

Analyse de sécurité - Projet JO E-Tickets

Table des matières

1. Authentification des utilisateurs
2. Billets électroniques
3. Protection des données personnelles (RGPD)
4. Sécurité de l'application web
5. Sécurité de l'infrastructure
6. Traçabilité et détection des fraudes
7. Conclusion

1. Authentification des utilisateurs

Éléments à sécuriser

- Identifiants de connexion
- Processus d'authentification
- Sessions utilisateurs

Méthodologie

- **Hashage des mots de passe:**
 - Utilisation de bcrypt, un algorithme de hachage à sens unique avec salage intégré
 - Protection contre les attaques par force brute et par table arc-en-ciel
 - Implémentation via Flask-Bcrypt

python

```
# Hashage du mot de passe Lors de L'inscription
hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')

# Vérification du mot de passe Lors de La connexion
is_valid = bcrypt.check_password_hash(user.password, submitted_password)
```

- **Politique de mots de passe robuste:**
 - Minimum 8 caractères
 - Au moins une lettre majuscule
 - Au moins une lettre minuscule
 - Au moins un chiffre

- Au moins un caractère spécial
- Validation côté serveur pour empêcher le contournement des règles

python

```
def validate_password(password):
    if len(password) < 8:
        return False, "Le mot de passe doit contenir au moins 8 caractères."
    if not any(c.isupper() for c in password):
        return False, "Le mot de passe doit contenir au moins une lettre majuscule."
    if not any(c.islower() for c in password):
        return False, "Le mot de passe doit contenir au moins une lettre minuscule."
    if not any(c.isdigit() for c in password):
        return False, "Le mot de passe doit contenir au moins un chiffre."
    if not re.search(r'[!@#$%^&(),.?":{}|<>]', password):
        return False, "Le mot de passe doit contenir au moins un caractère spécial."
    return True, "Le mot de passe respecte la politique de sécurité."
```

- **Authentification à deux facteurs (2FA):**

- Implémentation via TOTP (Time-based One-Time Password)
- Utilisation de la bibliothèque pyotp pour générer des codes temporaires
- Vérification des codes via un algorithme temporel
- Stockage sécurisé des secrets 2FA dans la base de données

python

```
# Génération du secret 2FA
def generate_2fa_secret():
    return pyotp.random_base32()

# Vérification du code 2FA
def verify_2fa_code(user, code):
    if not user.code_2fa_secret:
        return False
    totp = pyotp.TOTP(user.code_2fa_secret)
    return totp.verify(code)
```

- **Gestion des sessions:**

- Génération de jetons de session aléatoires et complexes

- Expiration automatique des sessions après inactivité (30 minutes)
- Stockage sécurisé côté serveur
- Régénération de l'ID de session après login pour prévenir la fixation de session

python

```
@app.before_request
def before_request():
    session.permanent = True
    app.permanent_session_lifetime = timedelta(minutes=30)
    session.modified = True
    g.user = current_user
```

- **Protection contre les attaques par force brute:**

- Limitation du nombre de tentatives de connexion (5 tentatives max)
- Délai croissant entre les tentatives échouées
- Verrouillage temporaire du compte après plusieurs échecs (15 minutes)

2. Billets électroniques

Éléments à sécuriser

- Génération des billets
- Clés de sécurité
- QR codes
- Vérification des billets

Méthodologie

- **Système de double clé:**

- **Clé utilisateur:** Générée à l'inscription, unique pour chaque utilisateur, invisible pour l'utilisateur

python

```
def generate_user_key(user_id, email):
    key_material = f"{uuid.uuid4()}{email}{user_id}{datetime.utcnow().timestamp()}{SECRET_KEY}"
    return hashlib.sha256(key_material.encode()).hexdigest()
```

- **Clé d'achat:** Générée à chaque transaction, unique pour chaque commande

```
python
```

```
def generate_purchase_key(user_id, order_id):  
    ... key_material = f"{uuid.uuid4()}{user_id}{order_id}{datetime.utcnow().timestamp()}{SECRET_KEY}"  
    ... return hashlib.sha256(key_material.encode()).hexdigest()
```

- **Clé combinée:** Fusion des deux clés précédentes pour chaque billet

```
python
```

```
def combine_keys(user_key, purchase_key):  
    ... combined = f"{user_key}|{purchase_key}|{uuid.uuid4()}|{SECRET_KEY}"  
    ... return hashlib.sha256(combined.encode()).hexdigest()
```

