Recherche de vulnérabilités Web

Guide synthétique pour un pentest applicatif

Objectif : apprendre à passer d'une reconnaissance méthodique à l'identification et à la validation contrôlée des principales vulnérabilités d'une application Web. Le document privilégie les tests simples, reproductibles et non destructifs. Il s'inspire notamment de l'OWASP Web Security Testing Guide (WSTG), qui recommande de cartographier les points d'entrée avant de tester les contrôles d'entrée, d'authentification, d'autorisation, de session et de logique métier.

1. Règle fondamentale : cartographier avant d'attaquer

Un pentest Web ne consiste pas à envoyer quelques payloads au hasard. Pour chaque fonctionnalité, il faut identifier ce que le client envoie au serveur, ce que le serveur renvoie, ce qui est stocké et ce qui est ensuite réutilisé. Les paramètres GET/POST, champs de formulaires, cookies, headers, JSON, fichiers et endpoints API sont autant de points d'entrée à inventorier.

À relever	Exemples	Pourquoi ?
URLs	/login, /admin, /api/users	Surface d'attaque
Paramètres	id, q, file, url, redirect	Points d'injection
Formulaires	login, recherche, upload	Validation et XSS/injection
API	REST, GraphQL	Contrôles d'accès et données
Sessions	cookies, tokens	Authentification/session
Fichiers	upload/download	Traversal, upload, exposition

2. Méthode de travail

- 1. Définir le périmètre : domaines, comptes de test, plages horaires, actions interdites.
- 2. Découvrir : DNS, sous-domaines, technologies, URLs, formulaires, API, fichiers et JavaScript.
- 3. Intercepter : utiliser un proxy HTTP pour observer les requêtes et réponses.
- 4. Établir les hypothèses : « ce paramètre est-il réfléchi ? », « cet objet appartient-il à l'utilisateur ? ».
- 5. Tester progressivement : commencer par des valeurs inoffensives et comparer les réponses.
- 6. Confirmer : reproduire la faiblesse et mesurer son impact sans détruire de données.
- 7. Documenter : URL, requête, réponse, preuve, impact, sévérité et recommandation.

3. Outils libres à connaître

Outil	Rôle principal
OWASP ZAP	Proxy, exploration, analyse passive et DAST ; excellent outil de formation.
Nmap	Découverte réseau/ports/services autour de la cible Web.

curl	Requêtes HTTP reproductibles et tests rapides.
WhatWeb / Wappalyzer	Identification de technologies et composants.
Nikto	Contrôles de configuration et fichiers/ressources connus.
Nuclei	Détection basée sur des templates ; à utiliser avec prudence et validation manuelle.
SQLmap	Automatisation SQLi ; à réserver aux environnements autorisés et après identification d'un point suspect.
Semgrep	Analyse statique du code lorsque le code source est disponible.

4. XSS — Cross-Site Scripting

Objectif : déterminer si une donnée contrôlée par l'utilisateur est réinjectée dans une page sans encodage adapté. Les formes principales sont reflected XSS, stored XSS et DOM XSS.

Comment tester

- Lister les champs : recherche, nom, commentaire, profil, paramètres d'URL, headers et données JSON.
- Commencer par un marqueur unique, par exemple WATCHIT_XSS_123, et vérifier où il réapparaît.
- Identifier le contexte : texte HTML, attribut HTML, JavaScript, URL ou DOM.
- Si le contexte paraît vulnérable, confirmer dans un environnement de test avec un payload XSS inoffensif.
- Pour le stored XSS, vérifier le second temps : saisie → stockage → affichage sur une autre page/utilisateur.
- Pour DOM XSS, examiner le JavaScript et les sources (URL, hash, localStorage) et sinks dangereux.

```
Exemple de test de réflexion : /search?q=WATCHIT_XSS_123
```

5. Injections : SQL, commandes et autres

SQL Injection

Rechercher tous les paramètres susceptibles d'être utilisés dans une requête SQL : login, recherche, identifiants, filtres, tri, API JSON, cookies, etc. Commencer par provoquer une différence de comportement ou une erreur contrôlée. Les requêtes préparées côté serveur sont le contrôle attendu.

- Comparer une valeur normale et une valeur contenant un caractère spécial (par exemple une apostrophe).
- Observer les erreurs SQL, différences de contenu, codes HTTP ou temps de réponse.
- Tester ensuite, dans un lab, des conditions booléennes simples ou utiliser un outil automatisé.
- Ne pas utiliser des tests destructifs sur UPDATE/DELETE/INSERT.

