Документација – Лабораториска Вежба 2

1. Вовед

1.1 Цел на проектот

Овој проект претставува комплетен систем за автентикација развиен во Python Flask, дизајниран да обезбеди максимална безбедност при управување со кориснички сметки. Системот имплементира модерни безбедносни протоколи вклучувајќи two-factor authentication (2FA), безбедно управување со сесии и административен надзор.

1.2 Проблематика и Решение

Во современата дигитална ера, заштитата на корисничките податоци и идентитети е од клучно значење. Овој систем решава неколку критични безбедносни предизвици:

- Заштита од неовластен пристап преку повеќестепена автентикација
- Безбедно складирање на лозинки со напредни техники за хеширање
- Заштита од сесиски напади преку правилно управување со сесии
- Спречување на SQL injection напади

1.3 Технолошка Архитектура

- Системот користи следниве технологии:
- Backend Framework: Flask 2.0+
- База на податоци: SQLite3
- Безбедносни библиотеки: Werkzeug Security
- Template Engine: Jinja2
- Стилизација: Pure CSS3

2. Теоретска Основа

2.1 Принципи на Безбедна Автентикација

Системот се заснова на три основни принципи на безбедноста:

2.1.1 Конфиденцијалност (Confidentiality)

- Корисничките лозинки никогаш не се складираат во чист текст
- Сите комуникации се заштитени со хеширање и енкрипција
- Session tokens се чуваат безбедно во HTTP-only cookies

2.1.2 Интегритет (Integrity)

- Податоците се валидираат на повеќе нивоа
- Корисничките внеси се санитизирани пред обработка
- Database transactions обезбедуваат конзистентност на податоците

2.1.3 Достапност (Availability)

- Системот е дизајниран да биде отпорен на DoS напади
- Автоматско чистење на истечени податоци
- Ефикасни database queries за одржување на перформансите

2.2 Криптографски Техники

Системот имплементира неколку клучни криптографски техники:

2.2.1 Хеширање на Лозинки

```
password_hash = generate_password_hash(password)
```

Технички детали:

- Алгоритам: PBKDF2 (Password-Based Key Derivation Function 2)
- Хеш функција: SHA-256
- Salt: Автоматски генериран 16-бајтен salt
- Итерации: 600,000 (Flask default)

2.2.2 Session Token Генерација

```
@staticmethod
def generate_session_token():
    """Generates secure session token"""
    return secrets.token_urlsafe(32)
```

Технички детали:

- Должина: 32 бајти (256 бита)
- Ентропија: Користи системски secure random generator
- Кодирање: URL-safe base64

3. Database Класа - Детална Анализа

3.1 Иницијализација на Базата

```
def init_database(self):
    """Initializes the database with required tables"""
    conn = sqlite3.connect(self.db_name)
    cursor = conn.cursor()

    # Users table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            email TEXT UNIQUE NOT NULL,
            password_hash TEXT NOT NULL,
            is_verified BOOLEAN DEFAULT FALSE,
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            last_login DATETIME NULL
        )
    ''')
```

- PRIMARY KEY AUTOINCREMENT: Обезбедува уникатни идентификатори и автоматизира нивно доделување
- UNIQUE ограничувања: Гаранираат интегритет на податоците на ниво на база
- DEFAULT вредности: Обезбедуваат конзистентност при внесување на нови записи
- FOREIGN KEY ограничувања: Одржуваат релационен интегритет меѓу табелите

3.2 Query Execution Метод

```python
def execute_query(self, query, params=(), fetchone=False, fetchall=False):
    """Executes database query"""
    conn = self.get_connection()
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()

    try:
        cursor.execute(query, params)

        if fetchone:
            result = cursor.fetchone()
            if result:
                result = dict(result)
        elif fetchall:
            result = cursor.fetchall()
            if result:
                result = [dict(row) for row in result]
        else:
            result = None

        conn.commit()
        return result

    except Exception as e:
        conn.rollback()
        print(f"Database error: {e}")
        raise e
    finally:
        conn.close()
```

Безбедносни карактеристики:

- **Parameterized queries:** Спречува SQL injection напади
- **Transaction management:** Обезбедува atomicity на операциите
- **Resource management:** Автоматско затворање на database конекции

4. Flask Application Architecture

4.1 Конфигурација на Апликацијата

```python
app = Flask(__name__)
app.secret_key = 'your-very-secret-key-change-in-production-2024'
app.config['SESSION_COOKIE_HTTPONLY'] = True
app.config['SESSION_COOKIE_SECURE'] = False
app.config['SESSION_COOKIE_SAMESITE'] = 'Strict'
```

Безбедносна анализа на конфигурацијата:

4.1.1 Session Cookie Безбедност

- **HTTPOnly:** Спречува пристап до cookies преку JavaScript (XSS заштита)
- **SameSite=Strict:** Спречува CSRF (Cross-Site Request Forgery) напади
- **Secure Flag:** Во production, ова би било True за HTTPS-only cookies

4.1.2 Secret Key Имплементација

- **Улога:** Користен за сигнирање на session cookies и CSRF tokens
- **Production препорака:** Треба да биде сложен string и да се чува безбедно

4.2 Authentication Middleware - Длабинска Анализа

```python
# Authentication middleware
@app.before_request
def check_auth():
    """Authentication middleware"""
    public_routes = ['login', 'register', 'verify_email', 'verify_login', 'static']

    if request.endpoint in public_routes:
        return

    session_token = request.cookies.get('session_token')
    if session_token:
        session_data = db.validate_session(session_token)
        if session_data:
            session['user_id'] = session_data['user_id']
            session['username'] = session_data['username']
            session['is_admin'] = (session_data['username'] == ADMIN_CONFIG['admin_username'])
            return

    if 'user_id' not in session:
        return redirect(url_for('login'))
```

Објаснување:

- **Middleware патерн:** Кодот се извршува пред секој request
- **White-list пристап:** Сите routes се приватни освен експлицитно наведените
- **Dual authentication:** Проверува и session и cookie-based authentication

4.3 Password Валидација - Комплексна Имплементација

```python
@staticmethod
def validate_password(password):
    """Password validation"""
    if len(password) < 8:
        return False, "Password must be at least 8 characters long"

    if not re.search(r"[A-Z]", password):
        return False, "Password must contain at least one uppercase letter"

    if not re.search(r"[a-z]", password):
        return False, "Password must contain at least one lowercase letter"

    if not re.search(r"[0-9]", password):
        return False, "Password must contain at least one digit"

    if not re.search(r"[!@#$%^&*(),.?\":{}|<>-_]", password):
        return False, "Password must contain at least one special character"

    return True, "Password is valid"
```

Криптографска анализа:

- **Ентропија:** Комбинацијата од 4 различни множества знаци ја зголемува ентропијата
- **Complexity:** Minimum 8 знаци обезбедува ~30 бита ентропија
- **Dictionary attack resistance:** Специјални знаци го спречуваат dictionary attacks

5. Two-Factor Authentication Систем

5.1 Теоретска Основа на 2FA

Two-Factor Authentication комбинира два од трите фактори на автентикација:

1. **Knowledge Factor** (лозинка - нешто што корисникот знае)
2. **Possession Factor** (email пристап - нешто што корисникот има)
3. **Inherence Factor** (биометрија - нешто што корисникот е)

5.2 Имплементација на 2FA Протокол

5.2.1 Генерирање на Верификациони Кодови

```python
@staticmethod
def generate_verification_code():
    """Generates 6-digit code"""
    return str(random.randint(100000, 999999))
```

Безбедносни карактеристики:

- **6-цифрен код:** 1,000,000 можни комбинации
- **Time-limited:** Кодот е валиден 10 минути
- **Single-use:** Секој код може да се користи само еднаш

5.2.2 Верификационен Процес

```python
def verify_code(self, user_id, code, purpose):
    """Verifies code and marks it as used"""
    query = """
        SELECT id FROM verification_codes
        WHERE user_id = ? AND code = ? AND purpose = ?
        AND datetime(expires_at) > datetime('now') AND used = FALSE
    """

    code_record = self.execute_query(
        query, (user_id, code, purpose), fetchone=True
    )

    if code_record:
        update_query = "UPDATE verification_codes SET used = TRUE WHERE id = ?"
        self.execute_query(update_query, (code_record['id'],))
        return True

    return False
```

Database-level валидација:

- **Time validation:** Користи SQL datetime functions за проверка на валидноста
- **State management:** used flag спречува повторна употреба
- **Purpose-specific:** Кодовите се специфични за намената (регистрација/најава)

5.3 Session Management Систем

5.3.1 Креирање на Сесија

```python
def create_session(self, user_id, session_token, expires_hours=24):
    """Creates a new session"""
    expires_at = datetime.now() + timedelta(hours=expires_hours)

    query = """
        INSERT INTO sessions (user_id, session_token, expires_at)
        VALUES (?, ?, ?)
    """
    try:
        self.execute_query(query, (user_id, session_token, expires_at.strftime('%Y-%m-%d %H:%M:%S')))

        # Update last_login with proper datetime
        self.execute_query(
            "UPDATE users SET last_login = datetime('now') WHERE id = ?",
            (user_id,)
        )
        return True
    except Exception as e:
        print(f"Error creating session: {e}")
        return False
```

Session безбедност:

- **Fixed expiration:** 24-часовен животен век на сесиите
- **Database-backed:** Session state се чува на серверска страна
- **Token-based:** Stateless authentication преку secure tokens

5.3.2 Валидација на Сесија

```python
def validate_session(self, session_token):
    """Validates session token - UPDATED to use datetime comparison"""
    query = """
        SELECT s.*, u.username, u.email, u.is_verified
        FROM sessions s
        JOIN users u ON s.user_id = u.id
        WHERE s.session_token = ? AND datetime(s.expires_at) > datetime('now') AND u.is_verified = TRUE
    """

    return self.execute_query(query, (session_token,), fetchone=True)
```

Мулти-условна валидација:

- **Token validity:** Проверува дали token постои
- **Time validity:** Проверува дали сесијата не е истечена
- **User status:** Проверува дали корисникот е верифициран

6. Безбедносна Анализа и Заштитни Механизми

6.1 Заштита од Распространети Напади

6.1.1 SQL Injection Заштита

```
query = "SELECT * FROM users WHERE username = ?"
return self.execute_query(query, (username,), fetchone=True)
```

Техничко објаснување:
Parameterized queries работат со подготовени statements каде параметрите се третираат како податоци наместо како дел од SQL кодот, спречувајќи извршување на малициозен SQL.

