

12. Risques applicatifs

2 décembre 2024

Développement web il3

Risques applicatifs des app web

HE-Arc (DGR) 2022

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

¹<https://owasp.org/>

- Fondation pour améliorer la sécurité des webapps
- Fondée en 2004, internationale, sans but lucratif
- Référence principale dans le domaine
- Propose :
 - Top 10 (web et mobile²) : Méthode³, CVSS⁴, CWE⁵
 - Grande communauté d'experts
 - Formation, documentation et ressources
 - Outils d'audit, de tests et de formation

Top 10⁶ OWASP 2021 (fr⁷ - historique⁸)

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 JS⁹

Injection de code

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

²<https://owasp.org/www-project-mobile-top-10/>

³<https://owasp.org/Top10/#methodology>

⁴<https://www.first.org/cvss/calculator/3.0>

⁵https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

⁶https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

⁷<https://owasp.org/Top10/fr/>

⁸<https://www.hahwul.com/cullinan/history-of-owasp-top-10/>

⁹https://cheatsheetseries.owasp.org/cheatsheets/NPM_Security_Cheat_Sheet.html

- 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, ...

Exemples¹⁰

```
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)

¹⁰https://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 (Exemple¹¹)

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, ...)

Exemple¹² (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 Attack¹³, Homoglyphes¹⁴, Unicode Spoofing¹⁵)

Risques non liés à l'application

