# Secure Software Engineering

# Agenda

Warum Django
Django 101
Umgesetzte Funktionen
Tests
Workflow
Ai
Fazit
Live Demo
Quellen

# Warum Django?

- Weit verbreitet
- Akitv maintained
- Python basierend
- Viele Sicherheitsvorkehrungen sind schon eingebaut
- Einfach erweiterbar (API)
- Admin Panel (Debuggen)

# Authentifizierung

- Session basierende authentifizierung mit Cookies
- Wird mitgelierfert von Django
- Cookie Eigenschaften:
  - session_key
  - session_data
  - expire_date
- HttpOnly: Kein Lesen durch javascript
- @login_required regelt view basiert die authentifizierung

Code Ausschnitt als Bespiel

# Authentifizierung

- Fixes / Schutz vor:
  - Broken Authentication
  - Session Hijacking

# Object-Relational Mapper (ORM)

- Sicheres Umgehen von Parametern und Erstellen von Abfragen, um sicherzustellen, dass böswillige Eingaben die beabsichtigte SQL-Logik nicht verändern können
- Liefert sichere funktionen wie filter() oder all()
- Schutz vor: Injection

Bespiel wie ein Model aussieht und eine query

# CSRF Protection

- Alle POST http request sind mit einem csrf token geschuetzt

Hier ein kleines Bespiel als Grafik

# Templating

- Djangos Templates escapen Variablen automatisch
  - schützt vor XSS
- Neue Sicherheitsluecke: SSTI

Hier eine Abbildung wie Templating funktoniert

Hier eine Abbildung wie SSTI funktoniert

# Views

- Python-Funktion, die Webanfragen verarbeitet und Antworten liefert
- Beliebige Logik kann implementiert werden

Hier eine Abbildung von einer simplem View

Hier eine Abbilung die bespiel Daten des request Objekts zeigt

# Forms

- Datenstruktur zur Eingabe und Validierung von User-Daten
- Prüft automatisch Eingaben auf Gültigkeit
- Kann mit beliebigen Model verknuepft werden

Hier eine Abbildung von einer View mit einem Model

Hier eine Abbildung wie das dann gerendert oder
als HTML aussieht

# Autorisierung

- AuthenticationMiddleware ermöglicht jeder View Zugriff auf request.user
- request.user repräsentiert den authentifizierten Benutzer, der die Anfrage stellt
- Schutz vor:
  - Broken Access Control
  - Insecure Direct Object References (IDOR)
  - Privilege Escalation

Hier eine Abbildung eines Bespiels mit request.user == ..

# Umgesetzte Funktionen

**Note erstellung / preview**

- markdown2html_safe: unsafe markdown → safe html
- sanitize_title: unsafe title → safe title

Place Holder fuer eine Grafik

- Nicht sanitisiertes HTML kann zu XSS führen, wenn eine Notiz öffentlich ist und von anderen Nutzern angesehen wird.

# markdown2html_safe

- Markdown parser erweitert mit Yt2iframe Extention
  - Extentention guckt nach Pattern: ![<Optional Titel>](embed:<Link>)
  - Extrahiert Video ID und checkt: http/https & valid Youtube Link
  - Convertiert zu: https://www.youtube-nocookie.com/embed/video_id
  - Erstellt iframe und wrapped in template (consent-banner)
  - Rendert markdown zu html
- Bleach: Sanitizer
  - Entfernt unsichere Tags/Attribute/Protocolle wie z.B. script oder onerror
  - Vordefinierte List an erlaubten Tags und jeweiligen Attributen
  - inframe src = ..youtube-nocookie.com..

# markdown2html_safe: Beispiel

Hier kommt noch ein Beispiel hin

# Umgesetzte Funktionen

**User Registration**

- Erwartet: username, email, password1, password2
- Django builtin Validatoren:
  - Username: max 150 zeichen lang und letters, numbers, and @ . + - _
  - Email: <name>@<domain>
  - Password:
    - Übereinstimmung zwischen dem Passwort und einer Reihe von Benutzerattributen
    - min 15 zeichen lang
    - Pruefung gegen common Passwoerter
    - Prueft ob nur nummer im Passwort sind

