We Test Pens Incorporated

COMP90074 - Web Security Assignment 2

PENETRATION TEST REPORT FOR PleaseHold - WEB APPLICATION

Report delivered: 15/05/2023

Executive Summary

This report contains the required test scenarios for the HR web application provided to PleaseHold Pty. Ltd (this report will also refer PleaseHold Pty. Ltd as “the company” and “the organisation”). Since the product has been released for production and due to a lacking testing environment, all tests have been performed carefully to not cause any unintentional Denial of Service for the server’s side.

- We Test Pens Incorporated has been tasked to perform penetration tests and to provide feedback and remediation.
- Three python files have been provided to reproduce the outcomes and will be guided through and discussed in the relevant sections of each found vulnerability under ‘proof of concept’.
- The four flags found within the company’s website are listed in the appendix section under ‘additional information’ and are linked in each section.

In conclusion, considering the lack of budget, time, and acknowledgment of the application under production, risks assessment has been performed for the company’s administrative personnel.

Overall risk assessment

- Overall risk assessment rate is considered **High** as extremely dangerous vulnerabilities, as well a minor vulnerability, have been found in the PleaseHold Pty. Ltd website. (Note: all risk assessments are based on Appendix 1 – Risk Matrix provided in the end of this report and is based on ISO31000)
- The fastest option in remediating the problematic vulnerabilities is the instalment of Web Application Firewalls (WAF). This is due to its protective abilities in website attacks. Although this would require the company to conduct a budget review.
- Depending on *risk appetite* of the company they may consider some of the following risks to be eliminated as a priority and may even perform no action for the lower risk vulnerability. This is based on information on a data leakage due to the current vulnerabilities and how this information can interfere with the company’s mission and values.

Table of Contents

Summary of Findings

A brief summary of all findings appears in the table below, sorted by Risk rating.

Risk	Reference	Vulnerability
Extreme	below	Blind SQL injection in Find User functionality
Extreme	below	This Vulnerability found in website validation of profile
High	below	This Vulnerability found in Question page
Low	below	This vulnerability found in API calls responds

Detailed Findings

This section provides detailed descriptions of all the vulnerabilities identified.

SQL Injection – Blind Injection resulted in data breach

Description	This vulnerability allows attackers to execute, manipulate, abuse database commands, and produce unexpected outcomes. In this test case, information about tables and underlying data could be attained (fetched) using blind SQLi injection. Methodology: HTTP response evaluated from **find-user.php?username=" USERNAME"** in **"find user"** page (find.php) using substring functionality of SQLi to brute force data stored in database. Initially, a single quote in the end of a username arose a 500 Internal error. Using logical conditions creates a right command resulting in the following payload: /find-user.php?username=anything'or+1+limit+1--+- Using this logic with the substring functionality resulted in finding table names, column names and their underlying data in database (information_schema table). **Risk statement:** The revealing of company and user's data may occur due to executing SQL commands which leads to data breaches and the disclosure of database and users' personal information to outside sources.
Proof of Concept	A proof of concept is attached in the appendix section of this report **(** **Figure 1 FLAG_SQL** Injection**)**. To reproduce this an exploit file named sql.py is provided as an attachment. **Step 1:** Run sql.py application. **Step 2:** Select 1 to receive a list of database table names. Select 2 to receive a list of a specific table's columns name. Select 3 to get data within a specific table and column. **Step 3:** you collect all rows and data for each column in any tablet of the database. Specifically, a flag was found in "Password" column of "Users" table.
Impact	**Potential impact on the organisation may be accessing the website database by the attacker to reveal personal information about Users such as their username, password, roles, etc.**

Risk Ratings	**Extreme** Data breach of all tables and their contents especially “Users” table is determined as “likely” to happen in which its consequences would be “Major” [3]

Recommendation	**Using prepared statements:**
	$stmt = $dbConnection->prepare('SELECT * FROM Users WHERE password = ?');
	$stmt->bind_param('s', $pass); // 's' specifies the variable type => 'string'
	$stmt->execute();
	
	$result = $stmt->get_result();
	
	Set char set and close errors (for instance in SQLi)
	mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT); // error reporting
	$dbConnection = new mysqli('127.0.0.1', 'username', 'password', 'test');
	$dbConnection->set_charset('utf8mb4'); // charset
	
	Blacklist and Whitelisting values
	Example of Whitelisting:
	$dir can only be 'DESC', otherwise it will be 'ASC'.
	if (empty($dir) || $dir !== 'DESC') {
	  $dir = 'ASC';
	}
	
