Authentication and Authorization

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#python-web

Table of Contents

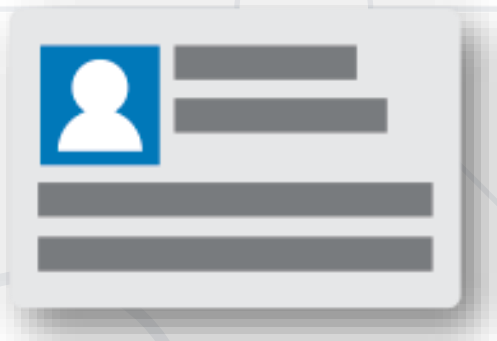
1. The **Identity** in the Web
2. **Authentication**
3. **Authentication** in **Django**
4. **Permissions** and **Authorization**
5. Web **Security**

The Identity in the Web

Authorization vs. Authentication

Authorization

What you can do



Authentication

Who you are

Authorization vs. Authentication

- **Authorization**
  - The process of determining the **user's permissions** on a computer or network
  - Questions: What **actions** are you **permitted** to **perform**? Can you **access/view** this page?
- **Authentication**
  - The process of **confirming** the **identity** of a user or computer
  - Questions: **Who** are **you**? **How** do you **prove** it?
  - Credentials can include **passwords**, **smart cards**, external **tokens**, etc.

Identification vs. Authentication

- **Identification**
  - The capability to uniquely **recognize** a **user** of a system or an **application** running within the system
  - The system utilizes the **username** to **identify** the user uniquely
- **Authentication**
  - The ability to **verify** that a **user** or **application** is **indeed** the entity it claims to be
  - The system **authenticates** the user by **checking** the **correctness** of the **password**

Authentication

How Authentication Works

- During **authentication**, the user's provided **credentials** are **compared** to those in an authorized user database
- If the credentials **match**, the process is completed, **granting** the user **access**
- The most **basic authentication** involves a user **ID** and **password**
  - Multiple **authentication factors** can **enhance** security

Authentication Factors

- These are pieces of data or attributes utilized to **authenticate** a user who requests **access** to a system
- **Single-factor authentication**
  - For instance, authentication through a **user ID** and **password**
- **Two-factor authentication**
  - This involves a **knowledge factor** (e.g., **password**) **combined** with a **biometric** or **possession factor** (e.g., **security token**) for enhanced security

Authentication in Django

Authentication in Django

Software University

- Django incorporates a **comprehensive** user **authentication system**
- It manages **both** **authentication** and **authorization**
- This system includes:
  - **Users**, **groups**, and **permissions**
  - A configurable **password hashing system**
  - Forms and view **tools** for **user login** and **content restriction**
  - A **pluggable** backend system
  - **Cookie-based** user **session** handling

Authentication in Django

Software University

- The configuration is **pre-included** in the `settings.py` file under the `INSTALLED_APPS` setting:
  - `'django.contrib.auth'`
    - Includes the **core** of the **authentication** framework along with its **default models**
  - `'django.contrib.contenttypes'`
    - Facilitates the **association** of **permissions** with **models**

Cookie-Based Authentication

- **Cookie-based authentication** in Django offers comprehensive **support** for **anonymous sessions**
- This feature **enables** the **storage** and **retrieval** of **arbitrary** data on a **per-site-visitor basis**
- The data is **stored** on the **server side**, and the mechanism **abstracts** the process of **sending** and **receiving cookies**

Cookie-Based Authentication

- Notably, the **cookies** only **contain** a **session ID**, **not** the **actual data**
- **`SessionMiddleware`** effectively **manages sessions** across **requests**
- **`AuthenticationMiddleware`** **associates users** with **requests** by utilizing **sessions**
- This **ensures** a **secure** and **efficient** approach to **handling user authentication** through **cookies** in Django

django.contrib.auth

Software University

- Serves the most **common project needs**
  - Allows **inheritance** from its **URLs**, **models**, **views**, and **forms**
- Effectively **handles** a **diverse range** of **tasks**
  - Incorporating a meticulous **implementation** of **passwords** and **permissions**
- Additionally, it **supports** **seamless extension** and **customization** of **authentication** processes

Permissions and Authorization

What is Authorization?

