



La sécurité des applications web

David THIBAU – 2024

david.thibau@gmail.com



Agenda

- **Introduction**

- Particularités des applications web
- Client
- Serveur
- Les types d'attaques
- Cryptographie / TLS

- **OWASP**

- Ressources
- Les Top Ten

- **Construire une application sécurisée**

- Architecture de la sécurité
- Code contre les attaques
- Revue de sécurité
- Découverte et gestion de vulnérabilités

- **Test de la sécurité**

- Introduction
- Techniques de reconnaissance
- Étendue des tests
- ZAP



Introduction

Particularités du web

Client

Serveur

Types d'attaque

Cryptographie



Sécurité du web

La sécurité du web est particulière car elle concerne plusieurs composants :

- **Navigateur** :
 - Accède à plusieurs sites en même temps :
Application business sécurisé, réseau sociaux, recherche
 - Exécute des scripts, stocke des données
- **Serveur** : Les attaquants peuvent envoyer n'importe quelle requête HTTP
- **Réseau** : Piratage de DNS, Vol de domaine, Faux Certificats, Scanning réseau



Motivation des attaques

Spam : Envoyé à partir d'une adresse IP légitime, moins susceptible d'être bloqué

Déni de service : Attaquer des concurrents ou demander une rançon

Virus : Infecter les utilisateurs en visite avec des logiciels malveillants, Infectez un serveur et utilisez-le pour infecter des centaines de milliers de clients.

Le vol de données : Voler des informations d'identification, des numéros de carte de crédit et des droits de propriété intellectuelle



Pourquoi la sécurité web est difficile

Objectif extrêmement ambitieux – Exécuter du code non fiable en toute sécurité

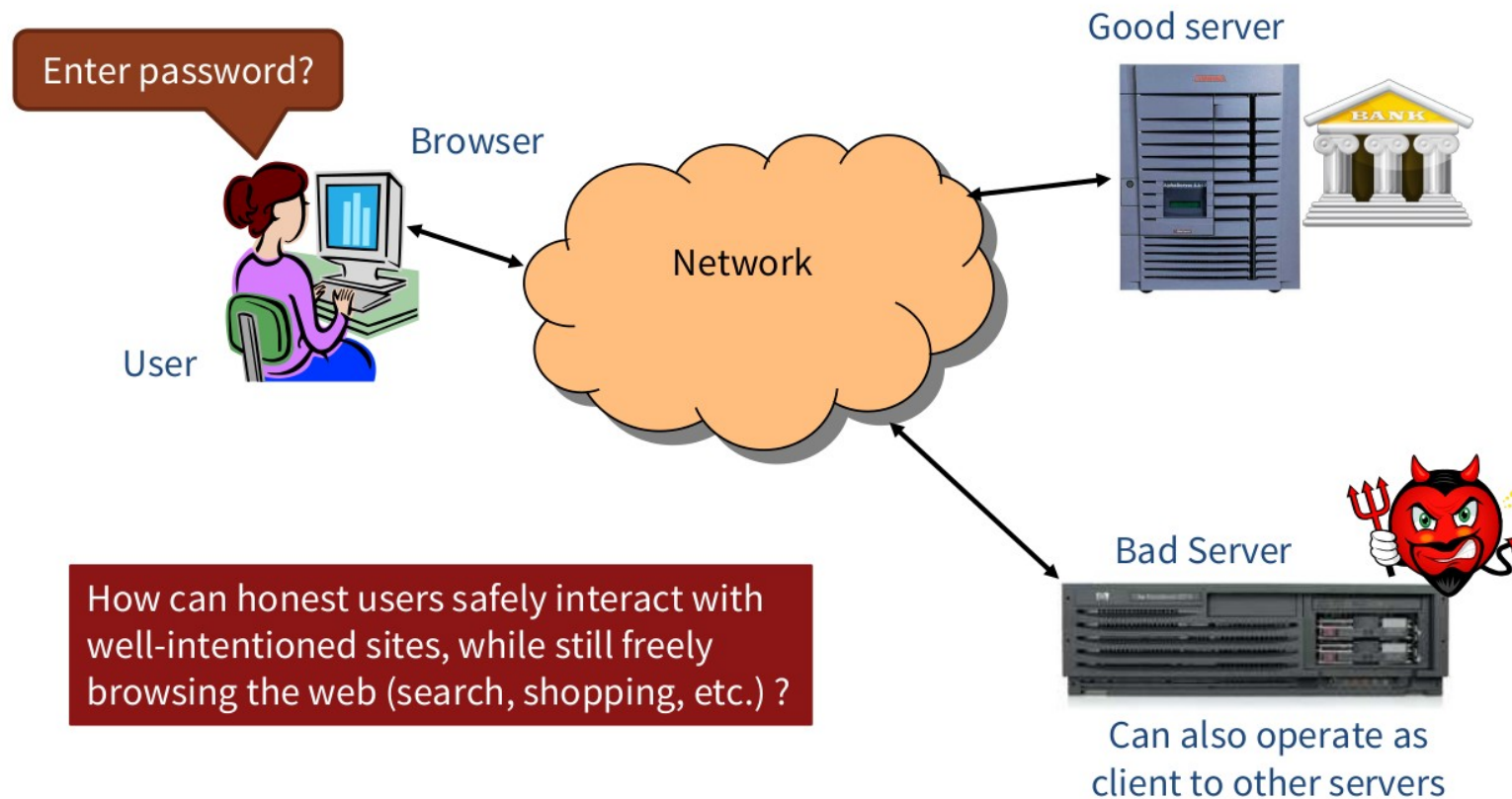
Différents sites interagissant dans un même onglet (« mashups »)

Fonctionnalités de bas niveau ; accès au matériel

Désir de haute performance

Les API ont évolué et nous avons des exigences strictes de compatibilité ascendante

Challenges





Objectifs Sécurité

Naviguez sur le Web en toute sécurité

Les utilisateurs doivent pouvoir visiter une variété de sites Web, sans encourir de préjudice :

- Aucune information volée (sans l'autorisation de l'utilisateur)
- Le site A ne peut pas compromettre la session sur le site B
- Prise en charge des applications Web sécurisées

Les applications livrées sur le Web doivent avoir les mêmes propriétés de sécurité que les applications autonomes

- Étant donné que de nombreuses applications mobiles sont des interfaces vers des sites Web,
- Prend en charge la sécurité des applications mobiles.



Introduction

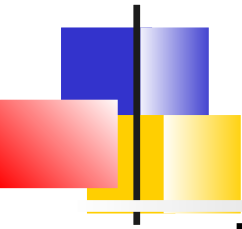
Particularités du web

Navigateur

Serveur

Types d'attaque

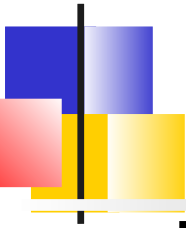
Cryptographie



Tâches du navigateur

Les sites – même malveillants – peuvent :

- Télécharger du contenu de n'importe où
- Démarrer des processus sur le client
- Ouvrir des sockets vers un serveur, ou même vers le navigateur d'un autre utilisateur
- Afficher les médias dans un grand nombre de formats
- Exécutez du code personnalisé
- Enregistrer/lire les données du système de fichiers



Fonctionnalités dangereuses

Une page web peut envoyer de l'information à n'importe quel site :

Ex :

- ``
- ClickJacking

```
<style>
iframe { width: 400px; height: 100px; position: absolute;
top:0; left:-20px; opacity: 0 ; z-index: 1; } </style>
<div>Click to get rich now:</div>
<!-- L'url du site de la victime →
<iframe src="/clickjacking/facebook.html"></iframe>
<button>Click here!</button>
```



Javascript

Javascript est puissant et a accès a bcp de ressources du navigateur.

Exemples, scanning de port :

- Demander à charger des Ips internes :

- ``

- *OnError ou timeout* : donne l'indication que l'URL demandée a un soucis => Port non Ouvert



Isolation

Le navigateur fournit une isolation basée sur les frames et onglets

Les règles du web isole les interactions entre un site et un autres site

- Same Origin Policy
- HTML5 Sandbox



Same Origin Policy

Règle fondamentale du web : Deux pages provenant de sources différentes ne doivent pas interférer les unes avec les autres

Chaque cadre d'une page a une origine :

protocole://hôte:port

Le cadre peut accéder à sa propre origine

- Accès réseau, Lecture/écriture DOM, Stockage (cookies)

Le cadre ne peut pas accéder aux données associées à un autre origine MAIS

- Peut proposer des liens, faire des redirections, soumission de formulaire
- Embarquer des ressources



Problèmes de SOP

Parfois, la politique est trop étroite :

Difficile pour *axess.formation.org* d'échanger des données avec *login.formation.org*

Parfois, la politique est trop large :

aucun moyen d'isoler

<https://web.formation.org/class/cs106a/> de

<https://web.formation.org/class/cs253/>

Cette règle est donc quelquefois assouplie



document.domain

Permet à 2 origines différentes de communiquer

Les 2 sites doivent être d'accord pour communiquer
et doivent partager un domaine de haut niveau

document.domain = 'formation.org'

Attention, n'importe quel sous-domaine peut se
joindre au groupe :
attacker.formation.org

=> En général une mauvaise idée et technique
dépréciée



Frames et hash

Envoyer des messages d'une page parent vers une page enfant qui ne sont pas de la même origine

- Les données sont encodées dans le fragment de l'URL (#)
- Le parent est autorisé à naviguer dans les iframes enfants.
- L'enfant peut poller les changements de fragment.



Example

site-a.com:

```
<h1>Parent: http://site-a.com:8000</h1>
<input name='val' />
<br /><br />
<iframe src='http://site-b.com:8001' width='100%' height='400px'></iframe>
<script>
  const input = document.querySelector('input')
  const iframe = document.querySelector('iframe')
  input.addEventListener('input', () => {
    iframe.src = `http://site-b.com:8001#${encodeURIComponent(input.value)}`
  })
</script>
```

site-b.com:

```
<h1>Child: http://site-b.com:8001</h1>
<div></div>
<script>
  const div = document.querySelector('div')
  setInterval(() => {
    div.textContent = decodeURIComponent(window.location.hash).slice(1)
  }, 100)
</script>
```



L'API *postMessage*

window.postMessage, provoque l'envoi d'un *MessageEvent* vers un site qui coopère

- MessageEvent encapsule données et origin
- Le site coopérant peut donc tester l'origine

Avantages :

- => Envoi de données arbitraires
- => Ne peut pas manipuler le DOM

Attention :

- L'émetteur doit restreindre les origines
- Le récepteur doit faire un test sur l'origine du message



Example

site-a.com:

```
<h1>Parent: http://site-a.com:8000</h1>
<input name='val' />
<br /><br />
<iframe src='http://site-b.com:8001' width='100%' height='400px'></iframe>
<script>
  const input = document.querySelector('input')
  const iframe = document.querySelector('iframe')
  input.addEventListener('input', () => {
    iframe.contentWindow.postMessage(input.value, 'http://site-b.com:8001')
  })
</script>
```

site-b.com:

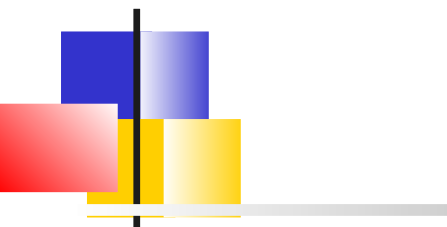
```
<h1>Child: http://site-b.com:8001</h1>
<div></div>
<script>
  const div = document.querySelector('div')
  window.addEventListener('message', event => {
    if (event.origin !== 'http://site-a.com:8000') return
    div.textContent = event.data
  })
</script>
```



CORS

Cross-Origin Resource Sharing est un mécanisme basé sur un en-tête HTTP qui permet d'indiquer toute origine autre que la sienne à partir de laquelle un navigateur devrait autoriser le chargement de ressources.

Dans ses dernières versions, CORS s'appuie sur une requête de ***preflight*** qui vérifie que la requête réelle sera autorisée.



Client

Server

Preflight request

```
OPTIONS /doc HTTP/1.1
Origin: https://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-type
...
```

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
...
```

Main request

```
POST /doc HTTP/1.1
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Origin: http://foo.example
...
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://foo.example
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 235
...
```



HTML5 Sandbox

Spécifier l'attribut sandbox de l'iframe

```
<iframe sandbox  
  src="http://untrusted.site.net/content"></iframe>
```

Attribut permettant d'interdire ou de laisser autorisé

- L'exécution de plugins .
- La soumission de formulaires.
- L'exécution de scripts.
- Les liens vers d'autres contextes de navigation sont désactivés.
- Le contenu n'est pas capable de traverser le DOM ou de lire les informations sur les cookies.



Etat du client

Le navigateur peut également stocker des informations sur le poste client

- Cookies : Généralement pour conserver la session utilisateur, échangé entre le client et le serveur
- LocalStorage, sessionStorage : Pas échangé avec le navigateur, accessible par script et par les outils de dév. Stocké en fonction de l'origine

Règle générale : Ne jamais stocker des informations sensibles côté client. (password, cb, ...)



Local Storage / Security Issues

Stockage par origine

Peut être consulté par l'utilisateur avec un accès local (varie selon le navigateur)

Accessible par JavaScript dans la page

- pas de httpOnly donc vulnérable aux attaques XSS

Les attaques XSS peuvent lire ou écrire le stockage local

- Ne stockez pas d'informations sensibles
- Ne faites pas confiance aux données lues à partir du stockage local



Introduction

Particularités du web

Navigateur

Serveur

Types d'attaque

Cryptographie



Stack Technologique

Les applications serveurs ont une stack technologie qui comprend plusieurs composants :

- L'OS
- Le serveur HTTP
- Le code applicatif s'exécutant au dessus d'un framework
- Un ensemble de services remote



Responsabilités

Du point de vue de la sécurité, le serveur est responsable de

- **L'authentification** : Mécanisme permettant de s'assurer de l'identité du client/utilisateur
- **L'autorisation** : S'assurer que les ressources sont accessibles seulement par les clients/utilisateurs autorisés
- La **disponibilité** : garantie que les données autorisées restent accessibles



Authentication

Les technologies d'authentification ont évoluées et peuvent donc être diverses à l'heure actuelle :

- **Basic or Digest authentication** : Login/password passé via l'entête Authorization
- **Form based authentication** : Login/password soumis par un formulaire
- **OpenId/oAuth2** : Authentification sur un service tiers puis token passé dans l'entête Authorization

Pour toutes ce techniques, afin que l'utilisateur ne soit pas obligé de se ré-authentifier.

Le client détient les informations d'identification de session :
Mémoire, Cookie, Local Storage

D'autres mécanismes peuvent s'ajouter :

- 2FA : Une deuxième preuve est demandé via le mobile en général
- WebAuthn : Basé sur la cryptographie à clé publique



Autorisation

Droit ou autorisation accordé à une entité pour accéder à une ressource

L'accès peut être accordé/refusé en fonction de l'identification du client

- IP
- Rôle de l'utilisateur RBAC
- Scope du client OAuth2
- ...

L'autorisation est généralement implémenté par le code applicatif (framework ou autre)



Introduction

Particularités du web

Navigateur

Serveur

Types d'attaque

Cryptographie




XSS

Cross-Site Scripting (XSS) s'appuie sur le fait que les applications web exécutent des scripts dans le navigateur. Un script malveillant peut alors s'exécuter

Différentes catégories de XSS :

- Stocké : le code malveillant est stocké dans une base de données avant exécution
- Réfléchi : le code malveillant est renvoyé dans une réponse du serveur
- Basé sur DOM : le code malveillant est à la fois stocké et exécuté dans le navigateur.



Exemple : XSS Stocké

1. L'utilisateur soumet un commentaire via un formulaire Web
2. Le commentaire de l'utilisateur est stocké dans la base de données
3. le commentaire est présenté via une requête HTTP par un ou plusieurs utilisateurs
4. Le commentaire dans la page est interprété comme DOM plutôt que comme texte



Exemple reflected XSS

Le résultat d'une recherche sur un site reprend les termes recherchés dans la réponse

```
support.mega-bank.com/search?  
query=open+<script>alert(test);</script>checking+account
```

Une fois la vulnérabilité détectée, il est alors possible d'envoyer des mails malicieux contenant des liens comme :

```
<a href="https://support.mega-bank.com/search?  
query=open+<script>alert('test');</script>checking+account">
```

```
Create a New Checking Account</a>
```



DOM XSS

Les DOM XSS s'appuie sur le fait que de nombreuses applications réécrivent le DOM à partir d'entrée de l'utilisateur.

Par exemple, le hash servant à filtrer une liste
investors.mega-bank.com/listing#usa

Le hash est utilisé pour réécrire le DOM

document.write(nMatches + ' matches found for ' + hash);

Un DOM XSS pourrait être

investors.mega-bank.com/listing#<script>alert(document.cookie);</script>



Résumé XSS

Exécutez un script dans le navigateur qui n'a pas été écrit par le propriétaire de l'application Web

Peut s'exécuter en coulisses, sans aucune visibilité ni intervention de l'utilisateur requise pour démarrer son exécution

Peut obtenir tout type de données présentes dans l'application Web actuelle

Peut librement envoyer et recevoir des données à partir d'un serveur Web malveillant

Se produisent à la suite d'une entrée utilisateur mal nettoyée et intégrée dans l'interface utilisateur.

Peut être utilisé pour voler des jetons de session, conduisant au piratage du compte

Peut être utilisé pour dessiner des objets DOM sur l'interface utilisateur actuelle, conduisant à un phishing parfait qui ne peuvent pas être identifiées par un utilisateur non technique



Cross-Site Request Forgery (CSRF)

Un attaque ***Cross-Site Request Forgery (CSRF)*** consiste à entrer de façon illégale dans un site tiers, par le biais d'un utilisateur autorisé.

L'attaque suppose que l'utilisateur est loggé dans le site victime

Elle concerne principalement les sessions gérées par les cookies



Exemple

Le endpoint suivant permet à un utilisateur loggé de faire un transfert d'argent

GET <http://bank.com/transfer?accountNo=1234&amount=100>

Le hacker convainc l'utilisateur visé de cliquer sur un lien pendant que sa session est active

``

Show Kittens Pictures ``

Ou à visiter un site web contenant :

``

En mode POST :

`<body onload="document.forms[0].submit()">`

`<form action="http://bank.com/transfer" method="POST">`

`<input type="hidden" name="accountNo" value="5678"/>`

`<input type="hidden" name="amount" value="1000"/>`

`<input type="submit" value="Show Kittens Pictures"/>`

`</form>`



CSRF et API Stateless

Pour les APIs stateless n'utilisant pas la session http, la vulnérabilité CSRF dépend de la manière dont le client stocke le jeton d'identification et d'accès à l'API.

- Variable globale Javascript : Le plus secure car les attaques CSRF ou XSS ouvre une nouvelle page qui n'a pas accès au variable globale de la page appelante.
Mais cela oblige l'utilisateur a se réauthentifier en cas de reload.
=> Peu Utilisé
- Stockage navigateur (Ex. session storage) : Protège de CSRF car le stockage à la différence des cookies n'est pas automatiquement envoyé vers le serveur.
- Cookies :
 - Stockage du token. Lecture du crédentiel par le client afin de remplir l'entête d'autorisation => Pas vulnérable au CSRF
 - Utilisé pour l'authentification avec le flag HTTP-only => Vulnérable au CSRF



SSRF

Server-side request forgery (SSRF) permet à un attaquant de faire en sorte que l'application côté serveur envoie des requêtes vers un emplacement involontaire, comme par exemple *localhost*, des adresses locales, ...

Exemple, le serveur lit ses résultats grâce à un autre service :

```
<select name="stockApi">
  <option value="http://stock.weliketoshop.net:8080/product/stock/check?
productId=3&storeId=1">London</option>
  <option value="http://stock.weliketoshop.net:8080/product/stock/check?
productId=3&storeId=2">Paris</option>
</select>
```

Le hacker modifie le HTML par

```
<select name="stockApi">
  <option value="http://localhost/admin">London</option>
  <option value="http://stock.weliketoshop.net:8080/product/stock/check?
productId=3&storeId=2">Paris</option>
</select>
```




XML External Entity (XXE)

XML External Entity (XXE) repose sur un analyseur XML mal configuré dans le code d'une application.

- La spécification XML inclut une annotation spéciale, entité externe, pour l'importation de fichiers externes. Cette directive est interprétée sur la machine sur laquelle le fichier XML est évalué.

L'API endpoint doit accepter un flux XML (SVG, HTML/DOM, XFDF (PDF), RTF)



Attaques

Direct XXE : Le endpoint reçoit du XML et le parse.

L'attaquant préfixe la payload avec :

```
<!ENTITY xxe SYSTEM "file:///etc/passwd"  
>]><xxe>&xxe;</xxe>
```

Indirect XXE : Le endpoint ne prend pas du XML mais génère une requête SOAP vers un autre Webservice à partir d'un de ses paramètres



Risques

Différents risques avec XXE :

- Récupération de fichiers : le contenu du fichier référencé par XXE est renvoyé dans la réponse de l'application.
- Attaques SSRF : l'entité externe référence une URL d'un système back-end.
- Transmettre des données sensibles du serveur à un système contrôlé par l'attaquant.
- Récupérer des données via des messages d'erreur contenant des données sensibles.



Injection

Les attaques par injection comportent deux composants principaux :

- un interpréteur
- une entrée fournie par l'utilisateur qui est d'une manière ou d'une autre lue dans l'interpréteur.

Elles peuvent se produire contre des utilitaires de ligne de commande comme FFMPEG (un compresseur vidéo), des bases de données, le système.



SQL Injection

Avec l'injection SQL, une chaîne SQL dans une payload HTTP l'exécution de requêtes SQL personnalisées au nom de l'utilisateur final.

Traditionnellement dans du code PHP qui ne découplait pas la couche de présentation et la couche d'accès aux données

Moins courant désormais



Code et command injection

Injection de code : Elles peuvent survenir lorsque le backend utilise un CLien utilisant des paramètres de la requête

Injection de commande : Elles peuvent survenir lorsque le back-end exécute une commande bash à partir des paramètres de la requête



Injection de format

Autoriser l'utilisateur à rentrer des chaînes de format permet à un attaquant de provoquer des erreur d'exécution ou de la divulgation d'informations.

Exemple :

```
Formatter formatter = new Formatter(Locale.US);  
String format = "The customer: %s %s has the balance %4$." + userInput + "f";  
formatter.format(format, firstName, lastName, accountNo, balance);
```

Détecté par les outils d'analyse de code comme
FindSecurityBugs



Denial of Service (DoS)

Les attaques ***Denial of Service (DoS)*** peuvent avoir de nombreuses formes :

- Attaque distribuée qui implique de nombreux clients.
- Simple utilisateur effectuant une requête inhabituelle
- Un utilisateur effectuant de nombreuses fois une requête simple.

Les conséquences peuvent aller de simple dégradation de performance au crash du serveur.



Autres types d'attaque

Brute-force : Essayer de large combinaisons d'entrée afin de trouver la bonne

- Peut associer des techniques d'optimisation : dictionnaire, observation statistiques

Credential stuffing : Utilisé des bases de crédits volés auparavant pour essayer de se connecter

Phishing : Inciter l'utilisateur à dévoiler ses informations sensibles

Man-in-the-middle : Le hacker relaie et transforme les requêtes/réponses

Fuzzing : Injecter des données aléatoires et analyser les réponses.

Ces techniques sont utilisées par l'attaquant ou le testeur.



Introduction

Particularités du web

Navigateur

Serveur

Types d'attaque

Cryptographie



Introduction

La cryptographie est la pratique et l'étude de techniques de communication sécurisée en présence d'attaques malveillantes.

Elle est utilisée pour divers aspects de la sécurité : confidentialité, l'intégrité , l'authentification et la non-répudiation.



Message digest

- Un résumé de message (*message digest*) condense en un nombre fixe d'octets un message de n'importe quelle longueur
- L'empreinte apporte 2 propriétés essentielles :
 - Si un seul des caractères du message est altéré, l'empreinte change complètement
 - Quelqu'un en possession du message initial ne *peut* pas générer un faux message ayant la même empreinte



Usage

L'algorithme qui effectue la transformation s'appelle une fonction de hash

Ces fonctions de hachage cryptographique ont de nombreuses applications dans la sécurité informatique :

- Signatures numériques : Identifier un commit, un layer docker
- Codes d'authentification de message (MAC).
- Empreintes digitales, pour détecter des données en double ou identifier de manière unique des fichiers
- Checksum pour détecter une corruption accidentelle de données.
- Sont également utilisés comme fonctions de hachage ordinaires, pour indexer des données dans des tables de hachage.



Algorithme de Hachage

De nombreux algorithmes pour calculer les empreintes existent mais les plus répandus à l'heure actuelle sont :

- MD5 : 1991 pour remplacer MD4
- SHA-1 : 1995 (Gouvernement US)
- SHA-2 : 2001 (National Security Agency)
- SHA-3 : 2015 (NIST)
- ...

Il n'est pas exclu qu'un jour ses algorithmes soient « crackés ».

En particulier, de récents travaux sont arrivés à déceler des régularités dans ces algorithmes et actuellement les spécialistes recommandent l'utilisation de SHA1.

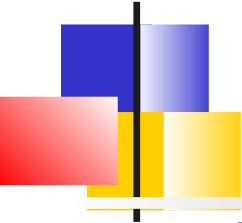


Stockage de mot de passe

Il y a de nombreux exemples de vol du stockage de mots de passe, backup ou autre.

Il est donc essentiel de rendre le plus difficile possible le recouvrement de données sensibles

- Ne jamais stocker en clair
- Hasher les mots de passe en utilisant un sel et éventuellement du poivre



Hash + salt

Première règle : Utilisé une fonction de hash avec un salt différent pour chaque user

password database

Alice	$\mathbf{s_A}$	$H(\text{pw}_A, \mathbf{s_A})$
Bob	$\mathbf{s_B}$	$H(\text{pw}_B, \mathbf{s_B})$
...
id	salt	hash

Pour valider :

$$H(\text{pw}_A, \mathbf{s_A}) \stackrel{?}{=} \text{StoredHash}(\text{Alice})$$



Poivre

Un **poivre (pepper)** est un secret ajouté au mot de passe lors du hachage.

A la différence du sel, il n'est pas stocké avec le mot de passe chiffré mais dans un support séparé.

Le poivre doit être suffisamment long afin qu'il ne soit pas vulnérable au *brute force*



Algorithme de hash

SHA-1 est à éviter car trop rapide et l'attaquant pourra utiliser rapidement le dictionnaire

Utilisez une fonction de hachage à clé (par exemple, HMAC) où la clé est stockée dans le hardware (HSM). Problèmes

- Pas la possibilité d'utiliser HSM sur des petits devices
- Si la clé est-elle même compromise



Ralentir le hash

PBKDF2, bcrypt : fonctions de hachage lentes

Lenteur en "itérant" une fonction de hachage crypto
comme SHA256

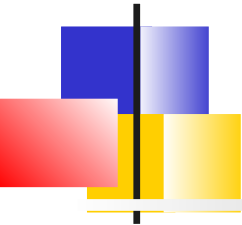
- Nombre d'itérations paramétré (par exemple, défini
sur 1 000 evals/s)

Problème : le matériel personnalisé (par exemple, le
GPU) peut évaluer la fonction de hachage
beaucoup plus rapide qu'un processeur de base

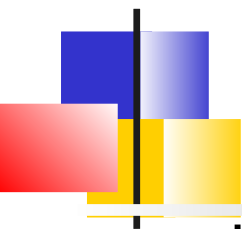
=> Utiliser un fonction de hash qui utilise beaucoup
de mémoire

Scrypt, Argon2¹

Signature numérique



- Si un message et son empreinte sont envoyés séparément, le destinataire peut s'assurer que le message n'a pas été altéré.
- Cependant, si tous les deux sont interceptés, il est facile de modifier le message et de créer une nouvelle empreinte.
- Les signatures numériques basées sur les clés privées et clés publiques permettent de résoudre ce problème.



Clés cryptographiques

Une **clé cryptographique** est une chaîne de caractères utilisée dans un algorithme de chiffrement pour modifier des données afin qu'elles apparaissent aléatoires.

- Comme une clé physique, il verrouille (chiffre) les données afin que seule une personne disposant de la bonne clé puisse les déverrouiller (déchiffrer).

La force de sécurité d'une clé dépend de son algorithme, de la taille de la clé, de la génération de la clé et du processus d'échange de clé

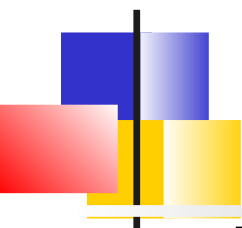
- Les clés peuvent être générés à partir de mot/phrase de passe via des algorithmes de dérivation de clé. Ces algorithmes utilisent généralement des données aléatoires lors de la génération (*salt*)



Utilisation des clés

Il y a 2 façons d'utiliser les clés :

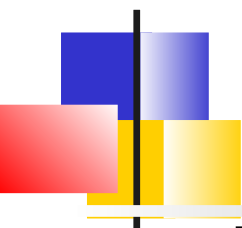
- **Cryptographie symétrique** : La même clé est utilisée pour encoder et décoder. Elle doit être inaccessible des utilisateurs non-autorisés
- **Cryptographie asymétrique** : Une clé sert à encoder, l'autre à décoder.
Une des clés est la clé publique l'autre la clé privée.
Seule la clé privée doit être gardée secrètement



Cryptographie asymétrique

La cryptographie asymétrique permet principalement 2 choses :

- Chiffrer un message : l'expéditeur utilise la clef publique du destinataire pour chiffrer son message. Le destinataire utilise sa clef privée pour déchiffrer le message de l'expéditeur, garantissant la confidentialité du contenu ;
- S'assurer de l'authenticité de l'expéditeur : L'expéditeur utilise sa clef privée pour chiffrer un message que le destinataire peut déchiffrer avec la clef publique de l'expéditeur ; c'est le mécanisme utilisé par la signature numérique pour authentifier l'auteur d'un message



Cryptographie symétrique

L'exigence que les deux parties aient accès à la clé secrète est l'un des principaux inconvénients de la cryptographie symétrique

Cependant, le chiffrement à clé symétrique les algorithmes de chiffrement de clé sont généralement meilleurs pour le chiffrement en masse.

- Ils ont une taille de clé plus petite, ce qui signifie moins d'espace de stockage et une transmission plus rapide.

Pour cette raison, le chiffrement à clé asymétrique est souvent utilisé pour échanger la clé secrète entre 2 systèmes.




Clé symétrique

Un technique de chiffrement utilisé dans TLS est le « **Cipher Block Chaining** » (**CBC**).

- Le message initial est découpé en bloc de taille fixe k .
- Le dernier bloc est complété avec des caractères pour que la longueur du message soit un multiple de k (Padding)
- Pour éviter que 2 blocs similaires est le même résultat :
$$\text{bloc}_n\text{chiffré} = \text{chiffrement}(\text{bloc}_n\text{clair} \oplus \text{bloc}_{n-1}\text{chiffré})$$
- Pour chiffrer le premier bloc, on définit un **vecteur d'initialisation**

L'Oracle Padding basé sur des brute force peut déchiffrer le message initial.

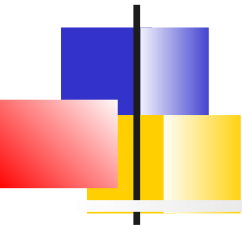
Cette menace peut être mitigée si l'on choisit un bon Vecteur d'initialisation



Cryptographie (asymétrique) avec clé publique

- Utilisation d'une paire de clés :
 - une **publique** pouvant donc transiter sur le réseau
 - l'autre **privée** jalousement gardée dans un *keystore*
- Pour décrypter les données cryptées avec une clé publique, il faut la clé privée et vice-versa
- Il est quasiment impossible de dériver une clé à partir de l'autre
- Séquence :
 - Un algorithme de cryptage génère une signature en utilisant la clé privée et le message original
 - La clé publique correspondante est fournie (au format X.509)
 - Elle permet de vérifier la provenance et l'intégrité du message
- Les algorithmes de cryptage (signature et vérification) les plus connus actuellement sont DSA et RSA

Échange de la clé par le chiffrement symétrique



- Alice génère une clé symétrique aléatoire et l'utilise pour crypter son message
- Alice crypte sa clé symétrique avec la clé publique de Bob
- Alice envoie la clé symétrique cryptée et le message cryptée
- Bob utilise sa clé privé pour décrypter la clé symétrique d'Alice
- Bob peut ensuite décrypter le message

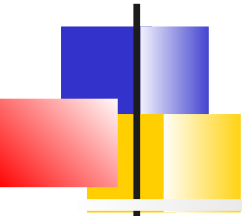


Signature numérique

La signature numérique est un mécanisme permettant de garantir la non-répudiation et l'intégrité d'un document électronique ainsi que d'en authentifier l'auteur.

La signature numérique utilise la cryptographie asymétrique

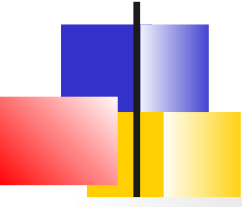
Les *keystore*



- Un ***keystore*** est une base de données (protégée par un mot de passe) stockant des certificats (clé publique signé par une autorité de certification) et des clés (publiques ou privées).
 - Chaque entrée de la base est accessible via un alias
- Un ***truststore*** contient les certificats en qui on a confiance

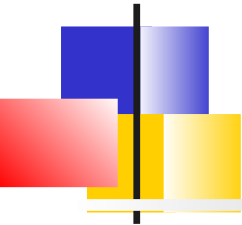
Exemple signature avec les outils Java

1. Génération de clés

- 
- Pour créer un *keystore* et générer une paire de clés
`keytool -genkey -keystore alice.certs -alias alice`
 - Après avoir saisi le mot de passe du *keystore*, *keytool* demande des informations d'identité et un mot de passe pour la paire de clé (qui peut être différent du *keystore*)
 - Les informations demandées sont le *Common Name CN*, *Organizational Unit OU*, *Organization O*, *Location L*, *State ST*, *Country C* (X500)

Exemple signature avec les outils Java

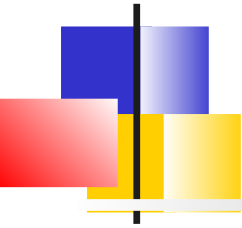
2. *Export de la clé publique*



- L'export de la clé publique s'effectue via :
`keytool -export -keystore alice.certs -alias alice
-file alice.cer`
- Le certificat *alice.cer* peut alors être envoyé et les informations initiales peuvent être affichées
`keytool -printcert -file alice.cer`
- Le certificat peut alors être importé dans un autre keystore :
`keytool -import -keystore bob.certs -alias alice -file
alice.cer`

Exemple signature avec les outils Java

3. Signature et vérification



- Pour la signature d'un message, il faut utiliser l'outil ***jarsigner***

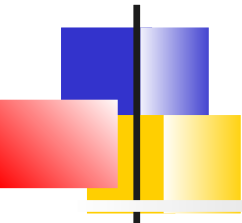
```
jar cvf document.jar document.txt
```

```
jarsigner -keystore alice.certs document.jar alice
```

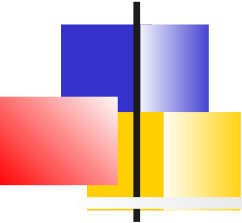
- La vérification utilise également ***jarsigner***

```
jarsigner -verify -keystore bob.certs document.jar
```

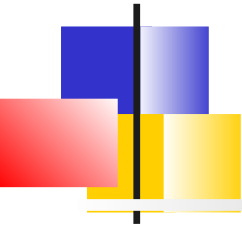
Autorités de certification

- 
- Par contre, n'importe qui peut remplir les informations du certificat avec de fausses données et donc usurper une identité !
 - Les certificats doivent donc être authentifiés par une personne de confiance : une **autorité de certification (CA)** Exemple Verisign
 - La clé publique est alors signée avec la clé privée de l'organisme de confiance
 - Cette clé publique est transmise à l'autorité par une voie sûre (rencontre physique, procédure de contrôle plus ou moins sûre selon le prix du certificat)
 - Il est courant qu'une signature numérique ait été signée par plusieurs autorités de confiance

Vérification de signature

- 
- La vérification de signature est alors déléguée à l'autorité de certification
 - La clé publique du CA est alors préinstallée dans un *keystore*
 - Toutes les clés signées par ce CA sont alors vérifiées implicitement et peuvent donc être importées directement dans un *keystore* possédant la clé publique du CA

TLS / SSL



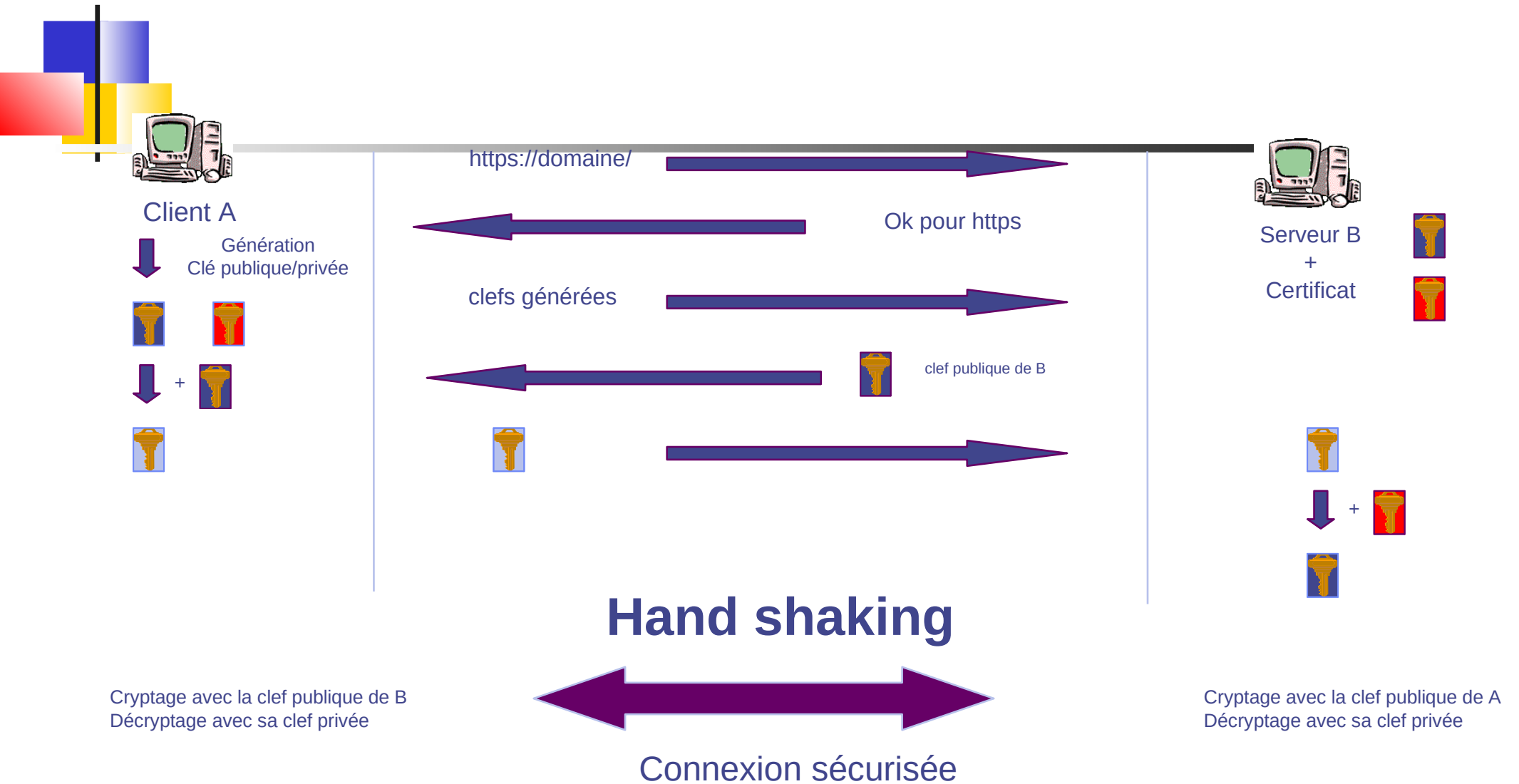
TLS/ SSL permet de sécuriser la connexion entre le navigateur et l'application Web.

Les données échangées sont cryptées (par le navigateur et par le serveur) avant d'être envoyés puis décrypter lors de leur réception.

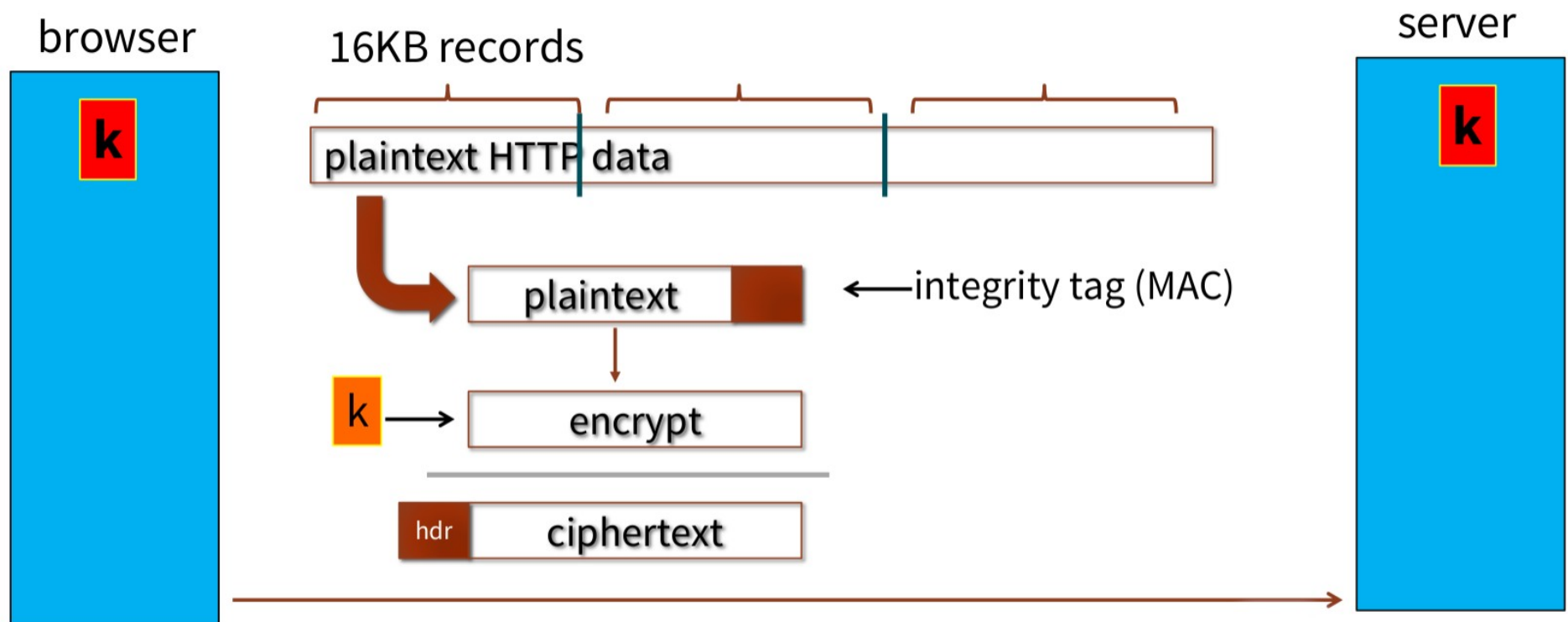
TLS / SSL peut résoudre les problématiques suivantes :

- L'authentification : Le client et le serveur sont sûrs de leur identité (mTLS)
- La confidentialité : Les données échangées même interceptées ne peuvent pas être décryptées
- L'intégrité : Les données échangées ne peuvent pas être altérées

TLS/ SSL : Hand Shake



TLS 1.2



Méthode de cryptage ***MAC-then-encrypt*** :

- La raison de nombreuses attaques sur TLS (BEAST, Lucky13, POODLE, ...)



Apports de TLS 1.3

Encrypt then MAC avec AES128-GCM

Forward secrecy : Même si la clé du serveur est compromise à un instant t .
Elle ne peut pas être utilisée pour décrypter des messages enregistrés auparavant

Option de configuration zéro aller-retour : le client peut envoyer des données chiffrées plus rapidement

Le certificat du serveur est crypté (auparavant, envoyé en clair) : confidentialité renforcée lorsque le serveur a plusieurs certificats

pre-shared key (PSK) : Initier une session TLS à partir d'un secret pré-partagé



OWASP

Introduction

Top10



Introduction

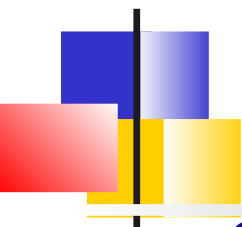
OWASP : Open Web Application Security Project est une organisation à but non lucratif fondée en 2001

Référence incontestée dans le monde de la sécurité des systèmes d'information

Publie des recommandations, méthodes et outils pour sécuriser les applications web et APIs

Les projets les plus connus :

- *Top 10 Projects* : liste des dix risques de sécurité en fonction des applications
 - OWASP Top 10 : Application web traditionnelles
 - OWASP API Security Top Ten : API Rest ou autres
- *WebGoat* : plateforme de formation
- *WebScarab* : proxy d'audits de sécurité.



Projets OWASP

OWASP Amass : Framework OpenSource pour établir une cartographie réseau des surfaces d'attaque et découverte d'actifs externes

OWASP Application Security Verification Standard : définir les contrôles de sécurité requis lors de la conception, le développement et le test d'applications et de services Web.

OWASP Cheat Sheet Series : Guides de bonnes pratiques à suivre pour les développeurs d'applications

OWASP CycloneDX : Full-stack BOM pour identifier les vulnérabilités

OWASP Defectdojo : Un outil pour l'intégration continue de la sécurité

OWASP Dependency-Check : Analyse statique des dépendances et de leurs vulnérabilités

OWASP Dependency-Track : Réduire les risques liés à la chaîne d'approvisionnement SBOM



Projets OWASP (2)

- OWASP Juice Shop** : Application non-secure utilisable pour la formation
- OWASP Mobile Application Security** : Dédié au application mobile
- OWASP ModSecurity Core Rule Set** : Ensemble de règles de détection d'attaques à utiliser avec ModSecurity ou des firewall compatibles
- OWASP Offensive Web Testing Framework** (OWTF) : Unifier les meilleurs outils pour rendre les tests d'intrusion plus efficace
- OWASP Software Assurance Maturity Model SAMM** : Moyen efficace et mesurable pour évaluer l'approche sécuritaire d'une organisation
- OWASP Security Shepherd** : Application web et mobile permettant aux ingénieurs de perfectionner leur test d'intrusion
- OWASP Top Ten** : standard de référence pour les risques de sécurité des applications web les plus critiques
- OWASP Web Security Testing Guide** : ressource pour les tests de cybersécurité pour les développeurs d'applications Web.
- OWASP API Security** (En cours) : Stratégies et solutions pour comprendre et atténuer les vulnérabilités des interfaces de programmation d'applications (API).



CWE

Common Weakness Enumeration ou
CWE est une liste des vulnérabilités
que l'on peut rencontrer dans les
logiciels

Cette liste est maintenue par
l'organisme MITRE



OWASP Top Ten

1. **Broken Access Control** : Un utilisateur accède à des opérations non permises
2. **Cryptographic Failures** : Transmission en clair, algorithmes de cryptography déprécié, faiblesse des clé, stockage des clés dans le dépôt, ...
3. **Injection** : Les données fournies par l'utilisateur ne sont pas désinfectés
4. **Insecure Design** : Mauvaise architecture due à l'incapacité de déterminer le niveau de conception de sécurité requis.
5. **Security Misconfiguration** : Mauvaise configuration, sécurité par défaut
6. **Vulnerable and Outdated Components** : De l'OS aux librairies
7. **Identification and Authentication Failures** : Voler l'identité de quelqu'un d'autre
8. **Software and Data Integrity Failures** : Pas de vérification que les données n'ont pas été modifiées
9. **Security Logging and Monitoring Failures** : Pas de moyen de détecter des attaques ou des brèches
10. **Server-Side Request Forgery** : Récupération d'une ressource distante sans vérifier l'URL demandée



API Security Top 10

1. **Broken Object Level Authorization** : Utilisation d'un identifiant d'un objet non autorisé
2. **Broken Authentication** : Authentification faible permettant à un utilisateur de se faire passer pour quelqu'un d'autre
3. **Broken Object Property Level Authorization** : Une propriété d'un objet que ne devrait pas manipuler un utilisateur
4. **Unrestricted Resource Consumption** : une API n'est pas protégée contre un nombre d'appel excessif
5. **Broken function level authorization** : Contrôles d'autorisation insuffisants sur des fonctions
6. **Unrestricted Access to Sensitive Business Flows** : Une opération métier qui peut être effectuée par un automate
7. **Server Side Request Forgery** : Appel d'un service distant sans vérification de l'URL
8. **Security Misconfiguration** : Mauvaise configuration ouvrant des portes pour les attaques
9. **Improper inventory management** : Exploitations des endpoints obsolètes ou peu sécurisés
10. **Unsafe Consumption of APIs** : Ne pas faire confiance aux APIs que l'on consomme.



OWASP

Introduction

Top 10

01 - Broken Access Control

API 01 – Broken Object Level Authorization

1- Broken Access Control : Contournement des mécanismes d'autorisation pour accéder à un objet non autorisé

Description OWASP :

Le contrôle d'accès applique une politique telle que les utilisateurs ne peuvent pas agir en dehors de leurs autorisations prévues.

Les défaillances entraînent généralement la divulgation non autorisée d'informations, la modification ou la destruction de toutes les données ou l'exécution d'une fonction métier en dehors des limites de l'utilisateur.





Vulnérabilités

Violation du principe du moindre privilège ou refus par défaut. Des ressources oubliées deviennent accessibles

Contournement des contrôles d'accès en modifiant l'URL, l'état interne de l'application, ...

Références direct d'objet non sécurisées :

Accès à l'API avec POST, PUT et DELETE non contrôlé

Élévation de privilège. Agir en tant qu'utilisateur sans être connecté ou agir en tant qu'administrateur en étant connecté en tant qu'utilisateur.

Manipulation des métadonnées : Jeton (JWT), cookie ou champ masqué pour élever les privilèges ou abuser de l'invalidation JWT.

Mauvaise configuration CORS : Accès à l'API à partir d'origines non autorisées/non approuvées.

Forcez la navigation vers des pages authentifiées en tant qu'utilisateur non authentifié ou vers des pages privilégiées en tant qu'utilisateur standard.



Parades

Deny par défaut

Implémentez des mécanismes de contrôle d'accès dans l'ensemble de l'application, minimiser le CORS.

Les contrôles d'accès doivent s'appuyer le propriétaire de l'enregistrement

Désactivez la liste des répertoires du serveur Web et assurez-vous que les métadonnées ne sont pas accessible.

Enregistrez les échecs de contrôle d'accès, alertez les administrateurs.

Limitez le débit autorisé de requêtes pour contrer les outils d'attaque automatisés.

Les identifiants de session doivent être invalidés sur le serveur après la déconnexion. Les jetons JWT doivent avoir un délai d'expiration court .



API 03 : Broken Object Property Level Authorization

Vulnérabilités :

- L'API expose des propriétés d'un objet qui sont considérés comme sensible
- L'API permet de modifier, ajouter, supprimer un attribut sensible d'un objet



Parades

S'assurer que l'utilisateur a accès aux propriétés que l'on expose.

Éviter les méthodes génériques telles que *to_json()* et *to_string()*.

Éviter d'utiliser des fonctions qui lient automatiquement l'entrée d'un client à des propriétés d'objet.

Implémenter un mécanisme de validation de réponse.
L'appliquer transversalement sur tous les points de l'API.

Renvoyer les structures de données minimum.



A05 - Broken Function Level Authorization

Vulnérabilités :

- Un utilisateur simple a accès à des point d'admin
- Un utilisateur peut changer la méthode HTTP (e.g. de GET à DELETE)
- Un utilisateur a accès à des fonctionnalités d'un autre groupe simplement en devinant l'URL



Parades

L'application doit disposer d'un module d'autorisation cohérent et facile à analyser (composant externe)

Il est appliqué transversalement

Parade :

- Deny par défaut
- Sécurisation spécifique admin



02 - Cryptographic Failures

L'application doit déterminer les données sensibles

- Qui transitent
- Qui sont stockées

Des techniques cryptographiques sont alors utilisées mais elles ne sont pas toutes sûres et évoluent





Vulnérabilités

Données transmises en texte clair : HTTP, SMTP, FTP

Algorithmes ou protocoles cryptographiques anciens ou faibles

Clés de chiffrement par défaut, ou faibles, Manque de gestion ou rotation des clés, clés archivées dans les référentiels de code source

Cryptage pas forcé : Entête HTTP manquants

Certificat de serveur et chaîne de confiance non validés

Paramètres d'initialisation ignoré, réutilisé ou généré trop facilement pour les opérations cryptographiques

Mots de passe utilisés comme clé cryptographique directement sans utilisation d'une fonction de dérivation

Fonctions de hachage obsolètes telles que MD5 ou SHA1

Méthodes de remplissage¹ cryptographique obsolètes telles que PKCS v1.5

Messages d'erreur cryptographiques exploitables

1. [https://en.wikipedia.org/wiki/Padding_\(cryptography\)](https://en.wikipedia.org/wiki/Padding_(cryptography))



Parades (1)

Classer les données traitées, stockées ou transmises par une application.

Ne stockez pas de données sensibles inutilement.

Assurez-vous de chiffrer toutes les données sensibles stockées.

S'assurer que les algorithmes, les protocoles et les formats des clés sont en place et à jour ; utiliser une bonne gestion des clés (rotation).

Chiffrez toutes les données en transit avec TLS. Forcer le chiffrement via HTTP Strict Transport Security (HSTS).

Désactivez la mise en cache pour les réponses contenant des données sensibles.

N'utilisez pas de protocoles hérités tels que FTP et SMTP pour le transport de données sensibles.



Parades (2)

Stockez les mots de passe à l'aide de fonctions de hachage comme Argon2, scrypt, bcrypt ou PBKDF2.

Les vecteurs d'initialisation des clés doivent être choisis en fonction du mode de fonctionnement. En général, utiliser un générateur de nombres pseudo-aléatoires. Dans tous les cas, ne jamais utilisé 2 fois le même vecteur.

Utiliser CSPRNG¹ pour générer les nombres aléatoires.

Utilisez toujours un cryptage authentifié au lieu d'un simple cryptage. Cela permet de s'assurer de l'intégrité de la donnée

Évitez MD5, SHA1, PKCS numéro 1 v1.5 .

Vérifier indépendamment l'efficacité de la configuration.

1. https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator

3 - Injection

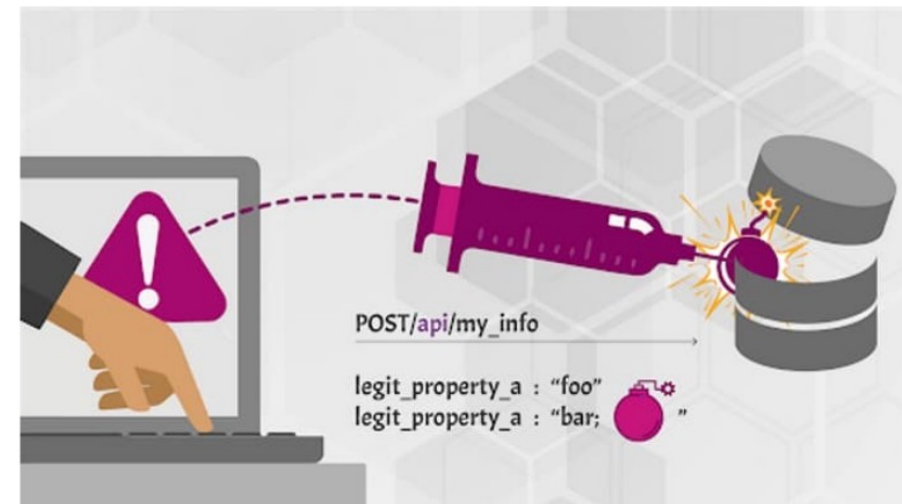
3 - Injection :

Des requête qui incluent des commandes malveillantes

Description : Les injections les plus courantes sont SQL, NoSQL, des commandes du système d'exploitation, le mappage relationnel d'objet (ORM), LDAP et l'injection de langage d'expression (EL) ou de bibliothèque de navigation de graphe d'objet (OGNL).

Le concept s'applique à tous les interpréteur.

L'examen du code source est la meilleure méthode pour détecter si les applications sont vulnérables aux injections.





Vulnérabilités

Les données fournies par l'utilisateur ne sont pas validées, filtrées ou nettoyées par l'application.

Les requêtes dynamiques ou les appels non paramétrés sans échappement contextuel sont utilisés directement dans l'interpréteur.

Les données hostiles sont utilisées dans les paramètres de recherche de mappage objet-relationnel (ORM) pour extraire des enregistrements sensibles supplémentaires.

Les données hostiles sont directement utilisées ou concaténées. Le SQL ou la commande contient la structure et les données malveillantes dans les requêtes dynamiques, les commandes ou les procédures stockées.



Parades

Utiliser une API sécurisée vers l'interpréteur.

Exemple *PreparedStatement*

Utilisez la validation positive¹ des entrées côté serveur.

Pour les requête dynamique restantes, échappez les caractères spéciaux à l'aide de la syntaxe d'échappement spécifique à cet interpréteur.

Utilisez LIMIT et d'autres contrôles SQL dans les requêtes pour empêcher la divulgation massive d'enregistrements en cas d'injection SQL.

1. Pattern identifiant les données valides, plutôt que de blacklister les invalides

4 - Insecure Design

4- Insecure Design : Conception de la sécurité manquante ou inefficace

Description :

Une conception non sécurisée est due généralement au manque d'évaluation des risques métier, et donc à l'incapacité de déterminer le niveau de conception de sécurité requis.

Une conception sécurisée peut toujours présenter des défauts d'implémentations conduisant à des vulnérabilités.

Une conception non sécurisée ne peut pas être corrigée par une implémentation parfaite.





Conception sécurisée

La conception sécurisée est une culture et une méthodologie qui évaluent constamment les menaces et garantissent que le code est conçu et testé de manière robuste pour empêcher les méthodes d'attaque connues.

Elle a une influence sur le cycle de vie de développement



Parades

Utilisation de bibliothèques ou framework qui applique des design pattern sécurisés

Inclure la sécurité dans le cycle de développement

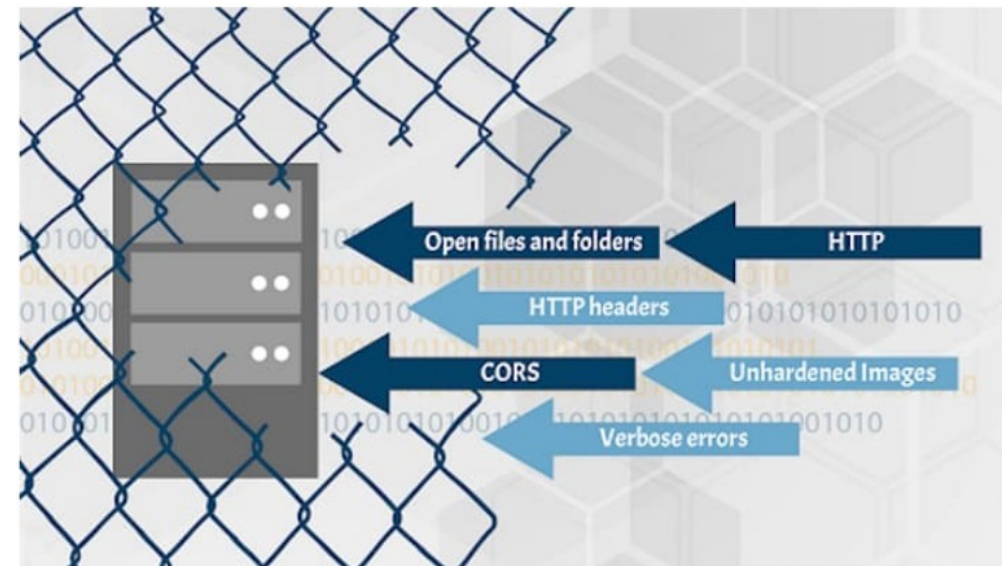
- Modéliser les menaces et leur impact
- Intégrer le langage et les contrôles de sécurité dans les user stories
- Ecrire des tests unitaires et d'intégration validant les réponses aux menaces
- Intégrez des contrôles à chaque couche de l'application (du frontend au backend)

Séparez en couche le système et le réseau en fonction des besoins d'exposition et de protection

Limiter la consommation de ressources par utilisateur ou service

5-Security Misconfiguration

Une mauvaise configuration des serveurs, des frameworks ou de l'API permet aux attaquants de les exploiter





Vulnérabilités

Renforcement de la sécurité des composants manquant ou autorisations mal configurées sur les services cloud.

Fonctionnalités inutiles activées (par exemple, des ports, des services, des pages, des comptes ou des privilèges inutiles).

Configuration par défaut toujours valide (compte par défaut par exemple).

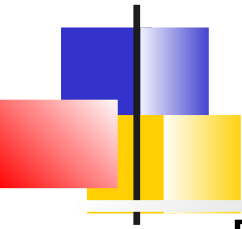
Erreurs trop informatives pour les utilisateurs.

Pour les systèmes mis à niveau mais dernières fonctionnalités de sécurité pas appliquées.

Les paramètres de configuration de la sécurité dans les serveurs d'applications, les frameworks d'applications les bases de données, etc., n'ont pas des valeurs sécurisées.

Le serveur n'envoie pas d'en-têtes ou de directives de sécurité.

Les composants ne sont pas à jour.



Parades

Déploiement reproductible : Les env de développement, de QA et de production configurés de manière identique mais avec des informations d'identification différentes.

Une plate-forme minimale sans fonctionnalités inutiles.
Supprimez ou n'installez pas les fonctionnalités et les frameworks inutilisés.

Examiner et mettre à jour les configurations relatives à toutes les notes de sécurité, mises à jour et correctifs dans le cadre du processus de gestion des correctifs .

Une architecture d'application segmentée qui fournit une séparation efficace et sécurisée entre les composants.

Envoi de directives de sécurité aux clients, par exemple, des en-têtes de sécurité.

6 - Vulnerable and Outdated Components

6- Vulnerable and Outdated Components :

Utiliser des composants obsolètes ou ayant des vulnérabilités





Vulnérabilités

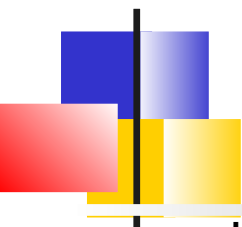
Ne pas connaître les versions de tous les composants que l'on utilise.

Inclure un logiciel vulnérable, non pris en charge ou obsolète. (OS, Serveur, (SGBD), librairies).

Ne pas rechercher régulièrement les vulnérabilités, ne pas s'abonner aux bulletins de sécurité liés aux composants que l'on utilise.

Ne pas corriger ou ne pas mettre à niveau la plate-forme sous-jacente, les infrastructures et les dépendances en temps opportun et en fonction des risques.

Les développeurs de logiciels ne testent pas la compatibilité des bibliothèques mises à jour, mises à niveau ou corrigées.



Parades

Un processus de gestion des correctifs doit être en place pour :

- Supprimer les dépendances inutilisées, les fonctionnalités inutiles, les composants, les fichiers et la documentation.
- Inventorier en permanence les versions des composants côté client et côté serveur et leurs dépendances à l'aide d'outils tels que versions, OWASP Dependency Check, retire.js, etc.
- Surveiller en permanence les sources telles que Common Vulnerability and Exposures (CVE) et National Vulnerability Database (NVD).
- Utiliser des outils logiciels d'analyse. Abonnez-vous aux alertes par e-mail pour les vulnérabilités de sécurité liées aux composants que vous utilisez.
- Obtenir les composants à partir des sources officielles via des liens sécurisés. Préférez les packages signés pour réduire le risque d'inclure un composant malveillant modifié
- Surveiller les bibliothèques et les composants qui ne sont pas maintenus ou qui ne créent pas de correctifs de sécurité pour les anciennes versions.

7 - Identification and Authentication Failures

7 - Identification and Authentication Failures : authentification faible permettant à un utilisateur de se faire passer pour quelqu'un d'autre

Description :

La confirmation de l'identité de l'utilisateur, l'authentification et la gestion de la session sont essentielles pour se protéger contre les attaques liées à l'authentification.





Vulnérabilités

Autoriser les attaques automatisées telles que le *credential stuffing*, où l'attaquant dispose d'une liste de noms d'utilisateur et de mots de passe valides.

Permettre la force brute ou d'autres attaques automatisées.

Autoriser les mots de passe par défaut, faibles ou bien connus, tels que "Mot de passe1" ou "admin/admin".

Utiliser des processus de récupération de mot de passe oublié faible, tels que les « réponses basées sur les connaissances ».

Stocker les mots de passe en clair ou faiblement hachés

Ne pas avoir d'authentification multifacteur.

Exposer l'identifiant de session dans l'URL.

Réutiliser l'identifiant de session après une connexion réussie.

Ne pas invalider correctement les identifiants de session.



Parades

Authentification multi-facteur (2FA)

Mettre en place des contrôles de mots de passe faibles.

Aligner les politiques de longueur, de complexité et de rotation des mots de passe¹

S'assurer que la création de compte, la récupération de créden-tiels, l'authentification utilisant des messages génériques quelque soit l'issue.

Limiter ou retarder les tentatives de connexion infructueuses, sans créer de scénario de déni de service. Enregistrer tous les échecs et alertez les administrateurs lorsque des attaques sont détectées.

Utiliser un gestionnaire de session intégré, sécurisé qui génère un nouvel ID de session aléatoirement. L'identifiant de session ne doit pas figurer dans l'URL, être stocké en toute sécurité et invalidé après la déconnexion ou l'inactivité.

8- Software and Data Integrity Failures

8 - Software and Data Integrity Failures : Code ou infrastructure qui ne protège pas contre les violations d'intégrité

Description : Exemple : Les attaquants téléchargent leurs propres mises à jour qu'ils exécutent sur les installations. Un autre exemple est celui où des objets ou des données sont encodés ou sérialisés dans une structure qu'un attaquant peut voir et modifier.





Vulnérabilités

Mise à jour de code ou de librairie
automatique sans vérification
d'intégrité

Données encodée ou sérialisée qu'un
attaquant peut modifier



Parades

- Utiliser des signatures numériques pour vérifier la provenance et la non altération des composants.

- S'assurer que les dépendances consomment des référentiels (Maven, npm, ...) approuvés.

- S'assurer qu'un outil de vérification de dépendance s'applique en continu (OWASP Dependency Check ou OWASP CycloneDX)

- Reviewing de code

- S'assurer que la pipeline CI/CD dispose d'un contrôle d'accès appropriés pour garantir l'intégrité du code.

- S'assurer que les données sérialisées échangées n'ont pas été altérées via des contrôle d'intégrité ou de signature numérique.

9 - Security Logging and Monitoring Failures

9-Security Logging and Monitoring

Failures : Le manque de monitoring, de surveillance et d'alertes appropriées permet aux attaquants de passer inaperçus.

Description :

Sans journalisation ni surveillance, les violations ne peuvent pas être détectées. Une journalisation, une détection, une surveillance et une réponse active insuffisantes se produisent à tout moment





Vulnérabilités

Les login/logout ou transaction importante ne sont pas consignés.

Les avertissements et les erreurs génèrent des messages de journal inexistants, inadéquats ou peu clairs.

Les journaux ne sont pas surveillés pour détecter toute activité suspecte.

Les journaux ne sont stockés que localement.

Les seuils d'alerte appropriés et les processus d'escalade des réponses ne sont pas en place ou efficaces.

Les tests d'intrusion et les analyses par les outils de test dynamique de sécurité des applications (DAST) (tels que OWASP ZAP) ne déclenchent pas d'alertes.

L'application ne peut pas détecter, escalader ou alerter des attaques actives en temps réel ou quasi réel.



Parades

S'assurer que tous les échecs de connexion, de contrôle d'accès et de validation des entrées sont consignés avec un contexte suffisant.

S'assurer que les journaux sont générés dans un format facilement utilisable par les solutions de gestion des journaux.

S'assurer que les données du journal sont correctement codées pour empêcher les injections.

S'assurer que les transactions de grande valeur disposent d'une piste d'audit avec des contrôles d'intégrité pour empêcher la falsification ou la suppression.

Les équipes DevSecOps doivent mettre en place une surveillance et des alertes efficaces afin que les activités suspectes soient détectées et traitées rapidement.

Établir ou adopter un plan de réponse aux incidents et de récupération.

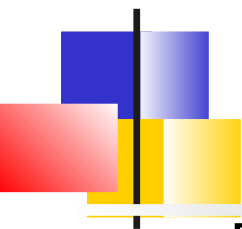
10 - Server-Side Request Forgery (SSRF)

10 - Server-Side Request Forgery (SSRF) : une application Web récupère une ressource distante sans valider l'URL fournie par l'utilisateur

Description

Permet à un attaquant de contraindre l'application à envoyer une requête spécialement conçue vers une destination inattendue, même lorsqu'elle est protégée par un pare-feu, un VPN ou un autre type de liste de contrôle d'accès (ACL) réseau.





Parades

Réseau :

- Segmenter la fonctionnalité d'accès aux ressources distantes dans des réseau séparé pour réduire l'impact de SSRF
- Appliquez des politiques de pare-feu « de refus par défaut » ou des règles de contrôle d'accès au réseau pour bloquer tout le trafic intranet non nécessaire

Couche applicative :

- Assainir et valider toutes les données d'entrée fournies par le client
- Appliquer le schéma, le port et la destination de l'URL avec une liste d'autorisation positive
- Ne pas envoyer de réponses brutes aux clients
- Désactiver les redirections HTTP lors de l'appel de ressource distante



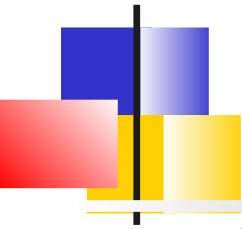
Batir une application secure

Architecture de la sécurité

Code contre les attaques

Revue de sécurité

Découverte et gestion de vulnérabilités



Segregation of code in logical components

- Makes code maintainable
- Easy to incorporate change
- Easy to build security controls



Inconvénients et vulnérabilités

Pas de possibilité de révoquation : un jeton reste valide jusqu'à son expiration

- Si un compte utilisateur doit être bloqué, il faudra attendre que le jeton expire pour que le blocage soit effectif
- Si un utilisateur veut changer son mot de passe, le jeton déjà généré reste effectif
- Même si le jeton est supprimé du navigateur à la fin de la session, il reste valide jusqu'à sa date d'expiration

None Alg : possibilité de générer des token JWT ni chiffré ni signé (algorithme none)

=> Danger si l'implémentation de la vérification du token accepte l'algorithme « none »

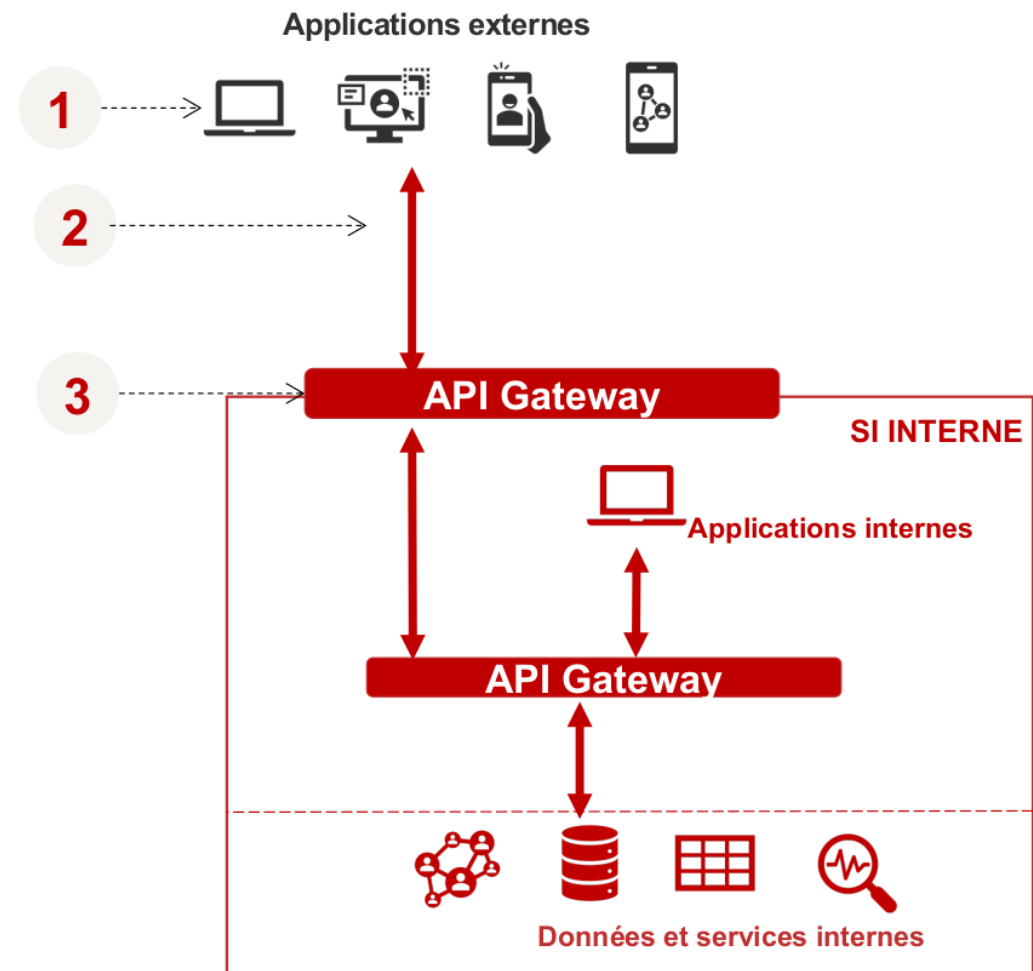
Alg swapping : Certaines implémentations peuvent être vulnérables à une attaque qui consiste à utiliser la valeur HS256 pour la propriété alg.

- Si le serveur attend un jeton signé par RSA et qu'il reçoit un jeton signé par HMAC, il considérera la clé publique comme une clé HMAC

Provenances des attaques

Les attaques peuvent se produire à 3 niveaux

1. Au niveau des applications consommatrices des APIs
 - Vol des credentials, ...
2. Au niveau d'une API
 - Données sensibles, injection de code, ...
3. Au niveau Infra/plateforme API Gateway
 - Accès aux fonctionnalités d'administration, ...





Comment y remédier ?

1 . Un socle technique solide

- Sécuriser l'infra/plateforme API Gateway
- Apporter un socle technique commun avec des solutions d'authentification éprouvées

2 . De la sécurité by Design

- Classifier les API selon leurs usages (APIs internes, partenaires, open)
- Se poser la question de la sensibilité des données lors de la conception d'une API

3 . Des équipes sensibilisés

- Sensibiliser les équipes aux enjeux sécuritaires
- Les former aux socles techniques/outils



Socle Technique

Portail d'entrée

Le **WAF (Web Application Firewall)** protège le SI des attaques par

- Dénis de services
- Injection de code (XML, Json, SQL,...)
- Cross-Site Scripting (XSS)
- XML External Entities (XXE)
- Attaque par force brute

Il permet également le filtrage IP et fait office de terminaison SSL/TLS entre l'extérieur et l'API Gateway

HTTPS pour les échanges avec l'extérieur



Socle Technique

Une Gateway sécurisée

Une API Gateway configurée pour un maximum de sécurité

- Rate Limiting
- Gestion des quotas
- Validation des entrées/sorties

Des solutions communes éprouvées d'identification/authentification

- Contrôle d'accès des applications
- Authentification des utilisateurs
- Habilitations fonctionnelles des utilisateurs



La sécurité by Design

Gestion des données sensibles

1. Séparation des environnements
Séparation stricte des environnements de test et de production
2. Stockage et gestion des secrets
Les secrets doivent être stockés et communiqués d'une façon sécurisée (Chiffrement des données).
3. Sécurité by Design
 - Seules les données nécessaires doivent être communiquées dans les réponses
 - Adapter le niveau de sensibilité des données au public cible



Exemples de bonnes pratiques

Identifiant non devinables : Il est recommandé que les Ids des ressources soient non devinables

- éviter les incrémentations par défaut.

Gérer les exceptions d'une façon à masquer les environnements backend et les technologies utilisées

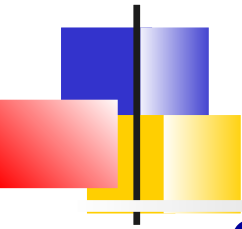
S'interdire l'utilisation de cookies



Surveillance

Installez un système d'alarmes :
Composant de surveillance basé sur de
l'intelligence artificielle pour

- connaître les comportements du trafic
- Détecter les anomalies
- Lever les alerte
- Ou bloquer automatiquement les menaces



CORS : Le « Cross-origin resource sharing » (CORS) permet d'ouvrir son API.

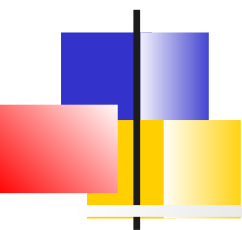
=> Gérer les hôtes ou domaines autorisés

CSRF (Cross-Site Request Forgery) : Concerne plus les applications Web. Les jetons des APIs Rest protège des CSRF

Anti-farming : Protection des données contre le scarping en détectant les patterns d'une ou plusieurs adresses IP

Rate-limiting : Protection l'utilisation d'une API en limitant le nombre de requêtes simultanées (secondes/minutes)

Throttling : Limiter le nombre de requêtes qu'une application peut faire dans une période donnée (heures/jours/Mois/Années), i.e. Quota



Batir une application secure

Architecture de la sécurité

Code contre les attaques

Revue de sécurité

Découverte et gestion de vulnérabilités



XSS

Règle majeure : Interdire que toute entrée utilisateur soit rendu directement dans le DOM

En javascript, c'est utiliser *innerText* plutôt que *innerHTML*

```
const userString = '<strong>hello, world!</strong>';  
const div = document.querySelector('#userComment');  
div.innerText = userString; // safe  
div.innerHTML = userString // not Safe
```



XSS : Désinfection des données d'entrée

Quelquefois on a besoin d'un sous ensemble de tags .

Par exemple : `` mais pas `<script>`

Il faut donc désinfecter l'entrée en supprimant les tags que l'on ne veut pas.

Mais ce n'est pas simple, il faut en général s'appuyer sur des librairies ou frameworks :

- *DOMSanitizer* d'Angular est un exemple



XSS : DOMParser

DOMParser permet de générer une représentation DOM d'une string.

Il est alors plus simple d'itérer sur tous les éléments et faire ses tests sur le tag, de construire un autre DOM puis l'injecter comme innerHTML

```
const parser = new DOMParser();  
const html =  
    parser.parseFromString('<script>alert("hi");</script>`');  
// itérer sur html.children
```




SVG

Les API comme Blob et SVG comportent des risques importants en tant que récepteurs dynamique, car elles stockent des données arbitraires tout en étant capables d'exécuter du code.

=> Leur utilisation est donc risquée

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
  <circle cx="250" cy="250" r="50" fill="red" />
  <script type="text/javascript">console.log('test');</script>
</svg>
```



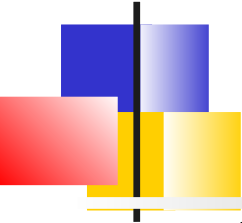
XSS : Désinfecter les hyperliens

Si l'on veut insérer des liens dans le DOM, on peut s'appuyer sur la désinfection des balises `<a>` par tous les navigateurs

```
<button onclick="goToLink()">click me</button>
const userLink = "<strong>test</strong>";
const goToLink = function() {
window.location.href = `https://mywebsite.com/${userLink}`;
// => https://my-website.com/<strong>test</strong>
};
```

Plutôt comme cela :

```
const goToLink = function() {
const dummy = document.createElement('a');
dummy.href = userLink;
window.location.href = `https://mywebsite.com/${dummy.a}`;
// => to: https://my-website.com/%3Cstrong%3Etest%3C/strong
};
```



XSS : CSS

Si l'application permet aux utilisateurs de saisir du CSS, cela peut devenir un vecteur d'attaque XSS

Par exemple : l'attribut *background:image*

Pour éviter ce vecteur d'attaque :

- Ne pas autoriser l'upload de CSS
- Autoriser uniquement la modification de champs spécifiques par l'utilisateur et générer soi-même la CSS
- Désinfecter les attributs CSS (plus dur)



XSS : Content Security Policy

CSP (Content Security Policy) est un outil de configuration de sécurité pris en charge par tous les principaux navigateurs.

Concernant les attaques XSS, il peut permettre de définir quels scripts externes peuvent être chargés, où ils peuvent être chargés et quelles API DOM sont autorisées à exécuter le script.

Il ne protège pas des DOM XSS

2 façons de configurer CSP :

- Via l'entête **Content-Security-Policy**
- Via **<meta-tag>**



Exemples CSP

Charger des scripts de l'origine et d'une seule URL

```
Content-Security-Policy: script-src "self"  
https://api.mega-bank.com
```

Images chargées de n'importe quel site, media de *example.org* et *example.net* et les scripts de *userscripts.example.com*

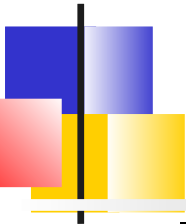
```
Content-Security-Policy: default-src 'self'; img-src *;  
media-src example.org example.net; script-src  
userscripts.example.com
```



CSRF : CSRF Token

La défense la plus efficace contre les attaques CSRF est le jeton anti-CSRF, appelé ***CSRF-Token***.

1. Le serveur envoie un jeton généré via la cryptographie
2. Chaque requête renvoie le jeton qui est vérifié quant à son authenticité et son expiration
3. Comme le jeton CSRF est unique pour chaque User, l'attaquant qui arrive à récupérer un jeton doit viser un seul utilisateur et *espérer* qu'il click avant son expiration



Méthodes pour éliminer CSRF

Il y a de nombreuses méthodes pour éliminer ou minimiser le risque d'attaque CSRF.