- IoT : souvent mal sécurisé (shodan.io¹⁶)
- DoS
- Spoofing (IP, DNS, ARP)

¹¹https://www.owasp.org/index.php/DOM_Based_XSS

¹²<https://www.owasp.org/index.php/CSRF>

¹³<https://www.xudongz.com/blog/2017/idn-phishing/>

¹⁴https://github.com/codebox/homoglyph/blob/master/raw_data/char_codes.txt

¹⁵<https://onlineunicodetools.com/spoof-unicode-text>

¹⁶<https://www.shodan.io/>

- 2017 : NIST 800-63-3²⁰ suivi par la NCSC²¹
 - 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
- Plusieurs tentatives pour s’en affranchir :
 - Microsoft²², passwordless²³ authentication
 - 2022 : Passkeys : JS API WebAuthN²⁴ + CTAP/U2F²⁵

Passkeys²⁶

- Paire de clés asymétriques au lieu d’un mot de passe
- Initiative de l’alliance FIDO²⁷
- 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 WebAuthn²⁸



Figure 2: Architecture

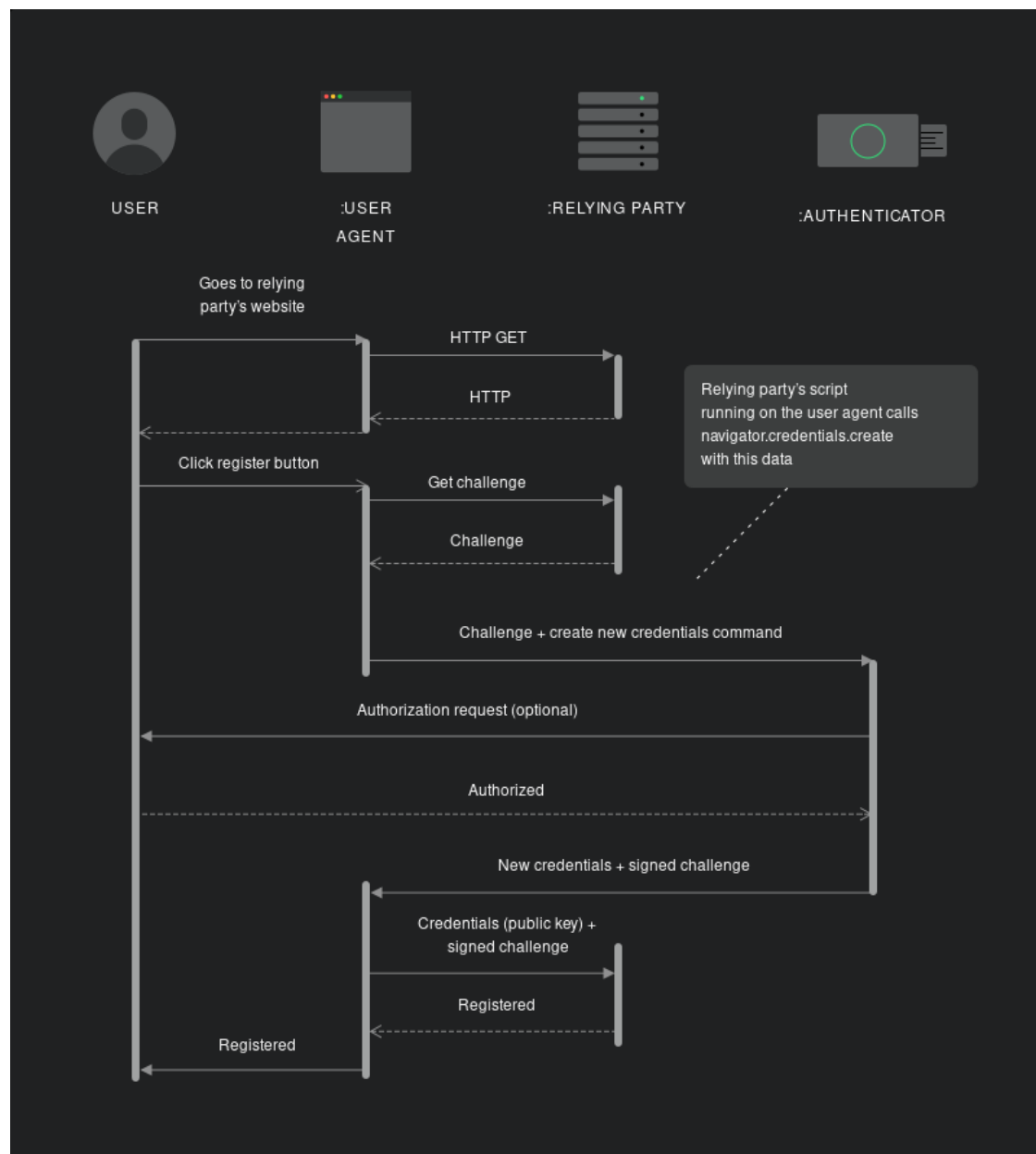


Figure 3: Reg

Passkeys : Acteurs²⁹

Passkeys : Enregistrement³⁰

Passkeys : Authentification³¹

Collecte d'information

- Toute information est bonne pour l'attaquant
 - Messages d'erreur
 - 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) : Honeypots³²

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) :

²⁰<https://nakedsecurity.sophos.com/2016/08/18/nists-new-password-rules-what-you-need-to-know/>

²¹<https://www.ncsc.gov.uk/guidance/password-guidance-simplifying-your-approach>

²²<https://www.microsoft.com/security/blog/2021/09/15/the-passwordless-future-is-here-for-your-microsoft-account/>

