

12. Risques applicatifs

10 décembre 2025

Développement web il3

Risques applicatifs des app web

HE-Arc (DGR) 2025

Risque

- Faille ou bug permettant d'altérer le fonctionnement
- Un attaquant pourra :
 - Modifier le fonctionnement
 - Accéder ou modifier les données
- Présence possible à tous les niveaux d'un système
 - Application
 - Serveur et Client
 - OS
 - SGBD, ...
- Responsabilité des développeurs :
 - OS, serveurs, langages : patches rapidement disponibles
 - nos applications : **c'est nous qui en sommes responsables**

OWASP¹

- Open Web Application Security Project
- Fondation pour améliorer la sécurité des webapps
- Fondée en 2004, internationale, sans but lucratif
- Référence principale dans le domaine

1. <https://owasp.org/>

- Propose :
 - Top 10 (web, mobile ², API ³, LLM ⁴) tous les 4 ans : Méthode ⁵, CVSS ⁶, CWE ⁷
 - Grande communauté d'experts
 - Formation, documentation et ressources
 - Outils d'audit, de tests et de formation (ex : Juice Shop ⁸)
 - Cheat Sheets ⁹ (yc pour CI/CD, Ajax, Laravel, Django,...;)

Top 10 ¹⁰ OWASP 2025 (fr ¹¹ - historique ¹²)

1. Contrôle d'accès défaillants
 2. Mauvaise configuration de sécurité
 3. Vulnérabilités des dépendances
 4. Défaillances cryptographiques
 5. Injections
 6. Conception non sécurisée
 7. Authentification de mauvaise qualité
 8. Manque d'intégrité des données et du logiciel
 9. Carences des systèmes d'alerte et de journalisation
 10. Mauvaise gestion des conditions exceptionnelles
- Non exhaustif : ex. : risques liés à Node JS ¹³

Injection de code

- Données mal validées : possibilité d'exécuter du code
- Passées par requêtes :
 - formulaires
 - URL
 - ...
- Type de code injectable : TOUS !
 - HTML

-
2. <https://owasp.org/www-project-mobile-top-10/>
 3. <https://owasp.org/www-project-api-security/>
 4. <https://genai.owasp.org/resource/owasp-top-10-for-l1m-applications-2025/>
 5. <https://owasp.org/Top10/#methodology>
 6. <https://www.first.org/cvss/calculator/3.0>
 7. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html
 8. <https://owasp.org/www-project-juice-shop/>
 9. <https://cheatsheetseries.owasp.org/>
 10. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
 11. <https://owasp.org/Top10/fr/>
 12. <https://www.hahwul.com/cullinan/history-of-owasp-top-10/>
 13. https://cheatsheetseries.owasp.org/cheatsheets/NPM_Security_Cheat_Sheet.html

- SQL
- Javascript
- ...

Injections SQL

- Modifier les requêtes envoyées au SGBD
- Obtention d'un résultat non prévu par le développeur
- Deviner la structure du code pour l'exploiter
- SQL est puissant : UNION, INTO DUMPFILE, ...

Exemples¹⁴

```
SELECT titre, num FROM livres WHERE num=2 UNION  
SELECT login, password FROM user INTO DUMPFILE 'www/exploit.txt'
```

Eviter les injections SQL

- N'accepter que des caractères valides
- A défaut, neutraliser les caractères dangereux
- Utiliser les entités HTML
- Vérifications strictes dans le code
- Eviter les noms prévisibles pour une appli critique

Cross Site Scripting (XSS)

- Injection de code (html et script)

14. https://fr.wikipedia.org/wiki/Injection_SQL



A High Level View of a typical XSS Attack

- Exécution par le navigateur du client

Cross Site Scripting (XSS)

- Enjeux : tout ce qui est possible en JS
 - Redirection
 - Lecture de cookies (session, ...)
 - Envoi d'info à un autre serveur
 - Modification du contenu de la page
 - ...
- Souvent utilisé pour transmettre le cookie de session

```

```

3 types de XSS

- Reflected XSS
 - Affichage d'une partie de la requête (recherche, erreur, ...)
- Stored XSS
 - Stockage dans la BDD et affichage (= exécution) par plusieurs clients
- DOM based XSS

- Exécutée lors de la modification du DOM (Exemple ¹⁵)

Cross Site Request Forgery (CSRF - Sea Surf)

- **Principe :**
 - Faire réaliser à quelqu'un une action à son insu, avec ses propres infos d'authentification (credentials)
 - Envoi par mail ou post forum de liens ou images
 - Les URL correspondent à actions (vote, suppression, ...)