- Authorization encompasses the process by which an **administrator** **bestows** **rights** upon **authenticated** users
- The **privileges** and **preferences** allocated to an **authorized account** are contingent upon the user's **permissions**
- An **administrator** **sets** the **configurations** for all these environment variables

Authorization and Permissions in Django

Software University

- Django incorporates a **built-in permissions system**
  - Offering a mechanism to allocate **permissions** to **particular users** or user **groups**
- While prominently employed by the Django **admin site**, this system is **flexible**, allowing **integration** into **custom code**
- **Customization** of **permissions** for distinct object instances of the same type is achievable, providing a granular approach to **access control**

Default Permissions

- **Default permissions** are **fundamental** to **controlling access** to different **actions** and **resources** within a Django application
- They can be **assigned** to **users** or **groups**, providing a **foundation** for implementing **access control** mechanisms
- Django's **default permissions** consist of **four** main types
  - **Add**, **Change**, **Delete**, **View**

Default Permissions

Software University

- Default **permissions** in Django are **automatically** created for **each model** **defined** in the **installed applications**
  - **Add**: Grants **permission** to **create** new instances of a model
  - **Change**: Allows **modification** of existing instances of a model
  - **Delete**: Permits the **removal** of instances of a model
  - **View**: Enables the user to **view** instances of a model

*More detailed info about users, groups, and permissions will be the subject of the next presentation.

Web Security

Most Common Web Security Problems

- **SQL** Injection
- Cross-site Scripting (**XSS**)
- URL/HTTP manipulation attacks (**Parameter Tampering**)
- Cross-site Request Forgery (**CSRF**)
- Brute Force Attacks (also **DDoS**)
- Insufficient **Access** Control
- Missing **SSL** (HTTPS) / **MITM**
- Phishing/Social Engineering

SQL Injection

Software University

- The following SQL commands are executed:
  - Usual search (no **SQL injection**):

```
SELECT * FROM Messages WHERE MessageText LIKE '%Nikolay.IT%';
```

  - SQL-injected search (matches **all records**):

```
SELECT * FROM Messages WHERE MessageText LIKE '%%%';
```

```
SELECT * FROM Messages WHERE MessageText LIKE '%' or 1=1 --%';
```

  - SQL-injected **INSERT** command:

```
SELECT * FROM Messages WHERE MessageText
LIKE '%'; INSERT INTO Messages(MessageText, MessageDate)
VALUES ('Hacked!!!', '1.1.1980') --%'"
```

SQL Injection

- Original SQL Query:

```
sql_query = "SELECT * FROM user WHERE name = '" + username + "' AND pass='"
+ password + "'";
```

- Setting username to **Admin** & password to **' OR '1'= '1** produces:

```
sql_query = SELECT * FROM user WHERE name = 'Admin' AND pass='' OR '1'='1''
```

- The result
  - The user with **username** "**Admin**" will log in **WITHOUT** a password
  - The **passed query** will turn into a **Boolean** expression which is **always True**

Cross Site Scripting (XSS)

- It is a **vulnerability** that enables users to **inject client-side scripts** into the browsers of other users
  - By **storing malicious scripts** in the database, which are later retrieved and displayed to other users
  - By getting users to **click a link**, **triggering** the execution of the attacker's JavaScript within the user's browser
- This vulnerability is particularly concerning as it can originate from any **untrusted source** of data
  - When data is **not adequately sanitized before** being included on a page

XSS in Django

- Django templates **protect** you **against** the majority of **XSS attacks**
- Django templates **escape** **specific characters** that are particularly dangerous to HTML, but it is **not** **entirely** **foolproof**

```
<style class={{ var }}>...</style>
```

  - If `var` is set to `'class1 onmouseover=javascript:func()'`, this can result in **unauthorized** JavaScript execution
  - **Quoting** the attribute value would fix this case

More about XSS in Django: *https://docs.djangoproject.com/en/4.2/ref/templates/language/#automatic-html-escaping*

