved security over the previous email-based password recovery.
2. Despite the claimed improvements, the email recipient in the password recovery mechanism remains manipulable.
3. The core vulnerability persists, allowing attackers to redirect the password to their own email.

Exploitation:

1. Examined the challenge page for any visible changes or new security measures.
2. Despite claimed improvements, used the same exploit method as in Basic 4:
    - Intercepted the password recovery request using Burp Suite.
    - Modified the recipient email address to the attacker's email.
    - Sent the modified request to trigger the password email.
3. Successfully received the password in the attacker's email inbox.
4. Used the obtained password to complete the challenge.

Impact:
An attacker can still easily obtain the administrator's password by exploiting the insecure email recipient in the password recovery mechanism, demonstrating that the additional security measures are ineffective.

Root Cause:

1. Inadequate security improvements that fail to address the core vulnerability.
2. Continued use of email for sending sensitive information (passwords).
3. Lack of proper server-side validation for critical operations.

Recommendations:

1. Implement a complete overhaul of the password recovery system:
   - Use time-limited tokens for password resets instead of emailing passwords.
   - Implement strong server-side validation for all inputs.
   - Ensure all sensitive operations require proper authentication.
2. Conduct thorough security audits and penetration testing to identify all potential vulnerabilities.
3. Implement multi-factor authentication to add an extra layer of security.
4. Train developers in secure coding practices and the importance of comprehensive security measures.

Lessons Learned:
This challenge demonstrates:

- The importance of addressing root causes in security vulnerabilities, not just surface-level fixes.
- The need for comprehensive security reviews when implementing improvements.
- The risks of relying on security through obscurity or minor tweaks to insecure systems.

Additional Notes:
This vulnerability, like Basic 4, could be classified as "Improper Input Validation" (CWE-20) and "Exposure of Sensitive Information to an Unauthorized Actor" (CWE-200). The persistence of the vulnerability despite claimed improvements also suggests "Insufficient Security Controls" (CWE-693).

basic-6-challenge

Challenge 6: Weak Encryption System

Vulnerability: Easily reversible custom encryption algorithm
Severity: High
Description: This challenge involves a custom encryption system that is publicly available. The encrypted password is provided, and the task is to decrypt it.

Challenge Details:

- Title: "Basic 6"
- Description: "Network Security Sam has encrypted his password. The encryption system is publicly available and can be accessed with this form."
- Encrypted Password: "6b65j9;k"
- Requirements: Understanding of basic encryption concepts
  Vulnerability Analysis:

1. The encryption system is publicly accessible, allowing analysis of its behavior.
2. The encryption algorithm is a simple substitution cipher with a predictable pattern.
3. The encryption method increases the second character by one each time.

Encryption Algorithm Analysis:

- Pattern: The second character of each input is incremented by one in the ASCII table for each subsequent character.
- Example: If input is "abcde", output might be "acdfg" (b+1=c, c+2=e, d+3=g, e+4=i).
- The first character and all characters after the second remain unchanged.

Decryption Method:

1. Identify the pattern: second character is incremented by an increasing amount.
2. To decrypt: decrement the second character by its position minus one.
3. Leave the first character and all characters after the second unchanged.

Exploitation:

1. Analyzed the given encrypted password: "6b65j9;k"
2. Applied the reverse of the observed encryption pattern:
   - First character '6' remains unchanged
   - Second character 'b' is decremented by 1 (2-1), resulting in 'a'
   - Third character '6' is decremented by 2 (3-1), resulting in '4'
   - Fourth character '5' is decremented by 3 (4-1), resulting in '2'
   - Remaining characters 'j9;k' are unchanged
3. Resulting decrypted password: "6a42j9;k"
4. Entered the decrypted password to complete the challenge

Impact:
While the use of encryption shows an attempt at security, the simplicity and public

availability of the algorithm make it trivial to break, rendering the password protection ineffective.

Root Cause:

1. Use of a weak, custom encryption algorithm.
2. Public availability of the encryption system, allowing easy analysis.
3. Lack of understanding of cryptographic principles.

Lessons Learned:
This challenge demonstrates:

- The dangers of using custom, untested encryption algorithms.
- The importance of understanding cryptographic principles in security design.
- How seemingly complex systems can often be broken with simple analysis.

