HackThisSite Basic Challenges

Challenge 1: HTML
——————————————————————————

Upon inspecting the source code, you will find the comment:
<!-- the first few levels are extremely easy: password is 74e20593 -->

Challenge 1 complete.

Challenge 2: Common Sense
——————————————————————————-

As they have failed to remember to upload the password script, the password is just pressing enter.

Challenge 2 complete.

Challenge 3: HTML
——————————————————————————-

When inspecting the source code, the password file was able to be accessed in
<input type="hidden" name="file" value="password.php" />
by appending password.php on the website, the password was revealed.

40ca7c92

Challenge 3 complete.

Challenge 4: HTML, email
——————————————————————————-

After inspecting the source code, you can change the password reminder email to your own email, thus receiving:

Sam,
Here is the password: '7e65cd72'.

Challenge 4 complete.

Challenge 5: HTML, JS, email
——————————————————————————-

Done similarly to challenge 4, with the exception of the inclusion of js. Upon changing the email reminder again, this is received:

Sam,
Here is the password: '0a9c3d9e'.

Challenge 5 complete.

Challenge 6: Cryptography
——————————————————————

The encrypted password is **573gegi;** and upon closer inspection through trial and error,

hello becomes hfnos
welcome becomes wfnfsrk
test123 becomes tfuw579
wow!123 becomes wpy$579

character 1: no change, character 2: 1 shift, character 3: 2 shift, character 4: 3 shift and so on, incrementing by 1 with each character increase. Doing this in ASCII values.

applying the same patterns in reverse, we get: 561dabc4

Challenge 6 complete.

Challenge 7: UNIX
————————————————————————

Using ;, you can use linux command injections.
For example, 2024;ls shows ls below the years, as the command entered in unix would be cal 2024;ls which shows an unusual file k1kh31b1n55h.php

using 2024;cat k1kh31b1n55h.php does not work

however, appending it to the website revealed: d88c8794

Challenge 7 complete.

Challenge 8: SSI
——————————————————————————

Upon trying to list the files of directories,
tshngmww.shtml hipykpqu.shtml ztxdhjxn.shtml avpfeoie.shtml fviqpmaw.shtml kqbybdzc.shtml dzrnmzgx.shtml npcsygfl.shtml whqxxojt.shtml ylomcmvu.shtml uhdppswp.shtml gzntiicx.shtml dzwbqiuu.shtml qvzuieng.shtml smcerykh.shtml qjhnmhmq.shtml znodwztr.shtml

was the output.

Upon using <!--#exec cmd="ls ../" —>, we then get
Hi, au12ha39vc.php index.php level8.php tmp! Your name contains 39 characters.

When appended on the site, it then reveals c046f860

Challenge 8 complete.

Challenge 9: SSI , UNIX
——————————————————————

This time we go back to challenge 8 to submit a new SSI specific for level 9. Upon editing the payload to <!--#exec cmd="ls ../../9/" --> we get the password in the same manner as challenge 8.

Challenge 9 complete.

Challenge 10: Javascript
———————————————————————

Using the hint of the cookies instructions in the task resources, but cookies would only show learn more. Upon changing settings to allow third-party cookies, I found the permissions of level 10 to be no and changed it to yes, solving it.

Challenge 10 complete.

Challenge 11: Apache
————————————————————————

After inspecting the source code, only the comment about a collection was found. Upon appending index.php, the password entry was found. Now to find the password.

Upon trying to append letters, /e revealed /e/l/t/o/n, which we then added .htaccess to, revealing
IndexIgnore DaAnswer.* .htaccess
<Files .htaccess>
require all granted
</Files>

After appending DaAnswer, it states The answer is simple! Just look a little harder.

Hilariously, copy pasting ‘simple’ into the index.php seems to have worked. After a lot of inspecting.

Challenge 11 complete.