	Further recommendations
	Storing sensitive data in hash format such as users’ passwords.
	Web Application Firewall (WAF)

References	[1] https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php [2] https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html [3] https://peoplesafe.co.uk/blogs/risk-assessment-guidelines/

SSRF - Server-side request forgery to gain access on Unauthorised Services

Description	SSRF is used against the server itself to access an unintended service (location). This vulnerability is found in profile page on validating (validate.php) websites. After sending GET requests to internal addresses like localhost and observing responses using a port scanner, specifically built for this vulnerability, a connection was established for the provided address on port 8873 (http://localhost:8873/) whilst localhost can be an IP address like 127.0.0.1. As a result of the content of "flag.txt" this was revealed in the following path: Documents/background-checks/sensitive/flag.txt **Risk Statement:** Accessing this path on open port after a simple port scanning may occur due to not blocking internal access to this port address which leads to data leakage of "background-checks".
Proof of Concept	A proof of concept is attached in appendix section of this report **(Figure 2 FLAG_SSRF)** The following steps will result in exploiting this vulnerability: **Step 1:** Run ssrf.py. **Step 2:** Enter username and password to login and get required cookies. **Step 3:** The algorithm will scan ports between 1 to 65535 to find an anomaly based on content length or status of the response. **Step 4:** Outcome with the open port will be printed in outputs.
Impact	**The impact of this attack is gaining access to the contents and documents under this vulnerable application (service). The impact could be dependent on the sensitivity of data and on the organisation's priorities and criteria.**
Risk Ratings	**Extreme** Since this vulnerability is found under the path named "background-check/sensitive" the assumption is that data is sensitive, and the impact would be Major and likely to occur.

Recommendation	- Blacklisting/Whitelisting inputs for the problematic addresses and their alternative representations. For example: 127.0.0.1 and 2130706433 or 127.1 .[4] IP address validation (Regex suggested) [5] - Instead of accepting complete URL from user, allow them to enter domain only. [5] - Monitor and verify DNS records [5] - Installation and using of WAF
References	[4] https://portswigger.net/web-security/ssrf [5] https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

XSS – Cross Site Scripting Injection

Description	XSS compromised the interactions with the webserver application to remove the probation functionality from the testing account. Reflected XSS vulnerability was found in “Anonymous question” page (question.php). This resulted in gaining authorisation and performing “pass_probation” function which is not initially allowed in the profile page. **Methodology**: To receive injected JavaScript responses, Beeceptor [6] has been utilised to receive injected codes responses. As a result, in the “question.php” page an XMLHttpRequest [7] successfully sent and response included an internal localhost referrer address (http://localhost:55665/x.html). Pass_probation function is an existing javascript function in “profile.php” and calling this function may result in passing probation for the given username, calling this function in the profile page or on the host address resulted in an “unauthorised” outcome. To make an internal request to call pass_probation function, question.php was used to input (inject) the JavaScript code. Injected code used a localhost following the function and given username. Doing so changed given username to a non-probation account and dropped a flag in profile page. This is discussed in Proof of concept. **Risk statement**: Improving access to non-probation mode in profile may happen due to vulnerability in “question.php” which leads to changing any username to non-probation user.
Proof of Concept	A proof of concept is attached in appendix section of this report **(Figure 3 FLAG_XSS)** Following steps can result in exploiting this vulnerability: **Step 1:** The following script can be injected in the question input field (question.php): *<script> var x = new XMLHttpRequest(); x.open("GET", 'http://localhost/pass_probation.php?user=**USERNAME**); x.send(); </script>* **Step 2:** User is no longer shown as probation in profile page and a flag is shown in profile (See Figure 2).

Impact	**An attacker could change any account access to a non-probation account and the impact of this on the organisation can be that certain functionalities can be accessed by unauthorised attackers/users.**
Risk Ratings	**High** Occurrence of this attack is “likely” since this is a cross site attack and more possible to be executed by any user on any browser without any pre-coding requirement, and the consequence is considered as Moderate because the organisation might not be willing to give certain users privilege of using non-probation accounts. However, currently at this point won’t give much access to the attacker as a non-probation account and further assessment will be required by the organisation.
Recommendation	- Since blacklisting might not be a case, because of the nature of questioning, input might include anything including tags. It is recommended to set rules for internal requests. - In the php file the given username can be controlled to check having permission before performing the function. - The installation of WAF
References	[6] https://beeceptor.com/ [7] https://www.w3schools.com/xml/xml_http.asp