Command Injection

Chercher les fonctions qui semblent appeler le système : ping, conversion, compression, traitement d'image, nom d'hôte, chemin de fichier, etc. Vérifier si une donnée utilisateur influence une commande système. La confirmation doit être réalisée avec une preuve inoffensive dans un laboratoire.

Autres injections à connaître

- LDAP / XPath / XML
- SSTI (templates)
- NoSQL
- Expression Language
- Injection dans des commandes ou interpréteurs spécialisés

6. Path Traversal / LFI

Rechercher les paramètres liés aux fichiers : file, path, filename, download, document, page, template, include. Le principe est de vérifier si l'application permet de sortir du répertoire prévu ou de charger une ressource non autorisée.

- Identifier un endpoint qui lit ou télécharge un fichier.
- Comparer le comportement avec un fichier autorisé puis un chemin relatif contrôlé dans un lab.
- Tester les variantes d'encodage uniquement dans un environnement prévu pour cela.
- Vérifier également les archives ZIP/TAR si l'application les extrait : un nom de fichier malveillant peut provoquer une écriture hors du répertoire cible.
- Ne pas lire ou exfiltrer de données réelles : une preuve minimale suffit.

```
/download?file=nom_du_fichier
```

7. Upload de fichiers

- Identifier les extensions et types MIME autorisés.
- Tester un fichier légitime valide puis vérifier le nom, l'emplacement et l'URL résultante.
- Vérifier si le serveur se base uniquement sur l'extension ou le Content-Type.
- Tester les noms particuliers, doubles extensions et tailles limites dans un lab.
- Vérifier si le fichier est servi depuis le webroot et s'il peut être interprété comme du code.
- Pour les contrôles antivirus, utiliser le fichier de test EICAR plutôt qu'un malware réel.
- Vérifier les archives si elles sont acceptées : extraction, chemins et taille décompressée.

Une bonne défense combine validation du type réel, renommage, stockage hors webroot lorsque possible, permissions minimales et contrôle du contenu.

8. Authentification et pages de login

- Tester les messages d'erreur : ne doivent pas permettre une énumération évidente des comptes.
- Tester la limitation des tentatives et la résistance aux essais répétés, sans provoquer de déni de service.
- Vérifier la récupération de mot de passe : token aléatoire, expiration, usage unique et invalidation.
- Vérifier que le changement de mot de passe invalide correctement les anciennes sessions selon la politique attendue.
- Tester MFA lorsqu'il existe : activation, récupération, changement de facteur et accès aux fonctions sensibles.
- Vérifier la fixation/rotation de session : l'identifiant de session doit être renouvelé après authentification.

Cookies de session

- Secure
- HttpOnly
- SameSite
- Path/Domain
- expiration et rotation

9. Autorisation, IDOR et contrôle d'accès

Très souvent, une application authentifie correctement un utilisateur mais vérifie mal ce qu'il a le droit de consulter ou modifier.

```
Alice → /api/appointments/123  ✓
Alice → /api/appointments/456  ?
Bob   → /api/appointments/123  ?
```

- Créer au moins deux comptes de test et, si possible, un compte administrateur.
- Changer les identifiants d'objets dans les URLs, JSON et formulaires.
- Tester lecture, modification, suppression et téléchargement.
- Tester les endpoints d'administration directement, même s'ils ne sont pas visibles dans l'interface.
- Tester aussi les différences de privilèges entre rôles : utilisateur → manager → administrateur.

10. CSRF, CORS et méthodes HTTP

CSRF

Pour chaque action sensible effectuée via le navigateur (email, mot de passe, suppression, paiement, rendez-vous), vérifier qu'une requête provenant d'un contexte non autorisé ne peut pas provoquer l'action. Vérifier la présence et surtout la validation effective d'un token CSRF lorsque le modèle d'authentification le nécessite.

CORS

- Observer Access-Control-Allow-Origin et Access-Control-Allow-Credentials.
- Une origine arbitraire ne doit pas être autorisée à lire des données sensibles authentifiées.
- Tester séparément les endpoints publics et authentifiés.

Méthodes HTTP

OPTIONS permet notamment d'énumérer les méthodes annoncées. Vérifier ensuite que les méthodes dangereuses ne sont pas utilisables sans autorisation.

