

12. Risques applicatifs

3 décembre 2025

Développement web il3

Risques applicatifs des app web

HE-Arc (DGR) 2024

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**

OWASP ¹

- 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 et mobile ²) tous les 4 ans : Méthode ³, CVSS ⁴, CWE ⁵
 - Grande communauté d'experts
 - Formation, documentation et ressources
 - Outils d'audit, de tests et de formation (ex : Juice Shop ⁶)
 - Cheat Sheets ⁷ (yc pour CICD, Ajax, Laravel, Django,...;)

Top 10 ⁸ OWASP 2021 (fr ⁹ - historique ¹⁰)

1. Contrôle d'accès défaillants
 2. Défaillances cryptographiques
 3. Injections
 4. Conception non sécurisée
 5. Mauvaise configuration de sécurité
 6. Composants vulnérables et obsolètes
 7. Identification & Authentification de mauvaise qualité
 8. Manque d'intégrité des données et du logiciel
 9. Carences des systèmes de contrôle et de journalisation
 10. Falsification de requêtes côté serveur
- Non exhaustif : ex. : risques liés à Node JS ¹¹

Injection de code

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

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

— ...

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, ...

Exemples¹²

```
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)

12. 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 ¹³)

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 ¹⁴ (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 ¹⁵, Homoglyphes ¹⁶, Unicode Spoofing ¹⁷)

Risques non liés à l'application

- IoT : souvent mal sécurisé (shodan.io ¹⁸)
- DoS
- Spoofing (IP, DNS, ARP)
- Buffer Overflows (surtout en C)
- Trojans, backdoors
- Usurpation de mots de passe : dictionnaire, force brute
- **SOCIAL ENGINEERING !!!**

13. https://www.owasp.org/index.php/DOM_Based_XSS

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

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

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

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

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

Authentication

- **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 ¹⁹)
- Our passwords habits revealed ²⁰
- xkcd's password strength ²¹
- 2017 : NIST 800-63-3 ²² suivi par la NCSC ²³
 - 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

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

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

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

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

23. <https://www.ncsc.gov.uk/guidance/password-guidance-simplifying-your-approach>

- Plusieurs tentatives pour s'en affranchir :
 - Microsoft²⁴, passwordless²⁵ authentication
 - 2022 : Passkeys : JS API WebAuthN²⁶ + CTAP/U2F²⁷

Passkeys²⁸

- Paire de clés asymétriques au lieu d'un mot de passe
- Initiative de l'alliance FIDO²⁹
- 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 WebAuthn³⁰
- Disponible sur Switch Edu-ID : **Testez!**

Passkeys : Acteurs³¹

Passkeys : Enregistrement³²

Passkeys : Authentification³³

Collecte d'information

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

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

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

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

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

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

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

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

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

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

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



FIGURE 2 – Architecture

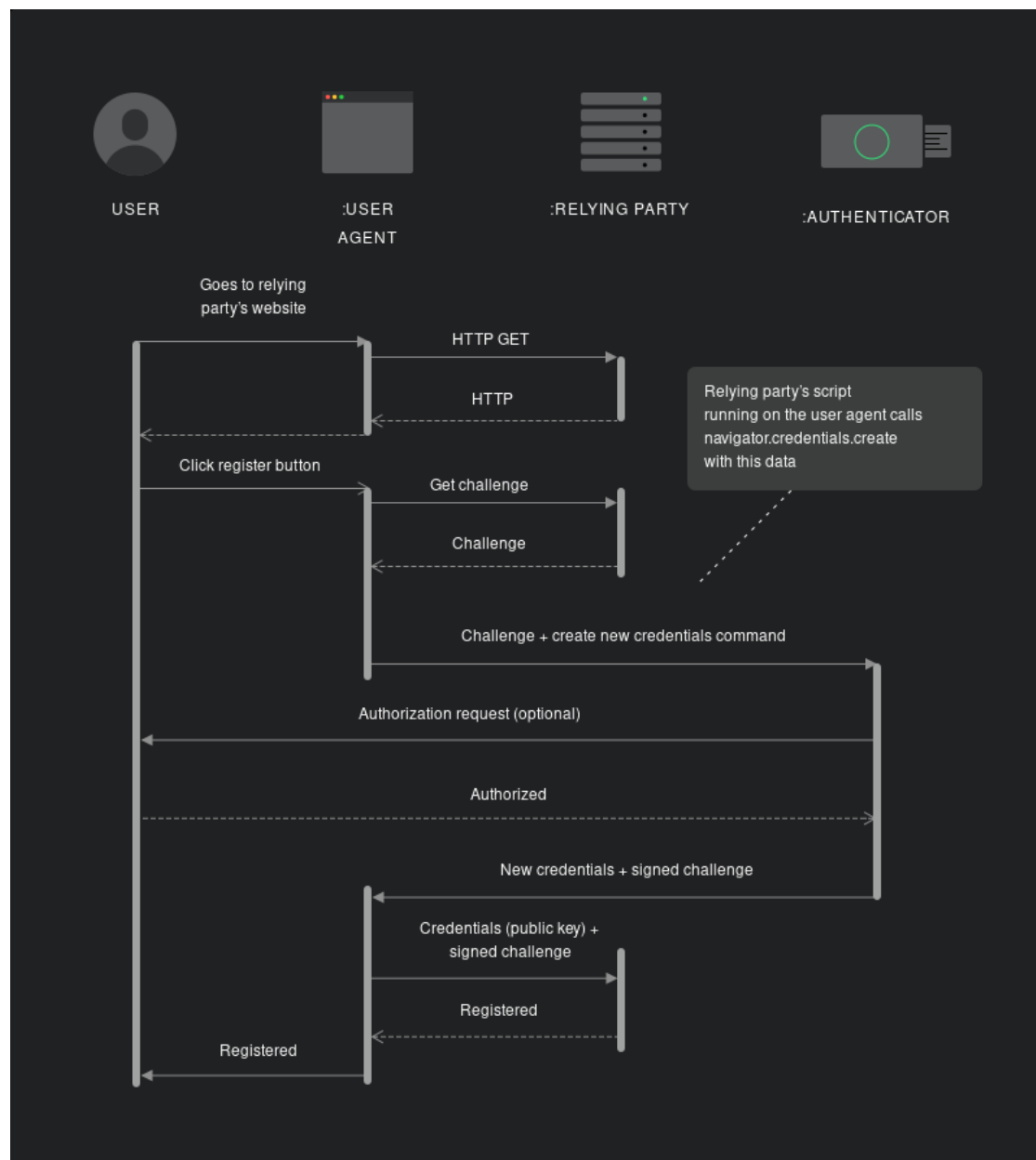


FIGURE 3 – Reg

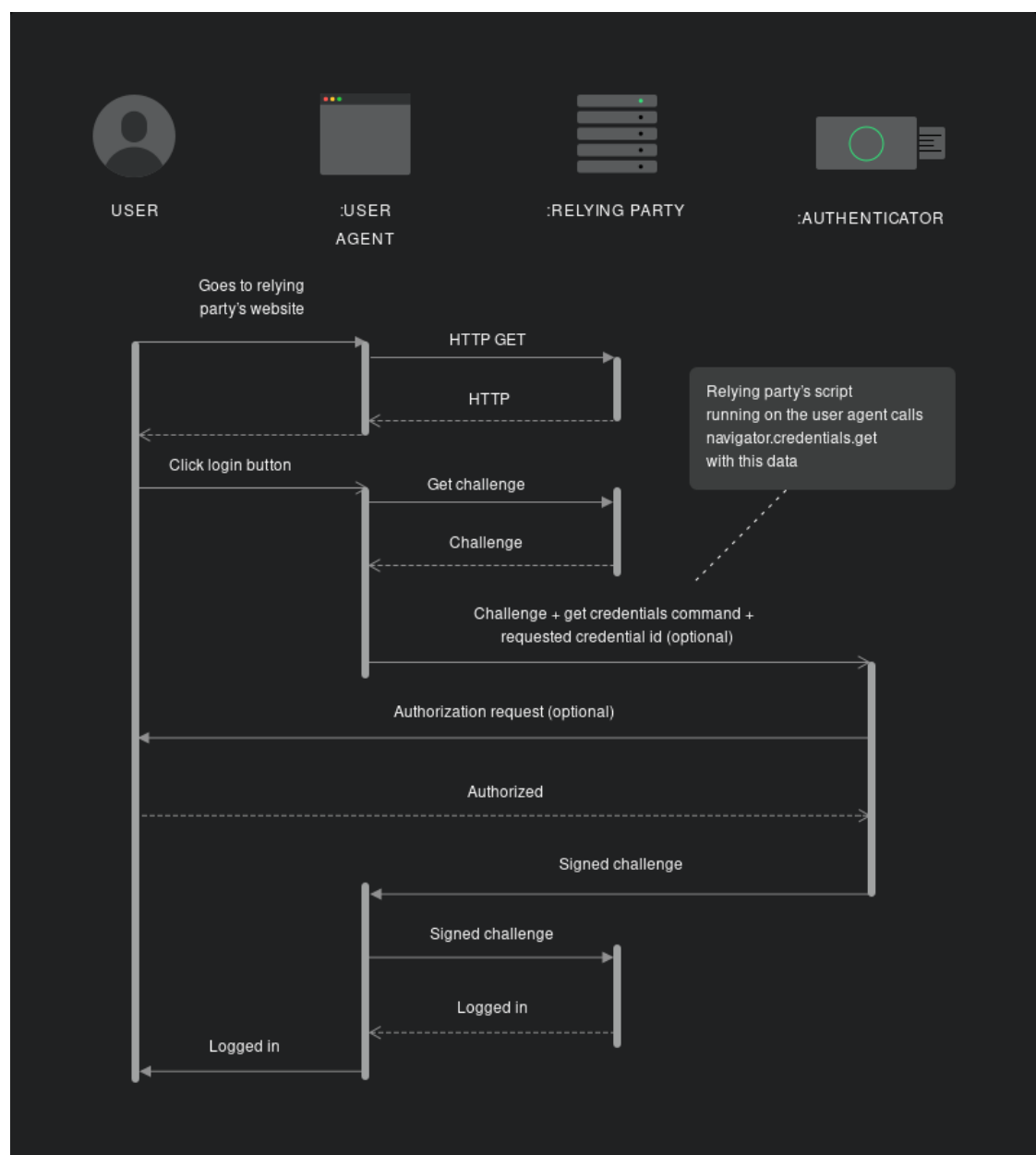


FIGURE 4 – Auth

- 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 ³⁴ (visualisation des attaques en 24h)
 - Autres cartes ³⁵ 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 2021 | Laravel | Django |
|-----------------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| A01 : Broken Access Control | Middleware d'autorisation, Gates et Politiques pour contrôle d'accès basé sur les rôles | Décorateur (login_required?), système de permissions intégré avec django.contrib.auth |

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

35. <https://www.google.com/search?q=ipvikings>

| OWASP 2021 | Laravel | Django |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| A02 : Cryptographic Failures | Hachage automatique des mots de passe avec Bcrypt, chiffrement des cookies via middleware EncryptCookies, fonction Crypt pour données sensibles | Fonction make_password() pour hachage sécurisé, gestion de SECRET_KEY avec rotation recommandée |
| A03 : Injection | ORM Eloquent avec requêtes paramétrées par défaut, protection contre SQL injection si utilisation correcte des bindings | ORM Django avec requêtes paramétrées automatiques, fonction RawSQL() avec placeholders pour requêtes brutes |
| A04 : Insecure Design | Kits de démarrage sécurisés (Breeze, Fortify, Jetstream) avec fonctionnalités d'authentification robustes | Architecture secure by default, documentation de sécurité complète et commande check –deploy |