# Umgesetzte Funktionen

**User Registration**

- Password wird gehashed in DB gespeichert:
  - 1.000.000 iteriert
  - algo: pbkdf2_sha256
  - salt: random string
- Email wird an user versendet
  - Erst nach klicken auf den Link wird der Account aktiviert und anmeldung ist moeglich
- mögliche Schwachstellen sowie deren Vorbeugung:
  - **Username or Email Enumeration**
  - **Brute Force Attacks**
  - **Data Breach**
  - **SQL Injection**
  - **SSTI**

# Umgesetzte Funktionen

**User Login/Logout**

- login: username, password
- AuthenticationForm ueberprueft username.password == input.password
- Logout nur eine POST und session wird geloescht
- mögliche Schwachstellen sowie deren Vorbeugung:
  - **Username Enumeration**
  - **Injection**

# Umgesetzte Funktionen

**Note Suche**

- Erwartet query paramter in url
- Sucht nach notes anhand title und gibt eigene und public notes zurueck
- mögliche Schwachstellen sowie deren Vorbeugung:
  - **Broken Access Control**
  - **Cross-Site Scripting (XSS)**
  - **SQL Injection**

# Umgesetzte Funktionen

**Notes listen**

- Erwartet nichts
- Listet alle eigenen Notes und oeffentliche gestellte
- mögliche Schwachstellen sowie deren Vorbeugung:
  - **Broken Access Control**
  - **Cross-Site Scripting (XSS)**
  - **SQL Injection**

# Umgesetzte Funktionen

**Note angucken**

- Get Request zu /notes/<uuid>
- Notes koennen angeguckt werden unabhaengig vom owner
- Damit Templates nicht escaped: mark_safe(note.content)
- mögliche Schwachstellen sowie deren Vorbeugung:
  - **SQL Injection**
  - **Cross-Site Scripting (XSS)**

# Umgesetzte Funktionen

**Reset Passwort**

- Erwartet eine email
- Wenn ein User mit dieser Email existiert wird eine Email verschickt
  - Link in der Email: /reset/<uidb64>/<token>
- Neues Passwort zweimal eingeben
- Gleiche Validatoren wie bei Account Erstellung und password1 == password2
- Token wird nach erfolgreichen setzen des Password oder nach 3 Tagen ungueltig
- mögliche Schwachstellen sowie deren Vorbeugung:
  - **User Enumeration**
  - **Token Abuse**
  - **User ID Disclosure**
  - **SQL Injection**

# Umgesetzte Funktionen

**API**

- Authentifizierung via Token
- Funktionen: Token genieren, Noten listen, Note erstellen und eine spezielle Note bekommen via uuid
- Token werden nach 10h ungueltig
- Serializers handeln JSON input und output
- mögliche Schwachstellen sowie deren Vorbeugung:
  - **Unauthorized Access**
  - **Injection**
  - **Broken Access Control**

# Umgesetzte Funktionen

Weitere Features:

- Account Loeschung
- Account Profile

# Tests

- Backend
  - Redirect wenn keine session
  - Api token generation
  - Note preview: Sanitizer
- Frontend
  - Login
  - Preview Note

Video wie ein Test mit selenium aussieht

# Workflow

- IDE
  - Codium
  - Zed
- Git feature branch workflow
- Zap: HTTP Requests testen (XSS)
- XStrike

- CI/CD
  - Synk
  - CodeRappid

Hier eine Abbildung von einen Synk Scan report

# AI

- ChatGPT um Django Konzepte zu verstehen
- Einzelene Funktion via AI Agent implementiert
  - API → Kann schnell teuer werden
- Sicherheit Relevants in Prompts erwaehnen

Beispiel wo AI code 500 geworfen hat

Ai usage Diagramm

# Fazit

Was ich gelernt habe:

- Funktionalität gruendlich vor Bibliotheks-Implementierung prüfen
- Git Workflow / neu commands
- Django
- Testen testen testen (implementierung von feature → neuer Bug)
- Wie kompleziert die DSGVO ist
- Neu Sicherheitsluecken wie SSTI und Clickjacking
- Chromium > Firfox (Website debuggin)

Allgemein Spass gemacht sichere Webanwendung zu bauen (Blue Team)

Sicherheitskonzepte vertieft → mehr Kompetenz Pentesting

# Live Demo

# Quellen

# Danke fuers zuhoeren :)