²³<https://hacks.mozilla.org/2014/10/passwordless-authentication-secure-simple-and-fast-to-deploy/>

²⁴<https://en.wikipedia.org/wiki/WebAuthn>

²⁵<https://u2f-key.tech/fr/>

²⁶<https://medium.com/webauthnworks/introduction-to-webauthn-api-5fd1fb46c285>

²⁷<https://fidoalliance.org/members/>

²⁸<https://webauthn.guide/>

²⁹<https://auth0.com/blog/introduction-to-web-authentication/>

³⁰<https://www.freecodecamp.org/news/intro-to-webauthn/>

³¹<https://www.freecodecamp.org/news/intro-to-webauthn/>

³²<https://hackertarget.com/cowrie-honeypot-analysis-24hrs/>

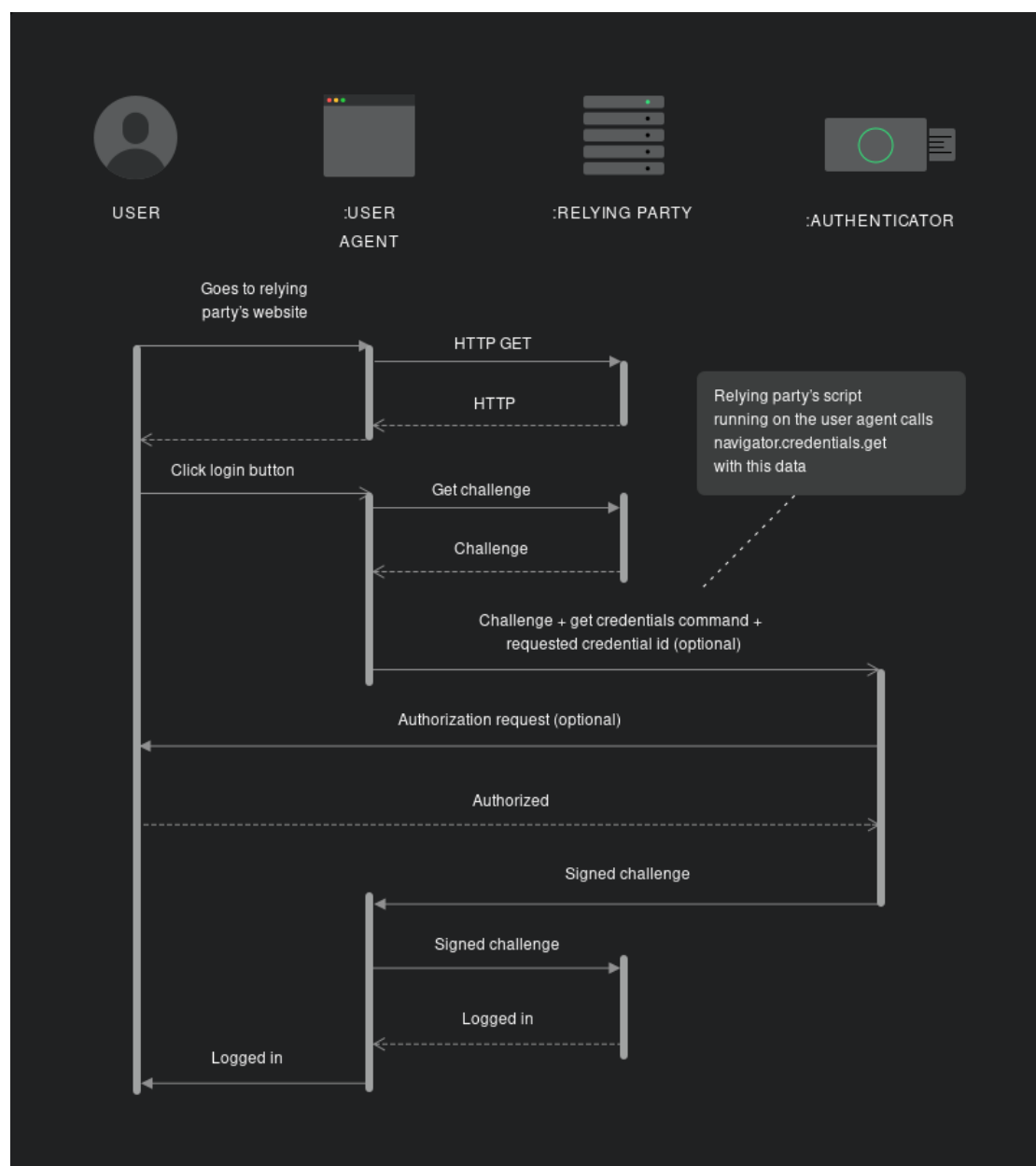


Figure 4: Auth

- Session, IP, user agent, proxy, ...
- Utiliser un framework
 - ces bonnes pratiques sont déjà implémentées
- Suites et logiciels de test

Références

- Référence
 - OWASP³³, webinar fr 2016³⁴
 - WebAuthn : w3c³⁵, MDN³⁶
- Exemples, explications
 - Présentation XSS et CSRF³⁷ en français
 - Protection CSRF³⁸ en français
- Utilitaires, tutos, exercices
 - Web Goat³⁹
 - Insecure Labs⁴⁰
 - Google-Gruyere⁴¹

Sources

³³https://www.owasp.org/index.php/Main_Page

³⁴<https://www.youtube.com/watch?v=pHI2zitLph8>

³⁵<https://www.w3.org/TR/webauthn/>

³⁶https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API

³⁷https://www.journaledunet.com/developpeur/tutoriel/php/031030php_nexen-xss1.shtml

³⁸<https://www.apprendre-php.com/tutoriels/tutoriel-39-introduction-aux-cross-site-request-forgeries-ou-sea-surf.html>

³⁹<https://www.owasp.org/index.php/Webgoat>

⁴⁰<https://www.insecurelabs.org/task>

⁴¹<https://google-gruyere.appspot.com/>