Exemple ¹⁶ (SOP, CORS)

Phishing

- Site sosie d'un site officiel :
 1. L'utilisateur saisit ses données...
 2. ... l'attaquant les récupère...
 3. ... et les utilise sur le site officiel
- Difficile à contrer pour le développeur
- L'utilisateur doit être prudent
- Bien lire les URLs et le GUI du navigateur pas toujours suffisant
- Ne pas utiliser de lien dont on n'est pas sûr de la source (Homograph Attack ¹⁷, Homoglyphes ¹⁸, Unicode Spoofing ¹⁹)

Risques non liés à l'application

- IoT : souvent mal sécurisé (shodan.io ²⁰)
- DoS
- Spoofing (IP, DNS, ARP)
- Buffer Overflows (surtout en C)
- Trojans, backdoors
- Usurpation de mots de passe : dictionnaire, force brute
- **SOCIAL ENGINEERING !!!**

15. https://www.owasp.org/index.php/DOM-Based_XSS

16. <https://www.owasp.org/index.php/CSRF>

17. <https://www.xudongz.com/blog/2017/idn-phishing/>

18. https://github.com/codebox/homoglyph/blob/master/raw_data/chars.txt

19. <https://onlineunicodetools.com/spoof-unicode-text>

20. <https://www.shodan.io/>

Authentification

- **Identification** : annoncer qui on est
 - **Authentification** : prouver qu'on est la personne qu'on prétend être :
 1. Avec quelque chose que l'on **sait** (PIN, mot de passe)
 2. Avec quelque chose que l'on **possède** (téléphone, token, ...)
 3. Avec quelque chose que l'on **est** (biométrie)
 - La sécurité augmente si on combine ces facteurs
 - Important de prendre en compte l'utilisabilité

Top 500 passwords cloud



FIGURE 1 – top 500 passwords cloud

Mots de passe

- 30% of users have a password from the top 10'000 (source ²¹)
 - Our passwords habits revealed ²²
 - xkcd's password strength ²³
 - 2017 : NIST 800-63-3 ²⁴ suivi par la NCSC ²⁵
 - Mots de passe longs plutôt qu'avec des caractères spéciaux
 - Ne forcer le changement qu'en cas de nécessité
 - Autoriser et accompagner l'utilisation de password managers
 - Utiliser la 2FA

21. <https://mojoauth.com/blog/why-are-businesses-still-using-passwords/>

22. <https://visual.ly/our-password-habits-revealed>

23. <https://xkcd.com/936/>

24. <https://pages.nist.gov/800-63-3/>

25. <https://www.ncsc.gov.uk/guidance>

• • • • • • • •

- Plusieurs tentatives pour s'en affranchir :
 - Microsoft ²⁶, passwordless ²⁷ authentication
 - 2022 : Passkeys : JS API WebAuthN ²⁸ + CTAP/U2F ²⁹

Passkeys ³⁰

- Paire de clés asymétriques au lieu d'un mot de passe
- Initiative de l'alliance FIDO ³¹
- Fin 2022 : intégrée à Android, iOS, win11 et MacOS
- Résolution de challenges : pas d'info sensible sur le réseau
- 3 acteurs :
 - User Agent : Humain / Navigateur
 - Relying Party : Serveur (service auquel on veut s'authentifier)
 - Authenticator : Clef USB / Smartphone / OS + biométrie
- Communication :
 - User Agent <=> Authenticator : CTAP / U2F
 - User Agent <=> Relying Party : API JS WebAuthn ³²
- Disponible sur Switch Edu-ID : **Testez !**

Passkeys : Acteurs ³³

Passkeys : Enregistrement ³⁴

Passkeys : Authentification ³⁵

Collecte d'information

- Toute information est bonne pour l'attaquant
 - Messages d'erreur

26. <https://www.microsoft.com/security/blog/2021/09/15/the-passwordless-future-is-here-for-your-microsoft-account/>

27. <https://hacks.mozilla.org/2014/10/passwordless-authentication-secure-simple-and-fast-to-deploy/>