| A05 : Security Misconfiguration | Variable APP_DEBUG à désactiver en production, validation des permissions fichiers (775/664) | DEBUG = False obligatoire en production, middleware de sécurité activé par défaut |
| A06 : Vulnerable Components | Outil Enlightn Security Checker pour scanner les dépendances vulnérables | Outils tiers comme safety et Bandit pour détecter vulnérabilités dans dépendances |
| A07 : Authentication Failures | Packages Laravel Sanctum et Passport pour authentification API, limitation de tentatives via rate limiting | Validateurs de mots de passe configurables, packages django-axes pour protection contre brute-force |

| OWASP 2021 | Laravel | Django |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| A08 : Software and Data Integrity Failures | 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 : Logging & Monitoring Failures | Système de logs intégré, packages tiers pour monitoring en temps réel disponibles | Logging configurable, outils comme django-axes pour traçabilité des tentatives d'authentification |
| A10 : Server-Side Request Forgery (SSRF) | Validation stricte des URLs, éviter requêtes basées sur input utilisateur non validé | Validation des URLs externes, middleware de sécurité pour en-têtes HTTP |
| Bonus : CSRF Protection | Middleware VerifyCsrfToken activé par défaut, directive Blade (csrf?) pour formulaires, token automatique pour AJAX via Axios | Middleware Csrf-ViewMiddleware activé par défaut, tag template {% csrf_token %} pour formulaires |
| Bonus : XSS Protection | Échappement automatique via {{ }} dans Blade, attention avec {!! !!} qui désactive la protection | Échappement automatique de toutes les variables dans templates, éviter mark_safe() et filtre safe |

(Généré par perplexity.ai)

Références

- Référence
- OWASP ³⁶, webinar fr 2016 ³⁷

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

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

- WebAuthn : w3c ³⁸, MDN ³⁹
- Exemples, explications
 - Présentation XSS et CSRF ⁴⁰ en français
 - Protection CSRF ⁴¹ en français
- Utilitaires, tutos, exercices
 - Juice Shop ⁴²
 - Web Goat ⁴³
 - Google-Gruyere ⁴⁴
- Passkeys developer Cheat Sheet ⁴⁵

Sources

-
- 38. <https://www.w3.org/TR/webauthn/>
 - 39. https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API
 - 40. <https://www.journaldunet.com/solutions/dsi/1209139-comment-eviter-les-failles-cross-site-scripting-xss/>
 - 41. <https://www.apprendre-php.com/tutoriels/tutoriel-39-introduction-aux-cross-site-request-forgeries-ou-sea-surf.html>
 - 42. <https://owasp.org/www-project-juice-shop/>
 - 43. <https://www.owasp.org/index.php/Webgoat>
 - 44. <https://google-gruyere.appspot.com/>
 - 45. <https://www.corbado.com/blog/passkeys-cheat-sheet>