Additional Notes:
This vulnerability could be classified as "Use of a Broken or Risky Cryptographic Algorithm" (CWE-327) and "Use of a Key Past its Expiration Date" (CWE-324).

basic-7-challenge

Challenge 7: Command Injection Vulnerability

Vulnerability: Unsanitized input leading to command injection
Severity: Critical
Description: This challenge involves a script that displays a calendar based on user input. However, the script is vulnerable to command injection, allowing execution of arbitrary commands on the server.

Challenge Details:

- Title: "Basic 7"
- Description: "The password is hidden in an unknown file, and Sam has set up a script to display a calendar."
- Requirements: Basic UNIX command knowledge

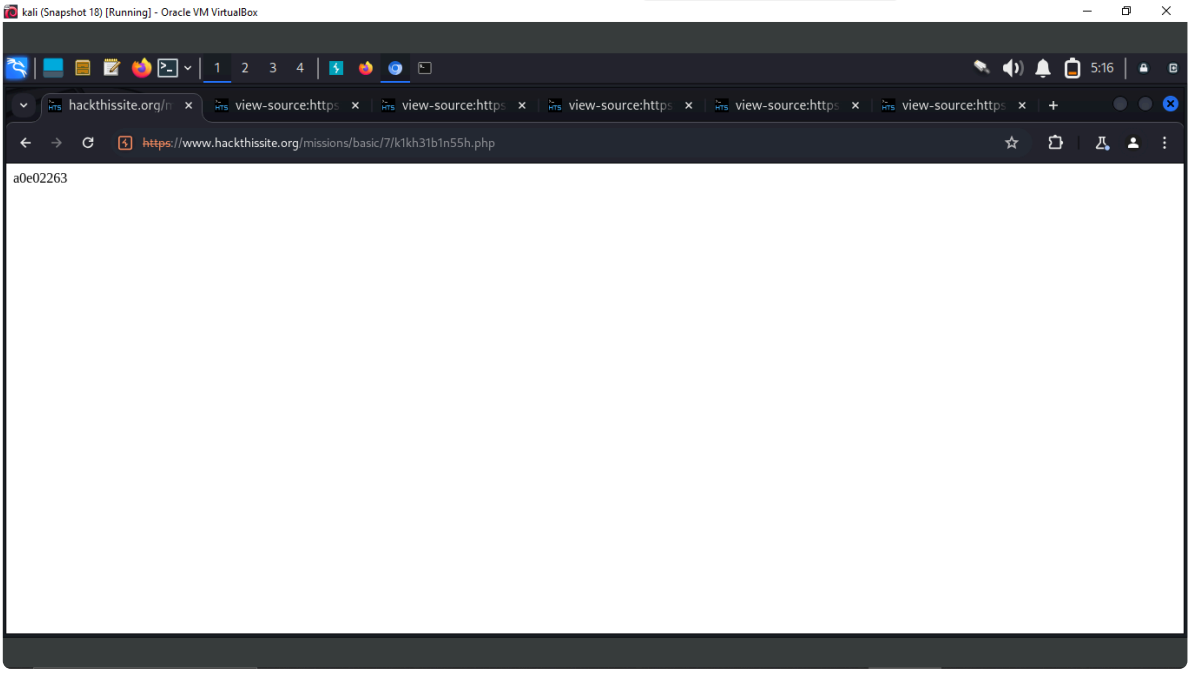
Vulnerability Analysis:

1. The script takes user input to display a calendar for a specific year.

2. User input is not properly sanitized before being used in a system command.
3. This allows for injection of additional commands, which are then executed on the server.