28. <https://en.wikipedia.org/wiki/WebAuthn>

29. <https://proton.me/blog/fr/universal-2nd-factor-u2f>

30. <https://medium.com/webauthnworks/introduction-to-webauthn-api-5fd1fb46c285>

31. <https://fidoalliance.org/members/>

32. <https://webauthn.guide/>

33. <https://auth0.com/blog/introduction-to-web-authentication/>

34. <https://www.freecodecamp.org/news/intro-to-webauthn/>

35. <https://www.freecodecamp.org/news/intro-to-webauthn/>



FIGURE 2 – Architecture



FIGURE 3 – Reg

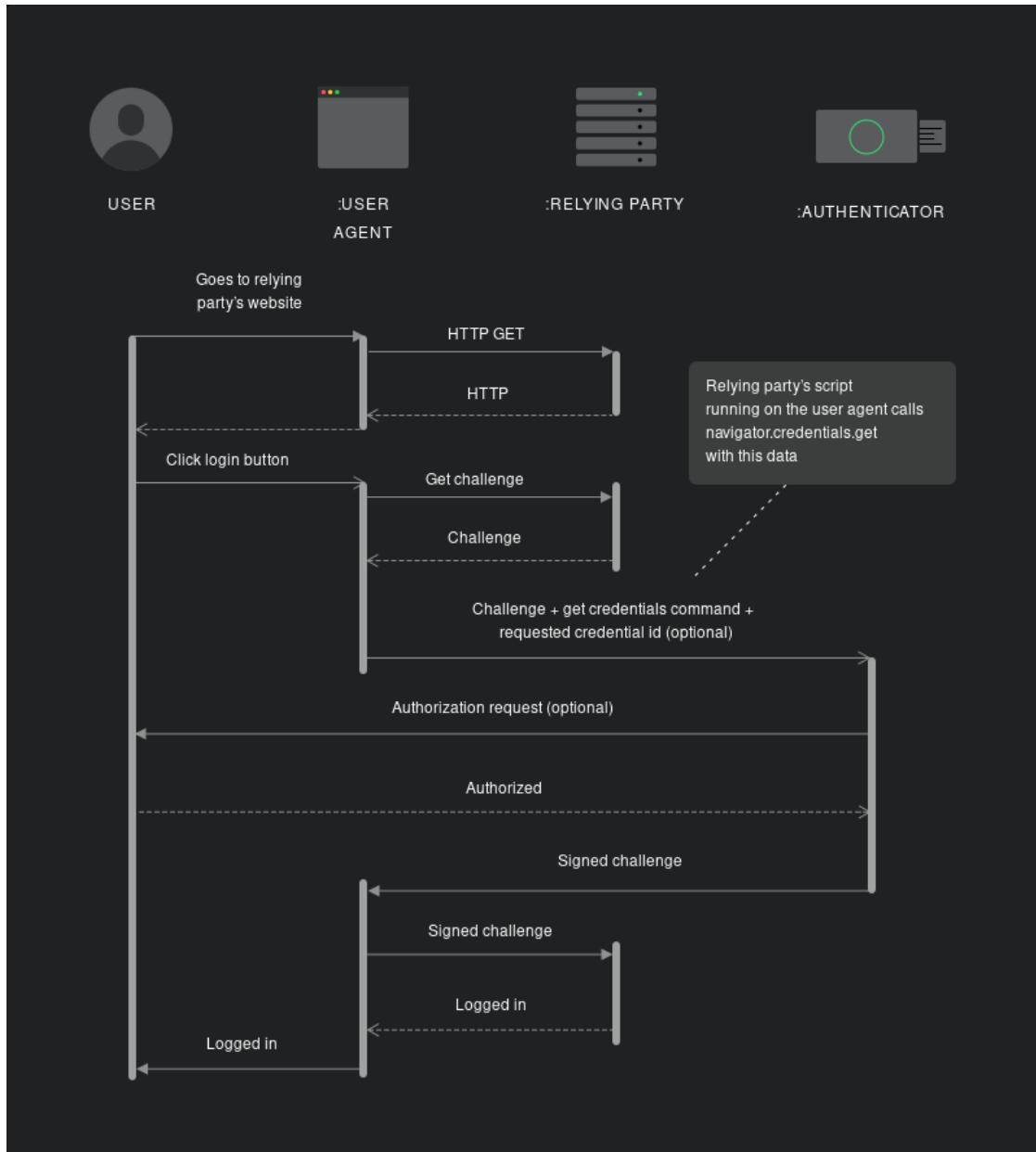


FIGURE 4 – Auth

- Configuration OS serveur
- Configuration serveurs (http, sql, php, ...)
- Identifiants et commentaires dans sources -au cas où-
- SOCIAL ENGINEERING !
- Le développeur doit laisser filter un minimum d'info !
- Utilisée aussi par les “white hats” (ethical hackers) :
 - Cowrie Honeypot ³⁶ (visualisation des attaques en 24h)
 - Autres cartes ³⁷ de menaces et attaques

Bonnes pratiques

- Configuration stricte du serveur
- Valider toutes les entrées (formulaires, requêtes HTTP)
- Filtrage/encodage de toutes les entrées en entités HTML
- Ne jamais afficher directement une saisie de formulaire
 - Ni aucune donnée transmise par HTTP avant de l'avoir filtrée !
- Tester ses formulaires avec des expressions à risques
- Contrôler le maximum de paramètres (même si redondant) :
 - Session, IP, user agent, proxy, ...
- Suites et logiciels de test
- Utiliser un framework
 - ces bonnes pratiques sont déjà implémentées

Laravel, Django et le top 10 OWASP

OWASP 21	Laravel	Django
A01 Accès	Role based AC	Décorateur @login_required, django.contrib.auth
A02 Crypto	Passwords : Bcrypt, EncryptCookies, Crypt	make_password() , gestion SECRET_KEY
A03 Injection	ORM Eloquent, protection injection SQL	ORM Django, RawSQL() avec placeholders pour requêtes brutes
A04 Conception	Starter Kits (Breeze, Fortify, Jetstream)	Architecture secure by default, check -deploy
A05 Config	APP_DEBUG à FALSE, permissions fichiers (775/664)	DEBUG = False obligatoire en production, middleware de sécurité activé par défaut

36. <https://hackertarget.com/cowrie-honeypot-analysis-24hrs/>

37. <https://www.google.com/search?q=ipviking>

OWASP 21	Laravel	Django
A06 Dépendances	Enlightn Security Checker (scan dépendances)	tiers : safety, Bandit, ... pour dépendances
A07 Auth	Sanctum et Passport auth. API, rate limiting	Validateurs passwords, django-axes contre brute-force
A08 Intégrité	Protection contre mass assignment via \$fillable et \$guarded, éviter forceFill() sur données non validées	Protection automatique contre mass assignment via ModelForm avec Meta.fields (approche liste blanche)
A09 Logs	logs intégré, monitoring en temps réel	Logging configurable, outils comme django-axes
A10 SSRF	Validation stricte des URLs	Validation des URLs externes, sécurité en-têtes HTTP
CSRF Protection	VerifyCsrfToken, (<code>csrf?</code>) pour forms, token automatique pour AJAX via Axios	CsrfViewMiddleware, <code>{% csrf_token %}</code> pour forms
XSS Protection	Échappement automatique via <code>\{\ }</code> dans Blade	Échappement automatique dans templates, éviter <code>mark_safe()</code> et filtre safe

(Généré par perplexity.ai)

Références

- Référence
 - OWASP ³⁸, webinar fr 2016 ³⁹
 - WebAuthn : w3c ⁴⁰, MDN ⁴¹
- Exemples, explications
 - Présentation XSS et CSRF ⁴² en français
 - Protection CSRF ⁴³ en français
- Utilitaires, tutos, exercices
 - Juice Shop ⁴⁴
 - Web Goat ⁴⁵
 - Google-Gruyere ⁴⁶

38. <https://www.owasp.org/>

39. <https://www.youtube.com/watch?v=pHI2zitLph8>

40. <https://www.w3.org/TR/webauthn/>

41. https://developer.mozilla.org/en-US/docs/Web/API/Web.Authentication_API

42. <https://www.journaldunet.com/solutions/dsi/1209139-comment-eviter-les-failles-cross-site-scripting-xss/>

43. <https://www.apprendre-php.com/tutoriels/tutoriel-39-introduction-aux-cross-site-request-forgeries-ou-sea-surf.html>

44. <https://owasp.org/www-project-juice-shop/>

45. <https://www.owasp.org/index.php/Webgoat>

46. <https://google-gruyere.appspot.com/>

- Passkeys developer Cheat Sheet⁴⁷

Sources

47. <https://www.corbado.com/blog/passkeys-cheat-sheet>