توثيق نظام البنك الآمن

نظام البنك الآمن هو تطبيق ويب مصمم لمحاكاة بيئة مصرفية إلكترونية آمنة تتيح للمستخدمين إدارة حساباتهم البنكية وإجراء التحويلات المالية مع ضمان سرية البيانات وحمايتها من التهديدات الإلكترونية الشائعة.
يهدف المشروع إلى تطبيق مبادئ البرمجة الآمنة (Secure Programming) من خلال:

- **تحديد الأصول المهمة للنظام وحمايتها.**
- **دراسة التهديدات المحتملة وتحليلها باستخدام نماذج أمنية متعارف عليها مثل STRIDE وDREAD.**
- **تطوير حالات استخدام آمنة وحالات إساءة (Abuse Cases) لزيادة الوعي بالمخاطر.**
- **تطبيق أفضل ممارسات التشفير، وإدارة الجلسات، والتحقق من المدخلات، والحماية من الهجمات الشائعة مثل SQL Injection وXSS وCSRF.**

يمثل المشروع تطبيقًا عمليًا لمعايير الأمان الحديثة ويُظهر كيفية دمج متطلبات الأمان ضمن دورة حياة تطوير البرمجيات (SDLC) لضمان موثوقية وأمان النظام المصرفي.

الجزء الأول: توثيق المشروع

1. هندسة متطلبات الأمان (SRE) (Security Requirements Engineering)

الأصول (4 – Assets) أصول رئيسية

1. **بيانات المستخدمين الشخصية**: أسماء، عناوين، أرقام هواتف، بريد إلكتروني.
2. **معلومات الحسابات البنكية**: أرقام الحسابات، الأرصدة، تفاصيل المعاملات.
3. **بيانات المصادقة**: كلمات المرور المشفرة، جلسات المستخدمين.
4. **سجلات المعاملات المالية**: تاريخ التحويلات، المبالغ، عناوين IP.

التهديدات (4 – Threats) تهديدات رئيسية

1. **SQL Injection**: محاولة حقن أكواد SQL ضارة للوصول للبيانات.
2. **Cross-Site Scripting (XSS)**: حقن أكواد JavaScript لسرقة جلسات المستخدمين.
3. **Brute Force Attacks**: محاولات اختراق كلمات المرور بالقوة.
4. **Session Hijacking**: اختطاف جلسات المستخدمين المصادق عليهم.

المتطلبات الأمنية (4 – Security Requirements) متطلبات

1. **تشفير البيانات الحساسة**: جميع البيانات المالية والشخصية مشفرة.
2. **مصادقة قوية**: كلمات مرور معقدة + قفل الحساب بعد 3 محاولات فاشلة.
3. **حماية من الحقن**: استخدام Parameterized Queries وتنظيف المدخلات.
4. **إدارة جلسات آمنة**: انتهاء صلاحية الجلسات + CSRF protection.

2. Abuse Cases و Security Use Cases

Abuse Cases (حالات الإساءة) – حالتان:

1. **محاولة اختراق الحساب:**
   - المهاجم: مستخدم ضار.
   - الهدف: الوصول لحساب مستخدم آخر.
   - الطريقة: تجربة كلمات مرور متعددة.
   - الحماية: قفل الحساب بعد 3 محاولات + قفل IP بعد 50 محاولة.
2. **محاولة حقن SQL:**
   - المهاجم: مطور ضار
   - الهدف: الوصول لقاعدة البيانات
   - الطريقة: إدخال أكواد SQL في نماذج الإدخال
   - الحماية: استخدام SQLAlchemy ORM + تنظيف المدخلات

Security Use Cases (حالات الاستخدام الآمنة) – حالتان

1. **تسجيل دخول آمن:**
   - المستخدم: عميل البنك.
   - الهدف: الوصول الآمن للحساب.
   - الخطوات: إدخال بيانات صحيحة ← التحقق ← إنشاء جلسة آمنة.
   - الحماية: تشفير كلمة المرور + CSRF token + session timeout.
2. **تحويل أموال آمن:**
   - المستخدم: عميل مصادق عليه.
   - الهدف: تحويل أموال بأمان.
   - الخطوات: اختيار الحسابات ← إدخال المبلغ ← تأكيد ← تنفيذ.
   - الحماية: التحقق من الرصيد + تسجيل المعاملة + تشفير البيانات.