Exploitation:

1. Identified the input field for the year in the calendar script.
2. Injected a command to list all files in the directory:
   - Input: `2023; ls -al`
   - This executes the `cal` command for 2023, then lists all files including hidden ones.
3. Observed the output, which included a file with an unusual name: `k1kh31b1n55h.php`
4. Accessed the file directly through the browser by appending the filename to the URL.
5. Retrieved the password from the contents of `k1kh31b1n55h.php`



```
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  31



      November 2023
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



      December 2023
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  31


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

Impact:
An attacker can execute arbitrary commands on the server, potentially leading to full system compromise, data theft, or service disruption.

Root Cause:

1. Lack of input sanitization in the calendar script.
2. Direct use of user input in system commands without proper validation or escaping.
3. Insufficient understanding of secure coding practices for handling user input.

Recommendations:

1. Implement strict input validation and sanitization for all user inputs.
2. Use parameterized queries or prepared statements when interacting with system commands.
3. Apply the principle of least privilege to the web server and script execution environment.

4. Implement a whitelist of allowed characters and commands for the calendar functionality.
5. Use APIs or libraries for specific functionalities (like displaying calendars) instead of relying on system commands.
6. Regularly conduct security audits and penetration testing to identify similar vulnerabilities.

Lessons Learned:
This challenge demonstrates:

- The critical importance of input validation and sanitization.
- How seemingly innocuous features can lead to severe security vulnerabilities.
- The need for developers to understand and anticipate potential security implications of their code.

Additional Notes:
This vulnerability is classified as "Improper Neutralization of Special Elements used in a Command ('Command Injection')" (CWE-77). It's a common and dangerous web application security flaw that can lead to complete system compromise.

basic-9-challenge

Challenge 9: Directory Traversal and Server-Side Include (SSI) Misconfiguration

Vulnerability: Directory Traversal and SSI Misconfiguration
Severity: High
Description: This challenge involves a misconfiguration in the server setup that allows for unintended access to the password file through clever use of directory traversal and SSI.

Challenge Details:

- Title: "Basic 9"
- Description: "Sam is going down with the ship - he's determined to keep obscuring the password file, no matter how many times people manage to recover it. This time the file is saved in /var/www/hackthissite.org/html/missions/basic/9/. In the last level, however, in my attempt to limit people to using server side includes to display the directory listing to level 8 only, I have mistakenly screwed up somewhere.. there is a way to get the obscured level 9 password."
- Requirements: Knowledge of SSI, Unix directory structure

- Password File Location: /var/www/hackthissite.org/html/missions/basic/9/

Vulnerability Analysis:

1. The server is misconfigured, allowing unintended access to files outside the intended directory.
2. SSI is enabled and not properly restricted to specific directories.
3. The challenge hints at a mistake in limiting SSI to level 8 only, suggesting a vulnerability in the configuration.
4. The script used for validation looks for the first occurrence of '<--', which can be exploited.

Exploitation:

1. Identified the vulnerability in the SSI configuration from the previous level.
2. Used the full path provided (/var/www/hackthissite.org/html/missions/basic/9/) in an SSI directive.
3. Crafted an SSI directive to list the contents of the specified directory.
4. Executed the SSI directive by including it in the URL or input field of Basic 9.
5. Retrieved the password from the revealed file in the directory listing.

Impact:
An attacker can access files and directories outside of the intended scope, potentially exposing sensitive information or system details. This could lead to further exploitation of the system.

Root Cause:

1. Improper configuration of SSI directives and directory restrictions.
2. Insufficient testing of security measures implemented after the previous challenge.
3. Lack of proper input validation and sanitization in the backend script.

Lessons Learned:
This challenge demonstrates:

- The importance of thorough testing when implementing new security measures.
- How misconfigurations can lead to unintended vulnerabilities, even when trying to improve security.
- The potential for chaining vulnerabilities across different levels or components of a system.

Additional Notes:
This vulnerability combines aspects of "Path Traversal" (CWE-22) and "Improper Neutralization of Special Elements used in an SSI Command ('SSI Injection')" (CWE-97). It highlights the importance of holistic security approaches that consider the interactions between different system components and security measures.

basic-10-challenge

Challenge 10: Client-Side Authentication Bypass

Vulnerability: Insecure Client-Side Authentication
Severity: High
Description: This challenge involves a flawed authentication mechanism that relies on client-side JavaScript and cookie manipulation for user validation, allowing easy bypass of the authentication process.

Challenge Details:

- Title: "Basic 10"
- Description: "This time Sam used a more temporary and 'hidden' approach to authenticating users, but he didn't think about whether or not those users knew their way around javascript..."
- Requirements: Knowledge of JavaScript, understanding of cookies and client-side authentication
- Authentication Mechanism: Client-side JavaScript and cookie-based

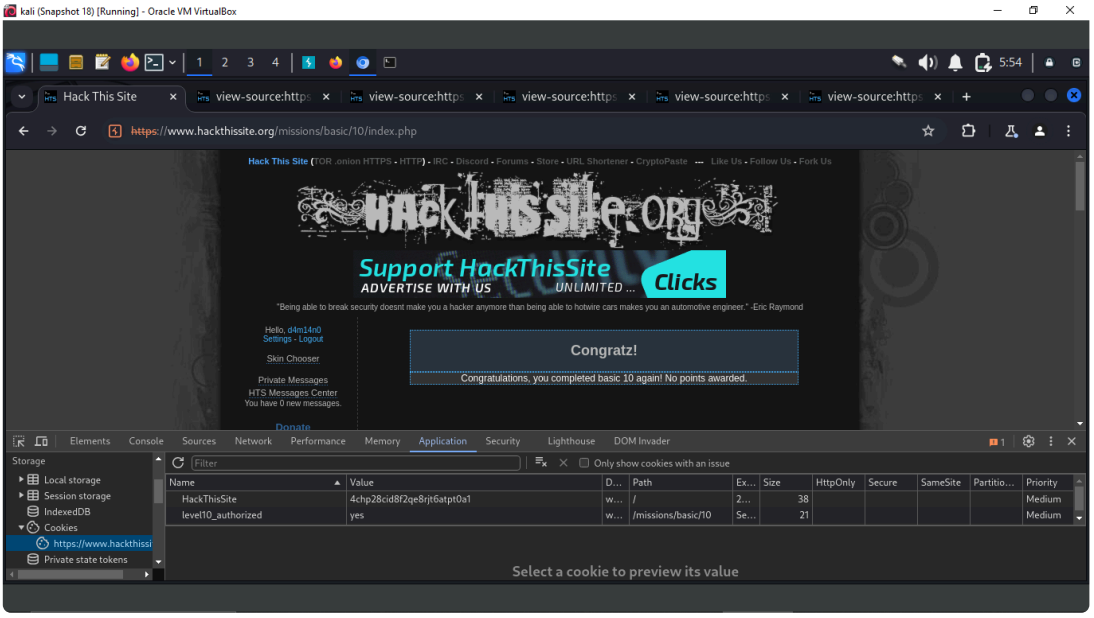
Vulnerability Analysis:

1. The authentication process relies on client-side JavaScript for user validation.
2. A cookie is used to store the authentication state.
3. The server trusts the client-side authentication without proper server-side verification.
4. The authentication state can be easily manipulated by modifying the cookie value.

Exploitation:

1. Identified that the authentication state is stored in a cookie.
2. Located the relevant cookie in the browser's developer tools.
3. Modified the cookie value from "yes" to "no" to bypass the authentication check.

4. Successfully submitted the form without providing a password.



Impact:
An attacker can easily bypass the authentication mechanism, gaining unauthorized access to protected areas or functionality of the website. This could lead to data breaches, unauthorized actions, or further exploitation of the system.

Root Cause:

1. Reliance on client-side authentication without server-side validation.
2. Improper implementation of security measures, trusting client-controlled data.
3. Lack of understanding of the limitations and vulnerabilities of client-side security mechanisms.

Recommendations:

1. Implement proper server-side authentication mechanisms.
2. Never rely solely on client-side validation for security-critical operations.
3. Use secure, HTTP-only, and SameSite cookies for session management.
4. Implement strong encryption and signing of cookies to prevent tampering.
5. Regularly audit and test authentication mechanisms for vulnerabilities.
6. Educate developers on secure coding practices and the limitations of client-side security.
7. Implement multi-factor authentication for additional security.
8. Use security headers like Content Security Policy (CSP) to mitigate potential XSS attacks that could manipulate cookies.

Lessons Learned:
This challenge demonstrates:

- The dangers of relying on client-side security mechanisms.
- How easily authentication can be bypassed when trusting client-controlled data.
- The importance of implementing proper server-side validation and authentication.
- The need for a thorough understanding of web security principles among developers.

Additional Notes:
This vulnerability falls under the category of "Improper Authentication" (CWE-287) and "Reliance on Untrusted Inputs in a Security Decision" (CWE-807). It's a common mistake in web application development, especially among less experienced developers or in rapidly developed prototypes.
basic-11-challenge

Basic 11 Challenge Report: Apache Music Site

Challenge Overview

Sam has decided to create a music site but struggles with understanding Apache. The challenge involves manipulating URL and directory structures to uncover hidden content.

Challenge Details

Title: Basic 11: Music Site Challenge
Objective: Uncover the hidden password to progress to the next stage.
Displayed Information:

```
I love my music! "First Episode at Hienton" is the best! The music seems to
change anytime I refresh.
```

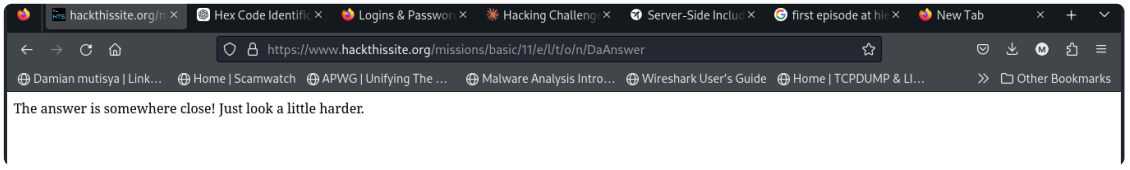
The site content suggests a focus on Elton John's music. Initial attempts to use the full name in the URL yielded no results. However, breaking down the name "elton" into individual characters led to accessing a parent directory.

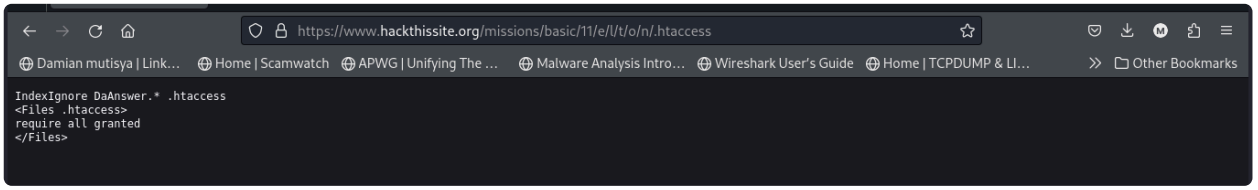
Steps Taken to Solve the Challenge

1. **Initial Exploration**:

- Observed the music details changing upon refreshing the site.
- Attempted to add the full name "Elton John" in the URL without success.

2. **Directory Traversal**:
   - Separated the letters in "elton" (e.g., "e/l/t/o/n") and accessed the parent directory.
3. **Using .htaccess**:
   - Referred to an Apache tutorial on `.htaccess` to understand access control.
   - Accessed a file named "DA answer".
   - Replaced the `.htaccess` content with the path "daanswer" to reveal the password "somewhere close".
4. **Using Gobuster**:
   - Used Gobuster, a directory scanning tool, to scan for available directories.
   - Discovered `index.php` which contained a password placeholder.
   - Entered the revealed password to progress to the next stage.





Analysis

Root Cause:

- Lack of secure directory listing and access control.
- Insufficient sanitization and protection of sensitive files.
- Potential misconfiguration of Apache server settings.

Impact

- Exposed directory structures and sensitive files.
- Unauthorized access to protected content.
- Compromised the integrity of the web application.

basic-8-challenge

Challenge 8: Server-Side Include (SSI) Injection Vulnerability

Vulnerability: Unsanitized input leading to SSI injection
Severity: Critical
Description: This challenge involves a PHP script written by Sam's daughter that takes user input. The script is vulnerable to SSI injection, allowing execution of server-side commands and directory traversal.

Challenge Details:

- Title: "Basic 8"
- Description: "The password is yet again hidden in an unknown file. Sam's daughter has begun learning PHP, and has a small script to demonstrate her knowledge."
- Requirements: Knowledge of SSI (Server-Side Includes)
- Password File Location: /var/www/hackthissite.org/html/missions/basic/8/

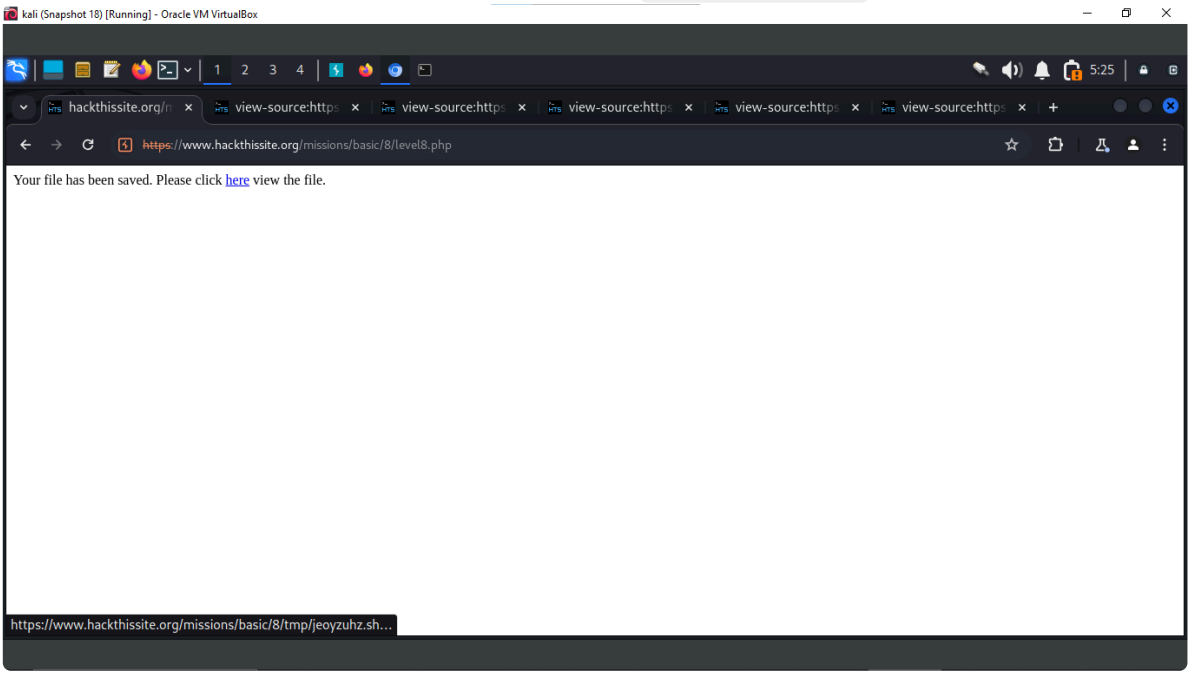
Vulnerability Analysis:

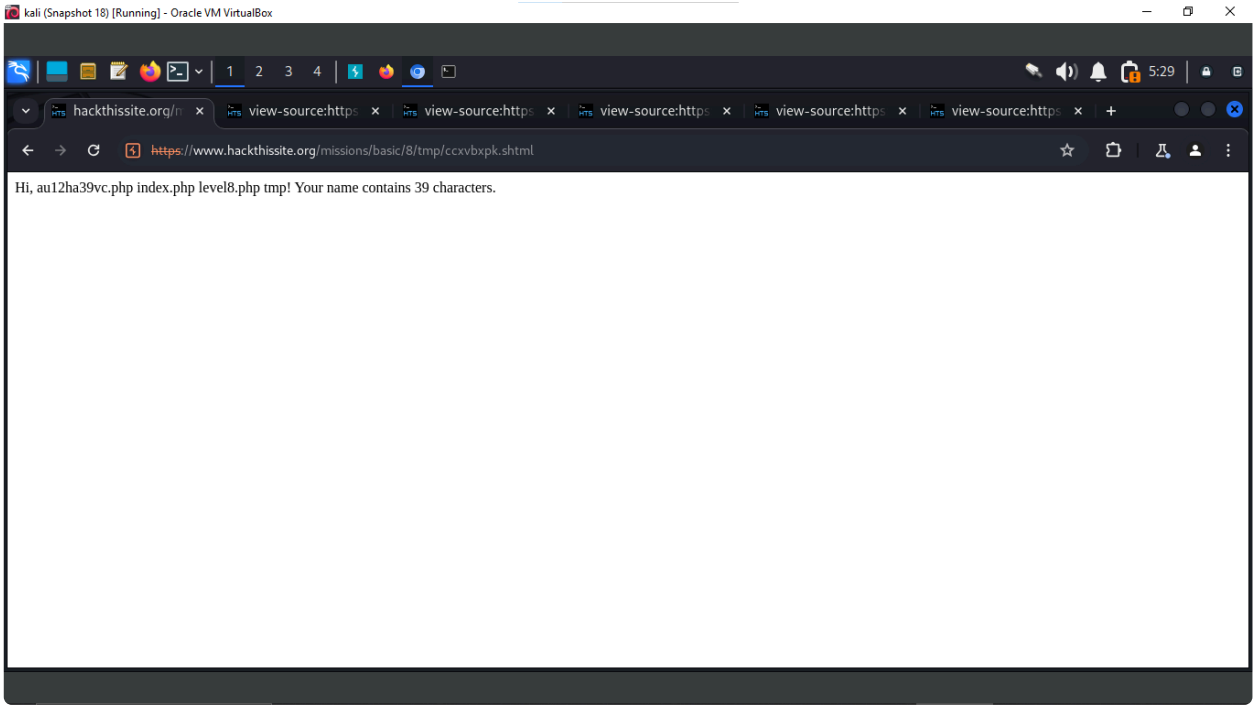
1. The PHP script takes user input, likely for a name or similar field.
2. User input is not properly sanitized before being used in the page.
3. The server has SSI enabled, allowing for injection of SSI directives.
4. This allows for execution of server-side commands and directory traversal.

Exploitation:

1. Identified the input field in the PHP script.
2. Injected an SSI directive to list directory contents:
   - Input: `<!--#exec cmd="ls ../" -->`
   - This executes the `ls` command on the parent directory.
3. Observed the output, which included a file named `tmp/jeoyzuhz.shtml`.
4. In the listing of the parent directory, identified a file named `au12ha39vc.php`.
5. Accessed `au12ha39vc.php` directly through the browser by appending it to the URL.

6. Retrieved the password from the contents of `au12ha39vc.php`.





Impact:
An attacker can execute arbitrary server-side commands, potentially leading to full system compromise, data theft, or service disruption. The ability to traverse directories also exposes sensitive file structures.

Root Cause:

1. Lack of input sanitization in the PHP script.
2. Server configuration allowing SSI execution without proper security measures.
3. Insufficient understanding of secure coding practices and the risks of SSI.

Lessons Learned:
This challenge demonstrates:

- The risks of allowing inexperienced developers to create production code without oversight.
- The importance of server configuration in web application security.
- How legacy features like SSI can introduce significant vulnerabilities if not properly managed.

Additional Notes:
This vulnerability is classified as "Improper Neutralization of Special Elements used in an SSI Command ('SSI Injection')" (CWE-97). It's a less common but potentially very dangerous web application security flaw that can lead to remote code execution.

Recommendations(R)

Consolidated Recommendations

1. **Sensitive Information Handling**:
   - Remove all sensitive information from source code comments.
   - Never store passwords or other sensitive data in plain text files.
   - Avoid hardcoding sensitive information in any part of the system.
   - Conduct regular code reviews to ensure sensitive information is not exposed in client-side code.
2. **Authentication and Authorization**:
   - Implement proper authentication mechanisms.
   - Consider implementing multi-factor authentication for enhanced security.
   - Always authenticate users before performing sensitive operations like password resets.
   - Implement strict access controls for all sensitive files and resources.
   - Use secure, HTTP-only, and SameSite cookies for session management.
3. **Password Security**:
   - Store passwords securely using cryptographic hashing algorithms.
   - Implement secure password reset mechanisms that don't involve sending passwords via email.
4. **Input Validation and Error Handling**:
   - Implement rigorous input validation and sanitization for all user-supplied data.
   - Implement robust error handling in all security-critical code.
   - Implement rigorous server-side validation for all user inputs, especially in security-critical functions.

5. **Encryption and Key Management**:
    - Avoid using custom encryption algorithms; always opt for well-established, peer-reviewed cryptographic standards.
    - Implement proper key management and use strong, random encryption keys for any encryption processes.
6. **Security Testing and Training**:
    - Utilize intercepting proxies (like Burp Suite) in security testing to identify and exploit potential vulnerabilities.
    - Implement comprehensive security testing that includes verifying the effectiveness of security improvements.
    - Conduct regular security training for developers, focusing on common vulnerabilities like command injection and how to prevent them.
    - Ensure that security improvements address the root cause of vulnerabilities, not just superficial aspects.
7. **Server and Configuration Management**:
    - Conduct a thorough review of server configurations, disabling unnecessary features like SSI unless absolutely required.
    - Properly configure `.htaccess` and implement robust input validation and sanitization.
    - Ensure Apache server settings are secure and follow best practices for web server configuration.
    - Regularly audit server configurations and security measures, particularly after changes.
8. **Development Practices**:
    - Implement a secure development lifecycle that includes code reviews, especially for code written by junior or learning developers.
    - Always use a "fail-secure" approach in authentication mechanisms.

Conclusions(C)

1. The "Basic" web challenges, starting with "The Idiot Test", demonstrate common but critical web application vulnerabilities. While some may seem trivial, they represent real-world issues that can lead to severe security breaches if left unaddressed. It's crucial for developers and security teams to be aware of these basic vulnerabilities and take steps to prevent them.
2. The "Basic 2" challenge highlights how even simple oversights in implementation can lead to complete authentication bypass. It emphasizes the need for thorough security testing and the importance of proper error handling in security-critical code.
3. The "Basic 3" challenge further emphasizes the importance of proper file security and access control. It demonstrates how seemingly minor oversights, such as

leaving sensitive files accessible, can completely undermine an authentication system.

4. The "Basic 4" challenge underscores the critical importance of thorough input validation and the potential security risks of seemingly convenient features like emailing passwords. It demonstrates how simple oversights in implementation can lead to complete compromise of authentication mechanisms.
5. The "Basic 5" challenge highlights a common pitfall in security: the assumption that minor changes or obscurity can fix fundamental flaws. It emphasizes the need for thorough, root-cause analysis and comprehensive security redesigns when addressing vulnerabilities.
6. The "Basic 6" challenge illustrates the risks of relying on custom cryptographic solutions. It emphasizes that security through obscurity is not a valid approach, and that even seemingly complex systems can be vulnerable to simple analysis techniques.
7. The "Basic 7" challenge highlights the severe risks associated with improper handling of user input. It demonstrates how a simple calendar display function can be exploited to gain unauthorized access to the server's file system, emphasizing the need for thorough security considerations in all aspects of web application development.
8. The "Basic 8" challenge underscores the importance of secure server configurations and the risks associated with legacy features like SSI. It also highlights the need for proper oversight and education when allowing inexperienced developers to contribute to production environments.
9. Basic 9 challenge highlights the necessity of thorough security testing, understanding server configurations, considering system interactions, and implementing defense-in-depth strategies.
10. Basic 10 challenge underscores the importance of robust server-side validation, understanding authentication and session management security, and avoiding reliance on client-side security measures.
11. Basic 11 challenge highlights the importance of proper server configuration and secure coding practices to prevent unauthorized access and directory traversal vulnerabilities.

Appendices

- How to View the HTML Source Code of a Web Page (computerhope.com) - Hint for Level 1, 2 & 3
- How to view source code – ViewSourcePage.com - Hint for Level 1, 2 & 3
- How to Edit Any Web Page in Chrome (or Any Browser) (howtogeek.com) - Hint for Level 4 & 5

- ASCII Table - ASCII Character Codes, HTML, Octal, Hex, Decimal - Hint for Level 6.
- Linux and Unix cal command tutorial with examples | George Ornbo (shapeshed.com) - Hint for Level 7
- cal command in Linux with Examples - GeeksforGeeks - Hint for Level 7
- Linux Commands Cheat Sheet | Red Hat Developer - Hint for Level 7
- Server-Side Includes (SSI) Injection | OWASP Foundation - Hint for Level 8 & 9
- View, edit, and delete cookies - Chrome Developers - Hint for Level 10
- Apache HTTP Server Tutorial: .htaccess files - Apache HTTP Server Version 2.4 - Hint for Level 11