6.1.2 XSS (Cross-Site Scripting) Заштита

- **Jinja2 auto-escaping:** Автоматско escape-ување на HTML characters
- **HTTPOnly cookies:** Спречува пристап до session cookies преку JavaScript
- **Content Security Policy:** (Може да се додаде) дополнителна заштита

6.1.3 CSRF (Cross-Site Request Forgery) Заштита

```
app.config['SESSION_COOKIE_SAMESITE'] = 'Strict'
```

Механизам: SameSite cookies не се испраќаат со cross-site requests, спречувајќи CSRF напади.

6.2 Rate Limiting и Brute-Force Заштита

- **Временско ограничување:** Кодовите се валидни 10 минути
- **Single-use codes:** Секој код може да се користи само еднаш
- **Session expiration:** Автоматско истекување на сесиите

7. Работни Процеси и Кориснички Патеки

7.1 Регистрационен Процес - Детална Секвенца

1. GET /register
2. Прикажување на регистрациска форма
3. POST /register (клиент испраќа податоци)
4. Серверска валидација на податоци
5. Хеширање на лозинката
6. Креирање на корисник во базата (is_verified = FALSE)
7. Генерирање на верификационен код
8. Зачувување на кодот во базата
9. Испраќање на кодот преку email (симулација)
10. Пренасочување кон /verify-email

7.2 Верификационен Процес

1. GET /verify-email
2. Прикажување на форма за верификационен код
3. POST /verify-email (клиент внесува код)
4. Валидација на кодот во базата
5. Ако кодот е валиден:
   - Ажурирање на корисник (is_verified = TRUE)
   - Означување на кодот како искористен
   - Пренасочување кон /login
6. Ако кодот е невалиден:
   - Прикажување на грешка
   - Останување на истата страна

7.3 Најавени Процес со 2FA

1. GET /login
2. Прикажување на форма за најава
3. POST /login (credentials валидација)
4. Проверка на лозинка и верификација
5. Генерирање на 2FA код
6. Зачувување и испраќање на 2FA код
7. Пренасочување кон /verify-login
8. POST /verify-login (2FA валидација)
9. Креирање на сесија и ажурирање на last_login
10. Поставување на session cookie
11. Пренасочување кон /dashboard

8. Административни Функции и Database Viewer

8.1 Привилегии и Пристапна Контрола

```
def is_admin():
    """Check if current user is admin"""
    return session.get('is_admin', False)

# Routes
@app.route('/')
def index():
    return redirect(url_for('login'))
```

Модел на пристапна контрола:

- **Role-based access:** Администраторски привилегии базирани на корисничка улога
- **Middleware проверка:** Автоматска валидација на привилегиите
- **Graceful degradation:** Прикажување на соодветна порака за грешка

8.2 Database Maintenance и Оптимизација

```python
def cleanup_expired_data(self):
    """Cleans up expired verification codes and sessions - UPDATED to use datetime"""
    try:
        # Delete expired verification codes
        self.execute_query(
            "DELETE FROM verification_codes WHERE datetime(expires_at) <= datetime('now') OR used = TRUE"
        )

        # Delete expired sessions
        self.execute_query(
            "DELETE FROM sessions WHERE datetime(expires_at) <= datetime('now')"
        )

        print("✅ Expired data cleaned up successfully")
        return True
    except Exception as e:
        print(f"Error during cleanup: {e}")
        return False
```

Перформансни карактеристики:

- **Автоматско чистење:** Одржување на оптимална големина на базата
- **Indexed queries:** Брзо извршување благодарение на индексите
- **Transaction safety:** Atomic операции за конзистентност на податоците

9. Заклучок

Овој Flask базиран систем за автентикација успешно имплементира комплетен безбедносен framework за управување со кориснички идентитети, комбинирајќи ги најдобрите практики во веб безбедноста со robust software engineering принципи. Преку имплементација на two-factor authentication со временски ограничени кодови, безбедно хеширање на лозинки со PBKDF2 алгоритам, заштита од SQL injection преку parameterized queries и signal-безбедносни сесии со HTTP-only cookies, системот обезбедува повеќеслојна заштита од современи закани. Модуларната архитектура со јасно одвоени database, business logic и presentation слоеви овозможува лесно одржување и проширување, додека целосниот authentication lifecycle - од регистрација со email верификација до безбедна најава со 2FA и административен надзор - го прави ова решение компактен но моќен инструмент за управување со кориснички пристапи што ги исполнува сите индустриски стандарди за безбедност.

- Линк до кодот: https://github.com/itztinq/flask-user-2fa