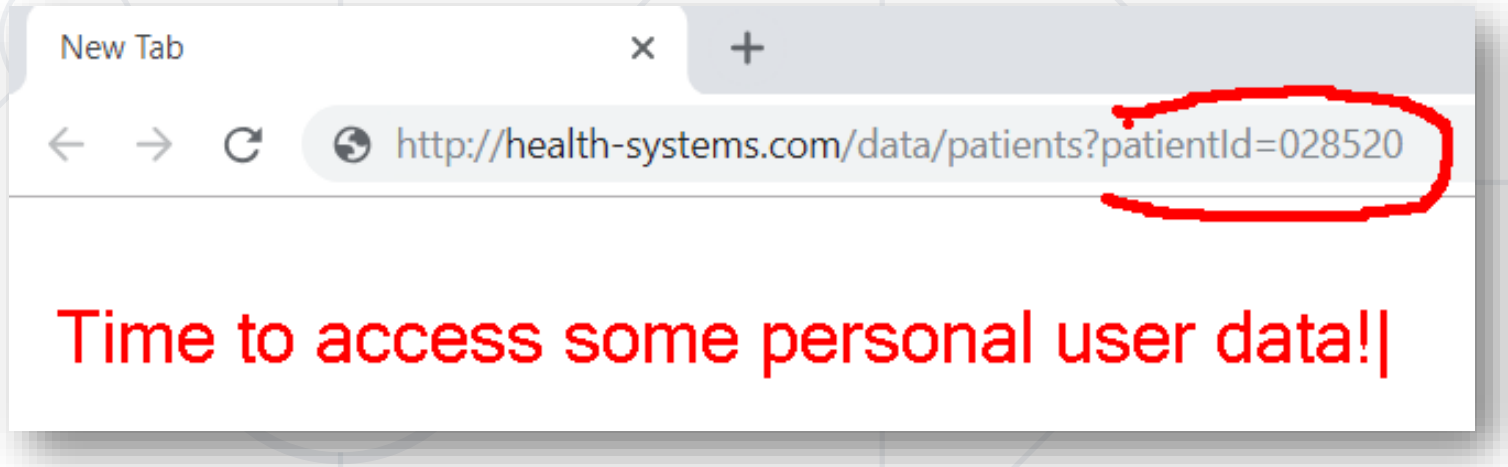
Bleach Library

- **Bleach** is a **security-focused** **HTML** **sanitizing** library based on an allowed-list approach
  - Designed to **escape** or **strip** **markup** and **attributes**
  - Primarily **crafted** for **sanitizing** **text** obtained from **untrusted** sources
- Bleach **ensures** a **secure** handling of **HTML content**
- To **incorporate** it into your project, **install** **Bleach** using the appropriate terminal command

```
pip install bleach
```

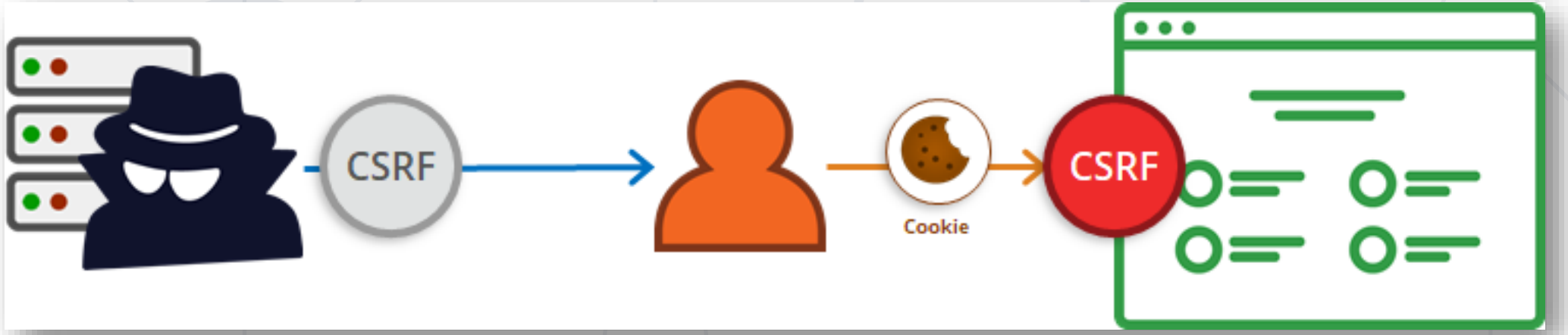
Parameter Tampering

- **Parameter Tampering** is the manipulation of **parameters** exchanged between the **client** and a **server**
  - Altered query strings, request bodies, cookies
  - Skipped data validations, Injected additional parameters



Cross-Site Request Forgery

- **Cross-Site Request Forgery** (**CSRF** / **XSRF**) is a web security attack leveraging the HTTP protocol
  - Enables the **execution** of **unauthorized** **commands** on behalf of a user by **exploiting** the **cookies** stored in their browser
  - The **attacker utilizes** the **valid permissions** of the user to **execute** requested commands **maliciously**, **without** the user's **knowledge**



Cross-Site Request Forgery

Software University

- What **Cross-Site Request Forgery** is:

```
<!-- SOME MULTI-COLOR USELESS CLICKBAIT CONTENT -->

<form action="http://good-banking-site.com/api/account" method="post">
    <input type="hidden" name="Transaction" value="withdraw">
    <input type="hidden" name="Amount" value="1000000">
    <input type="submit" value="Click to collect your prize!">
</form>
```

- Even an **accidental misclick** on the button can **trigger** the attack
- Implementing **defenses against** such attacks becomes imperative to safeguard both your application and its users

Summary

- **Identity**
- Authentication
- **Authorization**
- Default **Permissions**
- **Security**

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg