Sécurité des serveurs et des applications, Partie 2

Sécurité des applications web

Ce qu’on va voir

- Vecteurs d’attaques
- Path traversal - LFI - RFI
- Attaques XSS
- Attaques CSRF
- SQL injections

Vecteurs d’attaques 1/2

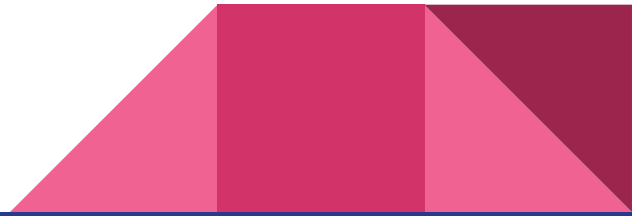
- Entrées utilisateurs :
  - Formulaires (requêtes GET et POST)
  - Fichiers “uploadés” (requêtes PUT)
  - autres requêtes HTTP (HEAD, TRACE, etc)
  - Cookies
- Fichiers d’installations (dans le cas de CMS)
- Librairies/logiciels dépréciés

Vecteurs d’attaques 2/2

- Langage de l’application :
  - type juggling : différences entre == et ===
  - librairie standard (options des fonctions standards)
  - fonctions dépréciées (exemple addslashes, htmlspecialchars en PHP)
  - obfuscation != sécurité

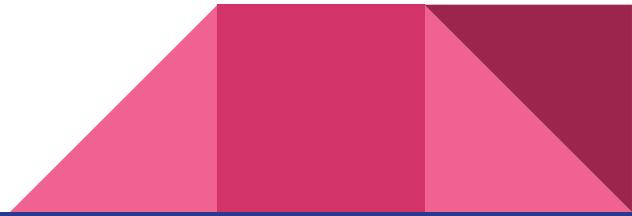
Indices pour un attaquant

- Fichiers de gestion de versions (.git)
- Fichiers log
- Gestion des erreurs
- robots.txt
- .htaccess/.htpasswd mal protégés



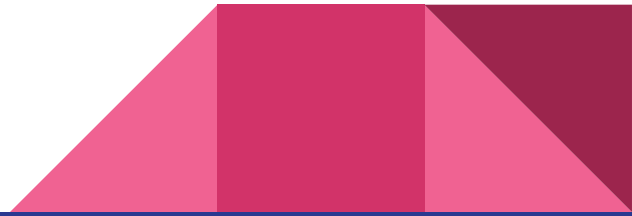
Path traversal - LFI - RFI

- Path traversal : parcours du système de fichiers à partir de l'application
  - http://mon-app.com/index.php?file=news
  - -> http://mon-app.com/index.php?file=%2fetc%2fpasswd%00
  - -> http://mon-app.com/index.php?file=%2e%2e%2fadmin%2findex
- Local File Inclusion :
  - path traversal + injection de code (sur le serveur local)
- Remote File Inclusion : path traversal + injection de code distant
  - exemple : XML External Entity (XXE)



Cross-Site Scripting (XSS)

- Injection de code Javascript côté client :
  - dans un message de forum
  - dans un mail
  - dans une URL (le javascript est exécuté dans la réponse du serveur)
- L'exécution du code injecté permet :
  - vol de données personnelles
  - vol de session
  - exploité une faille du navigateur (Heap/JIT spraying)
  - etc



Cross-site request forgery

- Un utilisateur précis (un admin par exemple) peut effectuer des modifications par envoie de requêtes
  - GET : http://mon-appli.com/index.php?action=toutCasse
  - POST (formulaire)
- Un attaquant peut forger une requête malicieuse et la faire exécuter par l'utilisateur ciblé
  - GET : url d'une image
  - POST : faux formulaire (phishing)
  - autres (PUT, DELETE) : en utilisant préalablement une XSS
- Contremesures : Same-origin policy (SOP) et Cross-origin resource sharing (CORS)

SQL injection (basique)

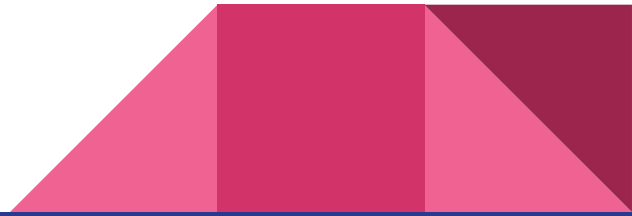
Requête originale :

“SELECT uid FROM Users WHERE name = '$name' AND password = '$passwd’ ”

Schémas d’attaques :

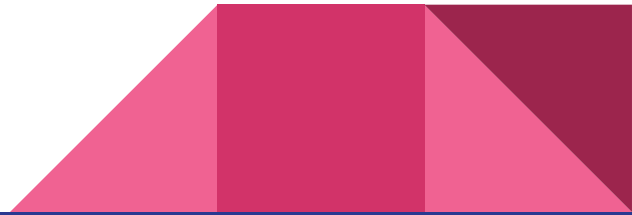
- $name = “lol’ ; --”
- $passwd = “lol’ or 1; --”

Utilisation : authentification, injection de code SQL



SQL injection (variantes)

- Error SQL injection :
  - le résultat de la requête provient d'un message d'erreur
  - l'attaque dépend du SGBD utilisé
- Blind SQL injection :
  - le résultat observable de la requête est un booléen
- Double blind SQL injection
  - aucun résultat visible => on utilise le temps en ajoutant des instructions sleep
- Plus de détails : https://sqlwiki.netspi.com



Et d’autres

- injections (LDAP, NoSQL, XPATH, PDF, Flash, etc)
- Sérialisation/Désérialisation
- Mauvaises fonctions de hash (ou mauvaise utilisation)
- Mauvais générateurs aléatoires (de nombres, d’identifiants uniques, etc)
- etc

Conclusion

- Les attaques “réelles” sont des combinaisons de plusieurs attaques “simples”
- Plusieurs outils existent pour vérifier automatiquement certains critères de sécurité
- Une faille de sécurité est vite arrivée et peut aussi se trouver automatiquement
- Avec les normes européennes types RGPD, la sécurité devient un enjeu légal