- **Génération sécurisée des QR codes:**

- Encodage des informations clés et de la clé combinée dans le QR code
- Structure des données JSON avec champs cryptés
- Utilisation d'une correction d'erreur élevée pour garantir la lisibilité

```
python
```

```
def generate_qrcode_data(ticket):
    data = {
        'qr_id': str(uuid.uuid4()), # Identifiant unique pour ce QR code
        'ticket_id': str(ticket.id),
        'key': ticket.cle_billet,
        'offer_id': ticket.offer_id,
        'user_id': ticket.user_id,
        'generated_at': datetime.utcnow().isoformat()
    }

    return json.dumps(data)

def create_qrcode_image(data):
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_H, # Haute correction d'erreur
        box_size=10,
        border=4,
    )
    qr.add_data(data)
    qr.make(fit=True)

    img = qr.make_image(fill_color="black", back_color="white")

    # Conversion en base64 pour L'affichage web
    buffered = BytesIO()
    img.save(buffered, format="PNG")
    img_str = base64.b64encode(buffered.getvalue()).decode('utf-8')

    return f"data:image/png;base64,{img_str}"
```

- **Validation des billets:**

- Vérification de l'authenticité par comparaison des clés
- Vérification du statut (utilisé/non utilisé)
- Marquage immédiat comme "utilisé" après validation pour éviter la réutilisation
- Journalisation des tentatives de validation

```
python
```

```
def verify_qrcode(qr_data):
    try:
        # Décoder Les données JSON
        data = json.loads(qr_data)

        # Extraire Les informations du billet
        ticket_id = data.get('ticket_id')
        key = data.get('key')

        if not ticket_id or not key:
            return False, "QR code invalide"

        # Récupérer Le billet
        ticket = Ticket.query.get(int(ticket_id))

        if not ticket:
            return False, "Billet introuvable"

        # Vérifier que le billet est valide
        if not ticket.est_valide:
            return False, "Billet déjà utilisé ou annulé"

        # Vérifier que la clé correspond
        if ticket.cle_billet != key:
            return False, "Clé de billet invalide"

        return True, ticket

    except json.JSONDecodeError:
        return False, "Format de QR code invalide"
    except Exception as e:
        return False, f"Erreur lors de la vérification: {str(e)}"
```

3. Protection des données personnelles (RGPD)

Éléments à sécuriser

- Informations personnelles des utilisateurs
- Données de paiement
- Journaux d'activité

Méthodologie

- **Minimisation des données:**

- Collecte uniquement des données nécessaires
- Séparation des données d'identification et des données transactionnelles
- Justification claire de chaque donnée collectée

- **Chiffrement des données sensibles:**

- Chiffrement des données personnelles en base de données
- Utilisation de Fernet (cryptographie symétrique) pour les champs sensibles
- Clés de chiffrement stockées séparément de la base de données

python

```
from cryptography.fernet import Fernet

# Génération d'une clé
key = Fernet.generate_key()

# Chiffrement
fernet = Fernet(key)
encrypted_data = fernet.encrypt(personal_data.encode())

# Déchiffrement
decrypted_data = fernet.decrypt(encrypted_data).decode()
```

- **Gestion des consentements:**

- Recueil explicite du consentement lors de l'inscription
- Possibilité de retirer le consentement
- Journal des consentements horodaté
- Information claire sur l'utilisation des données

- **Procédures d'effacement:**

- Mécanisme de suppression des comptes à la demande
- Anonymisation des données historiques nécessaires
- Suppression en cascade des données associées

```
python
```

```
def anonymize_user(user_id):
    user = User.query.get(user_id)
    if not user:
        return False

    # Anonymisation des données personnelles
    user.email = f"anonymized_{uuid.uuid4().hex[:8]}@example.com"
    user.username = f"user_{uuid.uuid4().hex[:8]}"
    user.nom = "Anonymisé"
    user.prenom = "Utilisateur"
    user.password = bcrypt.generate_password_hash("DELETED_ACCOUNT").decode('utf-8')

    # Conservation des données de sécurité pour maintenir l'intégrité des billets
    # mais en les rendant inutilisables pour de nouvelles actions
    user.est_verifie = False
    user.est_2fa_active = False

    db.session.commit()
    return True
```

- **Traitement des données de paiement:**

- Non-stockage des données de carte complètes
- Utilisation de tokens de paiement pour les transactions
- Simulation dans l'application actuelle, mais architecture prête pour intégration sécurisée d'un prestataire de paiement

4. Sécurité de l'application web

Éléments à sécuriser

- Requêtes HTTP
- Formulaires web
- API de l'application

Méthodologie

- **Protection contre les vulnérabilités OWASP Top 10:**

- **Injection (SQL, NoSQL):**

- Utilisation d'ORM (SQLAlchemy) avec requêtes paramétrées
 - Pas de construction dynamique de requêtes SQL

```
python
```

```
# Sécurisé contre L'injection SQL
user = User.query.filter_by(email=form.email.data).first()

# À éviter absolument
# query = f"SELECT * FROM users WHERE email = '{email}'"
```

- **XSS (Cross-Site Scripting):**

- Échappement automatique des variables dans les templates Jinja2
- Validation stricte des entrées utilisateur

```
html
```

```
<!-- Sécurisé contre XSS grâce à L'échappement automatique -->
<p>{{ user_input }}</p>

<!-- À éviter absolument -->
<!-- <p>{% raw %}{{ user_input|safe }}{% endraw %}</p> -->
```

- **CSRF (Cross-Site Request Forgery):**

- Génération de tokens CSRF pour tous les formulaires via Flask-WTF
- Vérification du token à chaque soumission de formulaire

```
python
```

```
# Dans Le formulaire
class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Mot de passe', validators=[DataRequired()])
    submit = SubmitField('Se connecter')

# Dans La route
@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit(): # Vérifie également Le token CSRF
        # Traitement du formulaire
        pass
    return render_template('login.html', form=form)
```

html

```
<!-- Dans Le template -->
<form method="POST">
    {{ form.hidden_tag() }} <!-- Inclut Le token CSRF -->
    {{ form.email.label }} {{ form.email }}
    {{ form.password.label }} {{ form.password }}
    {{ form.submit }}
</form>
```

- **Clickjacking:**

- En-têtes HTTP de sécurité (X-Frame-Options)
- Protection contre l'inclusion du site dans des iframes

- **Contrôle d'accès cassé:**

- Vérification rigoureuse des autorisations à chaque requête protégée
- Utilisation de décorateurs pour appliquer les contrôles systématiquement

python

```
@tickets_bp.route('/<int:ticket_id>')
@login_required
def detail(ticket_id):
    ticket = Ticket.query.get_or_404(ticket_id)

    # Vérifier que L'utilisateur est bien Le propriétaire du billet
    if ticket.user_id != current_user.id:
        flash('Vous n\'êtes pas autorisé à accéder à ce billet.', 'danger')
        return redirect(url_for('tickets.index'))

    return render_template('tickets/detail.html', ticket=ticket)
```

- **Sécurisation des communications:**

- Utilisation de HTTPS avec TLS 1.3
- Configuration stricte des en-têtes de sécurité
- Redirection automatique HTTP vers HTTPS

```
python
```

```
@app.after_request
def add_security_headers(response):
    response.headers['Content-Security-Policy'] = "default-src 'self'; script-src 'self'; style"
    response.headers['X-Content-Type-Options'] = 'nosniff'
    response.headers['X-Frame-Options'] = 'SAMEORIGIN'
    response.headers['X-XSS-Protection'] = '1; mode=block'
    response.headers['Strict-Transport-Security'] = 'max-age=31536000; includeSubDomains'
    return response
```

5. Sécurité de l'infrastructure

Éléments à sécuriser

- Serveur d'application
- Base de données
- Système de fichiers

Méthodologie

- **Sécurisation du serveur:**
 - Mises à jour régulières des composants système
 - Configuration minimal exposure (principe du moindre privilège)
 - Utilisation d'un reverse proxy (Nginx) devant le serveur d'application

```

nginx

# Configuration Nginx
server {
    listen 80;
    server_name exemple.com;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    server_name exemple.com;

    ssl_certificate /path/to/certificate.crt;
    ssl_certificate_key /path/to/private.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384;ECDHE-RSA-AES256-GCM-SHA384;ECDHE-ECDSA-CHACHA20-POLY1305@TLS1.3';

    location / {
        proxy_pass http://127.0.0.1:5000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

```

- **Sécurisation de la base de données:**

- Accès restreint via credentials sécurisés et réseau isolé
- Sauvegarde régulière avec chiffrement
- Séparation des privilèges (compte lecture seule pour les requêtes standards)

sql

```

-- Création d'un utilisateur avec droits limités
CREATE USER 'app_READONLY'@'localhost' IDENTIFIED BY 'strong_password';
GRANT SELECT ON jo_etickets.* TO 'app_READONLY'@'localhost';

```

- **Journalisation et surveillance:**

- Logs d'activité détaillés
- Alertes sur comportements suspects (nombreuses tentatives de connexion, scans de billets multiples)

- Audit régulier des journaux d'activité

python

```
# Configuration de Logging
import logging
from logging.handlers import RotatingFileHandler

# Configuration du Logger
def setup_logging(app):
    ... if not os.path.exists('logs'):
        ... os.mkdir('logs')

    ... file_handler = RotatingFileHandler('logs/jo_etickets.log', maxBytes=10240, backupCount=10)
    ... file_handler.setFormatter(logging.Formatter(
        '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]')
    ))
    ... file_handler.setLevel(logging.INFO)

    ... app.logger.addHandler(file_handler)
    ... app.logger.setLevel(logging.INFO)
    ... app.logger.info('JO E-Tickets startup')
```