3. Threat Modeling

STRIDE Model

- **Spoofing (انتحال الهوية):** حماية بـ strong authentication.
- **Tampering (التلاعب):** حماية بـ data integrity checks.
- **Repudiation (الإنكار):** حماية بـ audit logging.
- **Information Disclosure (كشف المعلومات):** حماية بـ encryption.
- **Denial of Service (رفض الخدمة):** حماية بـ rate limiting.
- **Elevation of Privilege (رفع الصلاحيات):** حماية بـ authorization checks.

DREAD Risk Assessment

التهديد	Damage	Reproducibility	Exploitability	Affected Users	Discoverability	المجموع
SQL Injection	10	8	6	10	7	41 (عالي)
XSS	8	9	8	8	8	41 (عالي)
Brute Force	9	10	5	6	9	39 (عالي)
Session Hijacking	9	6	7	7	6	35 (متوسط)

الجزء الثاني: تطبيق الحماية

1. الحماية من الهجمات الشائعة

Anti-XSS Protection

```python
# في utils/security.py
def sanitize_input(input_data, allowed_tags=None):
    if not input_data:
        return input_data
    if allowed_tags is None:
        allowed_tags = []
    cleaned = bleach.clean(input_data, tags=allowed_tags, strip=True)
    return cleaned.strip()
```

CSRF Protection

```python
# في app.py
csrf = CSRFProtect(app)

# في كل نموذج HTML
<input type="hidden" name="csrf_token" value="{{ csrf_token }}">
```

SQL Injection Prevention

```python
# استخدام SQLAlchemy ORM بدلاً من raw SQL
user = User.query.filter_by(username=username).first()
# بدلاً من: cursor.execute(f"SELECT * FROM users WHERE username='{username}'")
```

Path Traversal Protection

```python
# التحقق من المسارات وتنظيفها
def safe_path(path):
    return os.path.normpath(path).replace('..', '')
```

Input Validation and Sanitization .2

```python
# في utils/security.py
@staticmethod
def validate_account_number(account_number):
    if not account_number:
        return False
    pattern = r'^\d{10}$'
    return bool(re.match(pattern, account_number))

@staticmethod
def validate_amount(amount_str):
    try:
        amount = float(amount_str)
        return amount > 0 and amount <= 1000000
    except (ValueError, TypeError):
        return False
```

Secure Coding Practices .3

Authentication & Authorization

```
# في models/user.py
def set_password(self, password):
    self.password_hash = generate_password_hash(password, method='pbkdf2:sha256:100

def check_password(self, password):
    return check_password_hash(self.password_hash, password)

# في routes/auth.py
@login_required
def protected_route():
    # محمية بمصادقة المستخدم
```

Cryptography System

```
# في utils/security.py
def encrypt_sensitive_data(self, data):
    return self._cipher_suite.encrypt(str(data).encode()).decode()

def decrypt_sensitive_data(self, encrypted_data):
    return self._cipher_suite.decrypt(encrypted_data.encode()).decode()
```

Session Management

```
# في config.py
PERMANENT_SESSION_LIFETIME = timedelta(minutes=30)
SESSION_COOKIE_SECURE = True
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_SAMESITE = 'Lax'
```

Error Handling

```
# في app.py
@app.errorhandler(500)
def internal_error(error):
    db.session.rollback()
    return render_template('errors/500.html'), 500
```

Rate Limiting .4

```
# في app.py
limiter = Limiter(
    app,
    key_func=get_remote_address,
    default_limits=["100 per hour"]
)

# قفل الحساب بعد 3 محاولات
def lock_account(self):
    self.failed_login_attempts += 1
    if self.failed_login_attempts >= 3:
        self.account_locked_until = datetime.utcnow() + timedelta(minutes=30)

# قفل IP بعد 50 محاولة في الساعة
limiter.limit("50 per hour")(auth_bp)
```

Database Security .5

```
# استخدام SQLAlchemy مع حماية من SQL Injection
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()

# جميع الاستعلامات تستخدم ORM
users = User.query.filter_by(username=username).all()
```

Encryption for Sensitive Data .6

```
# تشفير أرقام الحسابات والبيانات الحساسة
encrypted_account = security.encrypt_sensitive_data(account_number)

# تشفير قوي لكلمات المرور
password_hash = generate_password_hash(password, method='pbkdf2:sha256:100000')
```

معايير الأمان المطبقة

- حماية من XSS, CSRF, SQL Injection, Path Traversal
- تشفير البيانات الحساسة
- مصادقة وتفويض آمن
- إدارة جلسات محمية
- معالجة أخطاء آمنة
- تحديد معدل الطلبات (Rate Limiting)
- فصل Models, Routes, Templates
- استخدام SQLAlchemy
- تشفير قوي لكلمات المرور

المطور : صخر محمد فراشة