SQL Wildcard – Use of % Wildcard in API calls

Description	The use of "%" on the API requests to fetch entire data has caused an SQL wild card attack. As a method for search functionality in database "like" clauses can be used to match zero or more occurrences of any characters. **Risk statement:** All data may be received in a fetch request due to not controlling inputs for wildcards characters which leads to revealing all information about courses and possibly leading to a denial of service. [8]
Proof of Concept	A proof of concept is attached in appendix section of this report **(Figure 2 FLAG_Wildcard SQL)** Steps to exploit this vulnerability: **Step 1:** Run wildcard.py. **Step 2:** Enter username and password to login and get required cookies. **Step 3:** Insert any input to fetch data (eg, OSCP). For the purpose of this exploit, enter "%" to fetch all data. **Step 4:** To find the flag enter: "COMP90074-1337"
Impact	**An attacker can get access to all data provided by API in one call and furthermore can cause Denial of service. However, it must be considered that, such a request can be required through a call.**
Risk Ratings	**Low** Consequence of this vulnerability is considered negligible since data is not considered sensitive and possibility is at the level of "possible".
Recommendation	- Do nothing can be applied. This could be ideal due to the budget and time pipeline for the organisation. - Blacklist SQL wildcard characters. - Using scaping quotes. [9] - Install and use WAF
References	[8] https://shahjerry33.medium.com/sql-wildcard-dos-hang-till-death-adbae66d1f7b [9] https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html#escaping-wildcard-characters-in-like-clauses

Appendix I - Risk Matrix

All risks assessed in this report are in line with the ISO31000 Risk Matrix detailed below:

Likelihood	Consequence				
	Negligible	Minor	Moderate	Major	Catastrophic
Rare	Low	Low	Low	Medium	High
Unlikely	Low	Low	Medium	Medium	High
Possible	Low	Medium	Medium	High	Extreme
Likely	Medium	High	High	Extreme	Extreme
Almost Certain	Medium	High	Extreme	Extreme	Extreme

Appendix 2 - Additional Information

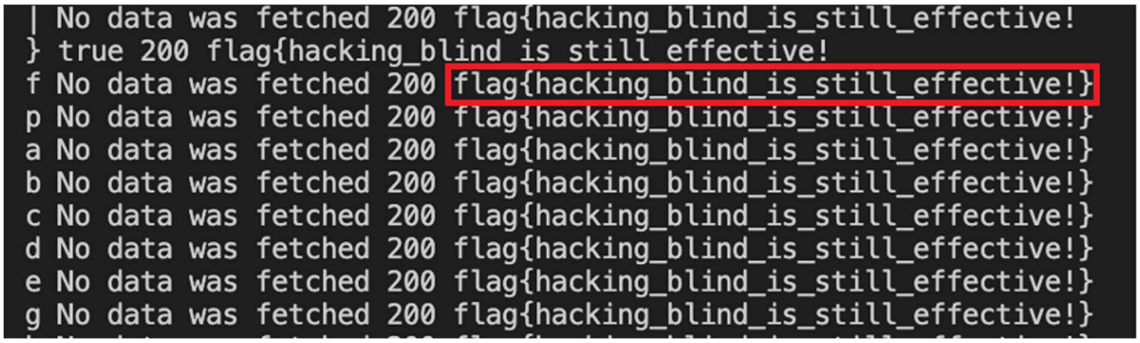

Figure 1 FLAG_SQL Injection

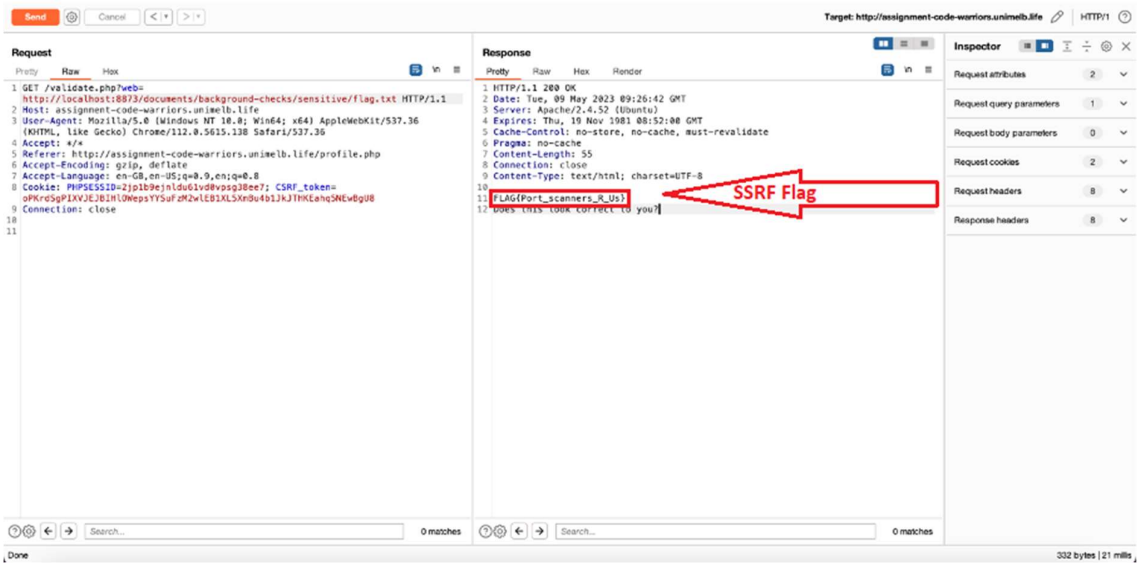

Figure 2 FLAG_SSRF

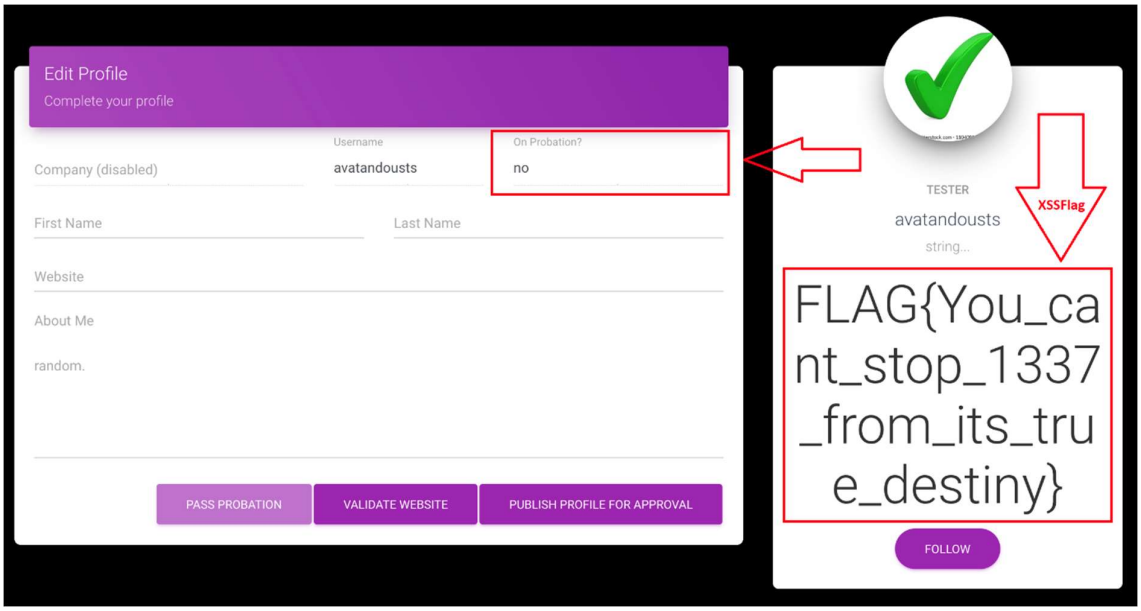

Figure 3 FLAG_XSS

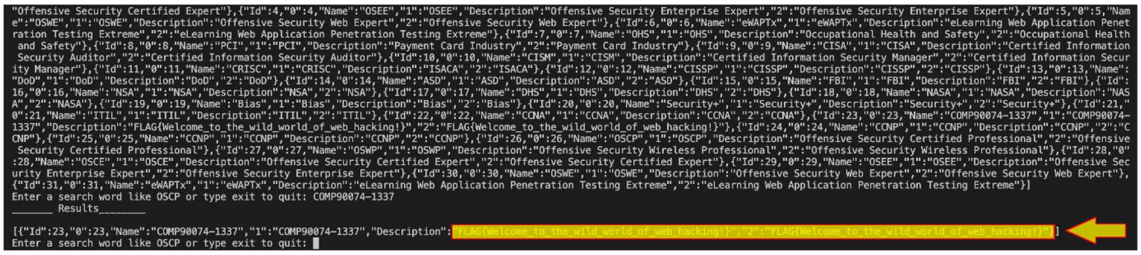

Figure 2 FLAG_Wildcard SQL