Advice to keep web applications more secure
——————————————

1. HTML Security

- **Sanitize user inputs**: Use tools or libraries to sanitize any user input, preventing cross-site scripting (XSS) attacks. For example, filter out <script> tags, attributes like onerror, and encode user-generated content safely.
- **Content Security Policy (CSP)**: Implement CSP headers to restrict what resources can be loaded by the browser, blocking inline JavaScript and limiting the sources of external scripts.
- **Prevent Cross-Site Request Forgery (CSRF)**: Use anti-CSRF tokens to prevent attackers from forging requests on behalf of an authenticated user.

2. Server Side Includes (SSI) Vulnerabilities

- **Disable SSI if not needed**: Disable SSI processing in the Apache server configuration (Options -Includes) if it’s not being used.
- **Limit SSI execution**: If SSI must be used, ensure that only trusted users can access pages that allow SSI commands, and only allow safe directives.
- **Sanitize Inputs**: If you must accept user-generated content via SSI, ensure inputs are sanitized and validate the input carefully to avoid execution of malicious code.

3. Apache Security

- **Keep Apache updated**: Regularly update Apache to ensure you have the latest security patches.
- **Limit the scope of accessible files**: Use proper file permissions and AllowOverride directives to restrict access to sensitive files (e.g., .htaccess, .htpasswd, etc.).
- **Disable directory listing**: Ensure Options -Indexes is configured so that directory listings aren’t visible to users.
- **Limit HTTP methods**: Only allow necessary HTTP methods (e.g., GET, POST, etc.) using the Limit or LimitExcept directive.
- **Use mod_security**: This Apache module can provide additional web application firewall protection.

4. Unix/Linux Security

- **Secure file permissions**: Restrict file permissions to the minimum necessary for each user. Use chmod to ensure sensitive files are not world-readable or executable.

- **Disable unnecessary services**: Disable or remove any unnecessary services and applications to reduce the attack surface.
- **Use SELinux or AppArmor**: These security frameworks can provide an extra layer of security to enforce access controls.
- **Log access attempts**: Monitor access logs to identify suspicious activity. Regularly check Apache or system logs for abnormal access patterns.
- **SSH Hardening**:
  - Use key-based authentication instead of passwords.
  - Disable root login and set strong passwords for non-root users.
  - Limit SSH access to specific IP addresses.
- **Regular system updates**: Regularly update your system and install security patches to protect against known vulnerabilities.

5. Web Application General Security

- **Input validation**: Validate all incoming data (especially from untrusted sources) to ensure it's in the expected format, length, and type. Use allowlists where possible.
- **SQL Injection Prevention**: Use prepared statements or ORM libraries to prevent SQL injection attacks.
- **Encrypt sensitive data**: Use TLS/SSL encryption for all data transmitted between the server and client. Additionally, consider encrypting sensitive data stored on the server.
- **Authentication and Authorization**:
  - Use strong authentication methods (e.g., multi-factor authentication).
  - Ensure proper role-based access control (RBAC) for users to limit their privileges to only what’s necessary.
- **Session Management**: Secure session cookies with flags such as HttpOnly, Secure, and SameSite to protect against session hijacking.
- **Error Handling**: Avoid detailed error messages that can leak sensitive server or application information. Provide generic error messages and log detailed errors server-side.
- **Backup and Recovery**: Regularly backup sensitive data and ensure you have a disaster recovery plan.

6. Automated Security Tools

- **Use static and dynamic analysis tools**: Automated security testing tools (e.g., **OWASP ZAP**, **Burp Suite**, **SonarQube**) can identify common security issues in your code or application.
- **Regular penetration testing**: Perform regular penetration testing to proactively find vulnerabilities and patch them.
- **Web Application Firewalls (WAF)**: Deploy a WAF to protect your application from common threats, including XSS, SQL injection, and other OWASP Top 10 vulnerabilities.

7. Monitor and Respond to Incidents

- **Intrusion detection and prevention systems (IDPS)**: Set up systems to detect and block any suspicious activity in real-time.
- **Logging and auditing**: Ensure that all security-related events are logged and reviewed regularly. Use tools like **ELK Stack** (Elasticsearch, Logstash, Kibana) for log aggregation and analysis.

By following these practices, you'll improve your web application's resilience against common vulnerabilities and minimize the risk of exploits.