Le WSTG recommande d'énumérer les méthodes supportées et de tester les possibilités de contournement liées aux méthodes HTTP.

11. SSRF, Open Redirect et Webhooks

Indice	À tester

url= / uri= / endpoint=	SSRF potentiel
webhook= / callback=	SSRF, validation d'URL
redirect= / next= / return=	Open Redirect
image_url= / import_url=	SSRF potentiel

- Déterminer si la requête est exécutée par le navigateur ou par le serveur.
- Pour SSRF, utiliser uniquement un serveur de test que vous contrôlez afin de confirmer que le serveur cible effectue la requête.
- Pour Open Redirect, vérifier si une URL externe peut être fournie sans validation.
- Ne pas sonder des réseaux internes ou des services tiers sans autorisation explicite.

12. API, GraphQL et JavaScript

- Rechercher /api/, /v1/, /v2/, swagger.json, openapi.json, GraphQL et documentation publique.
- Extraire les endpoints présents dans les fichiers JavaScript.
- Rechercher les secrets accidentellement intégrés : clés, tokens, URLs internes — en masquant toute valeur dans le rapport.
- Pour les API, tester systématiquement authentification + autorisation + validation des types + propriétés supplémentaires.
- Pour GraphQL, identifier l'endpoint et vérifier, dans un environnement autorisé, l'exposition excessive du schéma.
- Pour WebSocket, vérifier authentification, autorisation et validation des messages.

13. Configuration et informations divulguées

- Headers : CSP, HSTS, X-Content-Type-Options, Referrer-Policy, Permissions-Policy.
- Fichiers : .env, .git, sauvegardes, archives, logs, fichiers temporaires, anciennes versions.
- Débogage : stack traces, phpinfo, messages d'erreur détaillés.
- Serveur : versions et composants exposés inutilement.
- Répertoires : listing, consoles d'administration, interfaces de maintenance.
- TLS : certificats, protocoles/ciphers obsolètes, redirection HTTP→HTTPS.

14. Logique métier

Les scanners détectent mal les failles de logique métier. Il faut comprendre le processus puis essayer de le contourner.

- Réutiliser une action après annulation ou expiration.
- Modifier un objet appartenant à un autre compte.
- Contourner une limite de quantité, prix, nombre de réservations ou étapes.
- Répéter une opération qui devrait être unique.
- Utiliser deux requêtes concurrentes pour vérifier une condition de course dans un environnement de test.
- Modifier directement une requête/API alors que l'interface interdit l'opération.

15. Checklist de débutant

Famille	Questions rapides
Reconnaissance	Quelles URLs, technologies, sous-domaines, API et fichiers existent ?
Entrées	Quels champs GET/POST/JSON/cookies/headers puis-je contrôler ?
XSS	Ma valeur revient-elle ? Dans quel contexte ? Est-elle correctement encodée ?
Injection	Une entrée provoque-t-elle une erreur ou un comportement anormal côté serveur ?
Fichiers	Y a-t-il upload/download/path/include ?
Auth	Login, reset, MFA et sessions sont-ils robustes ?
Autorisation	Puis-je accéder à l'objet ou à l'action d'un autre rôle ?
CSRF/CORS	Les actions authentifiées peuvent-elles être déclenchées depuis une origine non autorisée ?
API	Les endpoints contrôlent-ils réellement identité et permissions ?
Config	Debug, secrets, backups, headers et méthodes HTTP sont-ils correctement configurés ?
Métier	Puis-je contourner une règle fonctionnelle ?

18. Références essentielles

- OWASP Web Security Testing Guide (WSTG) — méthodologie générale et cas de test : https://wstg.owasp.org/
- OWASP — SQL Injection : https://wstg.owasp.org/latest/4-Web_Application_Security_Testing/07-Injection/05-SQL_Injection/
- OWASP — Directory Traversal / File Include : https://wstg.owasp.org/latest/4-Web_Application_Security_Testing/05-Authorization/01-Directory_Traversal_File_Include/
- OWASP — Identify Application Entry Points : https://wstg.owasp.org/latest/4-Web_Application_Security_Testing/01-Information_Gathering/06-Identify_Application_Entry_Points/
- OWASP ZAP : https://www.zaproxy.org/
- Nmap : https://nmap.org/
- PortSwigger Web Security Academy : https://portswigger.net/web-security