6. Traçabilité et détection des fraudes

Éléments à sécuriser

- Processus d'achat
- Validation des billets
- Activités administratives

Méthodologie

- **Journalisation extensive:**

- Enregistrement de toutes les actions sensibles (création de compte, achat, validation de billet)
- Horodatage précis
- Conservation des adresses IP et des identifiants de session

```
python
```

```
def log_security_event(event_type, user_id, details, ip_address):  
    """  
        Enregistre un événement de sécurité dans le journal.  
    """  
  
    log_entry = SecurityLog(  
        event_type=event_type,  
        user_id=user_id,  
        details=json.dumps(details),  
        ip_address=ip_address,  
        timestamp=datetime.utcnow()  
    )  
  
    db.session.add(log_entry)  
    db.session.commit()
```

- **Système de détection des anomalies:**

- Surveillance des tentatives d'achat multiples
- Détection des comportements suspects (nombreuses tentatives de validation d'un même billet)
- Alertes automatiques sur les schémas inhabituels

```
python
```

```
def check_suspicious_activity(user_id, action_type):
    """
    Vérifie si l'activité récente d'un utilisateur est suspecte.
    """

    time_window = datetime.utcnow() - timedelta(minutes=10)

    # Compter les actions récentes
    recent_actions = SecurityLog.query.filter(
        SecurityLog.user_id == user_id,
        SecurityLog.event_type == action_type,
        SecurityLog.timestamp > time_window
    ).count()

    # Définir des seuils selon le type d'action
    thresholds = {
        'login_attempt': 5,
        'ticket_validation': 3,
        'payment_attempt': 3
    }

    if action_type in thresholds and recent_actions >= thresholds[action_type]:
        # Enregistrer l'alerte et notifier les administrateurs
        alert = SecurityAlert(
            user_id=user_id,
            alert_type=f"excessive_{action_type}",
            details=f"{recent_actions} tentatives en moins de 10 minutes",
            timestamp=datetime.utcnow()
        )
        db.session.add(alert)
        db.session.commit()

        # Envoi d'alerte aux administrateurs
        send_admin_alert(alert)

    return True # Activité suspecte détectée

return False # Activité normale
```

- **Audit des actions administratives:**

- Journal détaillé de toutes les actions des administrateurs
- Validation à deux niveaux pour les opérations sensibles (suppression d'offres, modification de billets)
- Notification des actions sensibles

```
python
```

```
@admin_bp.route('/offers/<int:offer_id>/delete', methods=['POST'])
@login_required
def delete_offer(offer_id):
    """
    Suppression d'une offre.
    """

    # Vérifier que l'utilisateur est administrateur
    if current_user.role != 'administrateur':
        log_security_event('unauthorized_admin_access', current_user.id,
                            {'action': 'delete_offer', 'offer_id': offer_id},
                            request.remote_addr)
        flash('Accès refusé. Vous devez être administrateur.', 'danger')
        return redirect(url_for('main.index'))

    offer = Offer.query.get_or_404(offer_id)

    # Vérifier si l'offre a des billets associés
    if offer.tickets:
        flash(f'Impossible de supprimer l\'offre "{offer.titre}" car elle a des billets associés')
        return redirect(url_for('admin.offers'))

    # Journaliser l'action avant suppression
    log_security_event('admin_delete_offer', current_user.id,
                        {'offer_id': offer_id, 'offer_title': offer.titre},
                        request.remote_addr)

    db.session.delete(offer)
    db.session.commit()

    flash(f'Offre "{offer.titre}" supprimée avec succès.', 'success')
    return redirect(url_for('admin.offers'))
```

Conclusion

La stratégie de sécurité du projet JO E-Tickets repose sur plusieurs piliers fondamentaux:

1. Une **approche multi-couches** qui protège chaque niveau du système
2. Un **système unique de double clé** pour garantir l'authenticité des billets
3. Le respect strict des **normes de protection des données personnelles**
4. Une **surveillance continue** pour détecter et prévenir les comportements frauduleux

Cette stratégie complète assure que même si un aspect de la sécurité était compromis, les autres mécanismes continueraient à protéger l'intégrité du système, offrant ainsi une protection robuste contre

les tentatives de fraude ou d'accès non autorisé, essentielle pour un événement de l'envergure des Jeux Olympiques.

La sécurité n'est pas un état mais un processus continu. Le système a été conçu pour être évolutif et permettre l'intégration de nouvelles mesures de sécurité au fur et à mesure que des menaces émergent ou que les technologies évoluent. Des audits de sécurité réguliers et des tests de pénétration sont prévus pour s'assurer que le système reste robuste face aux menaces actuelles et futures.