Les plus courantes sont :

- Faire en sorte que le requête **GET** ne soit que **stateless**
C'est les requêtes les plus simples pour les attaques CSRF
- Mise en place de défenses CSRF s'appliquant à toute l'application en utilisation un middleware par exemple. (Ex : *SpringSecurity*)



XXE : Protection

Il est facile de se défendre de XXE en désactivant les entités dans votre parseur XML.

```
factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
```

On peut aussi changer de format et choisir JSON



Injection SQL

Il n'est pas difficile de se protéger des injections SQL.

Les **Prepared Statement**, présent dans la plupart des langages backend, compilent d'abord la requête SQL puis associe ses paramètres aux variables provenant éventuellement de l'utilisateur. L'opération compilée préalable, ne peut pas être détournée.

Les bases de données ont également des protections spécifiques pour échapper les caractères dangereux.
Ex : *mysql_real_escape_string()*



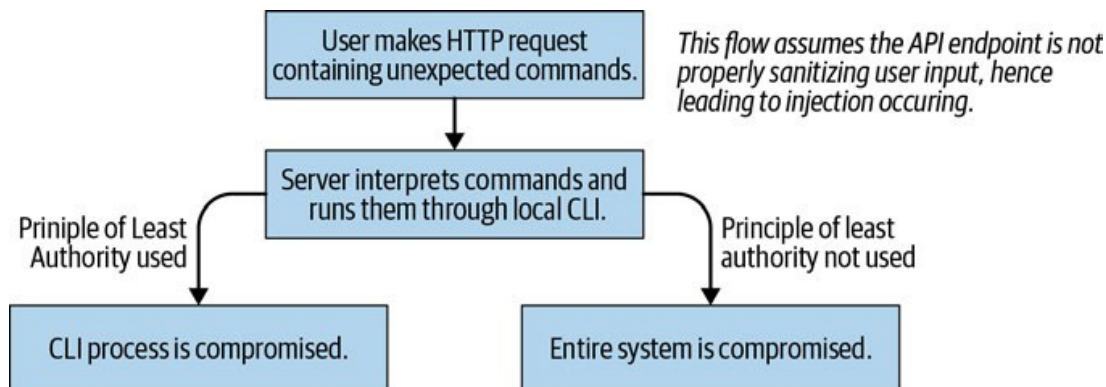
Autres injection

Identifier les cibles pour l'injection :

- Task schedulers, Bibliothèques de compression, Script de backup, Loggers, Appel système, Interpréteur ou compilateur

Implémenter des contrôles à partir de **white list** des commandes permises

Appliquer le principe de **least privilege** pour réduire l'impact d'une attaque





Regex DoS

Les *regex* malicieuses devraient être détectées lors de l'audit de code ou par les outils d'analyse statique

Ne pas donner la possibilité aux utilisateurs d'entrer ses propres regex



Autre Dos

Les DoS logiques sont beaucoup plus difficile à détecter.

- Aucune application peut se targuer de ne pas avoir de vulnérabilité Dos.
- Pour se protéger, on doit identifier le code qui utilise beaucoup de ressources système effectuer des tests de performance.

Distributed DoS : Le moyen le plus simple de se défendre est d'investir dans un service de gestion de bande passante : Web Application Firewall.



Gestion de dépendances

Pour se prévenir des vulnérabilités des dépendances, il faut être capable de générer l'arbre de dépendances et de leur version.

C'est que font les outils comme Maven ou npm

Comparer ces arbres aux CVE publics via des outils ou plugins spécialisés



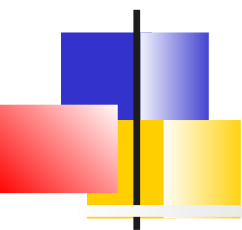
Npm fixer les versions de l'arbre de dépendances

npm inclut par défaut le caractère `^` de vant toute dépendance.

Si on le supprime, la dépendance utilisera la version exacte plutôt que le dernier patch.

Cependant cela ne protège pas des dépendances transitives.

La commande ***npm shrinkwrap*** crée un fichier `npm-shrinkwrap.json` qui fixe les versions des dépendance directes et transitives



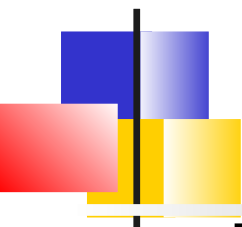
Batir une application secure

Architecture de la sécurité

Code contre les attaques

Revue de sécurité

Découverte et gestion de vulnérabilité



Revue de code

Presque tous les experts en sécurité s'accordent sur le fait que rien ne peut remplacer l'examen du code.

De nombreux problèmes de sécurité involontaires mais importants sont extrêmement difficiles à découvrir avec d'autres formes d'analyse ou de tests, tels que les tests d'intrusion.

Nécessite une expertise sécurité parmi les développeurs¹

Attention : le code déployé n'est souvent pas complètement identique, n'a pas accès au code compilé des librairies

La revue de code est complétée par des outils d'analyse statique de vulnérabilité

- Ex Sonar, CheckMarx
- dependencies check



Quand

Effectuer les revue de code de sécurité à chaque Merge Request

L'examen de la sécurité du code vérifie les vulnérabilités courantes tels que XSS, CSRF, injection, etc., mais plus important encore, il vérifie les vulnérabilités au niveau logique qui nécessitent une connaissance approfondie du contexte.



Comment démarrer

1. Évaluez le code côté client pour mieux comprendre la logique métier et les fonctionnalités utilisables.
2. Evaluer la couche API et sur quelles dépendance elle s'appuie.
3. Suivre les dépendances dans la couche API, en examinant attentivement les bases de données, les libraires, les fonctions de journalisation, etc.
4. Essayez de trouver toutes les API publiques qui pourraient être involontairement exposées.
5. Continuez sur le reste du code.



Détection d'anti-pattern

Blacklist : Ne sont efficace que si on a complète connaissance de l'appli.

```
const blacklist = ['http://www.evil.com', 'http://www.badguys.net'];
```

```
const isDomainAccepted = function(domain) {  
  return !blacklist.includes(domain);  
};
```

Boilerplate Code : Usage de configuration par défaut d'un framework

Trust-By-Default : Une application a le droit d'écrire en base, écrire dans le journal et lire le disque. Un seul compte pour toutes ces opérations. Et si une vulnérabilité est exploité, l'attaquant pourra tout faire

Mauvaise Séparation Client/Serveur : Logique d'authentification dans le client



Test de sécurité lors du développement

Au niveau développement, définir des suites de tests unitaire qui valide les exigences positives et négatives de contrôle de sécurité :

- Identité, authentification et contrôle d'accès
- Validation et encodage des entrées
- Chiffrement
- Gestion des utilisateurs et des sessions
- Gestion des erreurs et des exceptions
- Audit et journalisation

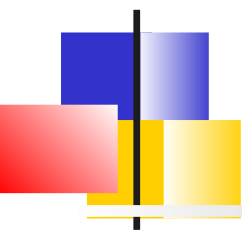


Tests d'intégration et de validation

L'environnement d'intégration est le premier environnement dans lequel les testeurs peuvent simuler des scénarios d'attaque réels par un utilisateur externe ou interne malveillant.

Ils peuvent être effectués manuellement ou par des outils de pénétration

Ils peuvent s'appuyer sur des checklist



Batir une application secure

Architecture de la sécurité

Code contre les attaques

Revue de sécurité

**Découverte et gestion de
vulnérabilité**



Introduction

Pour compléter le code review, il faut un processus de découverte des vulnérabilités qui ciblent le code de production

Il est typiquement intégré dans une pipeline continue

Il comprend :

- L'analyse statique
- L'analyse dynamique
- Les tests de non régression



Outils d'analyse statique

Quelques outils :

- Checkmarx (la plupart des langages)
- Sonarqube
- Bandit (Python—free)
- Brakeman (Ruby—free)

Ils détectent généralement :

- Reflected XSS, DOM XSS
- SQL Injection
- CSRF
- DoS

Ils fonctionnent mieux avec les langages statiquement typé

Ne sont pas efficace pour les vulnérabilités impliquant une connaissance approfondie des applications, un chaînage de vulnérabilités ou des vulnérabilités dans des langages typés dynamiquement.



Analyse dynamique

L'analyse dynamique nécessite l'exécution de code

=> plus coûteuse et plus lente.

Détecte de vraies vulnérabilités à la différence des analyses statiques

Outils :

- HCL AppScan payant
- Burp Suite
- ZAP (OWASP)



Test de non régression

Une suite de tests de régression de vulnérabilité est simple.

La vulnérabilité ayant été déjà détectée, elle doit garantir qu'elle ne soit pas à nouveau publiées.

Utilise les frameworks de test habituels :

- Jest
- Junit, RestAssured



Autres moyens de découverte

Offrir la possibilité aux utilisateurs finaux de reporter des bugs ou vulnérabilités facilement et sans risque

Bug Bounty Programs : Offrir des récompenses aux personnes trouvant des vulnérabilités dans son produit

S'adresser à un société spécialisée



Gestion des vulnérabilités

Les vulnérabilités détectées doivent suivre un workflow de traitement :

- Reproduire la vulnérabilité dans un environnement de staging afin de vérifier si ce n'est pas un faux positif
- Évaluer sa sévérité en lui donnant un score
- Évaluer ce que l'on va en faire, le trier
- La fixer



Common Vulnerability Scoring System (CVSS)

CVSS¹ est un système OpenSource pour classer les vulnérabilités en fonction de leur facilité d'exploitation et du type de données ou de processus qui peuvent être compromis.

Il offre 3 score:

- *Base* : La vulnérabilité seule. Le plus couramment utilisé
- *Temporal* : son évolution dans le temps
- *Environmental* : En prenant en compte l'environnement



Calculateur

Base Metrics ?

Exploitability Metrics

Attack Vector (AV):	<input checked="" type="radio"/> Network (N)	<input type="radio"/> Adjacent (A)	<input type="radio"/> Local (L)	<input type="radio"/> Physical (P)
Attack Complexity (AC):	<input checked="" type="radio"/> Low (L)	<input type="radio"/> High (H)		
Attack Requirements (AT):	<input checked="" type="radio"/> None (N)	<input type="radio"/> Present (P)		
Privileges Required (PR):	<input checked="" type="radio"/> None (N)	<input type="radio"/> Low (L)	<input type="radio"/> High (H)	
User Interaction (UI):	<input checked="" type="radio"/> None (N)	<input type="radio"/> Passive (P)	<input type="radio"/> Active (A)	

Vulnerable System Impact Metrics

Confidentiality (VC):	<input type="radio"/> High (H)	<input type="radio"/> Low (L)	<input checked="" type="radio"/> None (N)
Integrity (VI):	<input type="radio"/> High (H)	<input type="radio"/> Low (L)	<input checked="" type="radio"/> None (N)
Availability (VA):	<input type="radio"/> High (H)	<input type="radio"/> Low (L)	<input checked="" type="radio"/> None (N)

Subsequent System Impact Metrics

Confidentiality (SC):	<input type="radio"/> High (H)	<input type="radio"/> Low (L)	<input checked="" type="radio"/> None (N)
Integrity (SI):	<input type="radio"/> High (H)	<input type="radio"/> Low (L)	<input checked="" type="radio"/> None (N)
Availability (SA):	<input type="radio"/> High (H)	<input type="radio"/> Low (L)	<input checked="" type="radio"/> None (N)



Tester la sécurité

Introduction

Techniques de reconnaissance

Test de pénétration

ZAP



Techniques

Tout ne peut pas être testé, il faut donc faire des compromis

Différentes techniques doivent être appliquées pour tester la sécurité globale d'une application :

- Tester régulièrement l'implication des personnes, des process et des politiques vis à vis de la sécurité.
Le formaliser par des documents
- Modélisation des menaces
- Tests unitaires, d'intégration, de non régression
- Revue du code
- Tests de pénétration



Modèle de menace

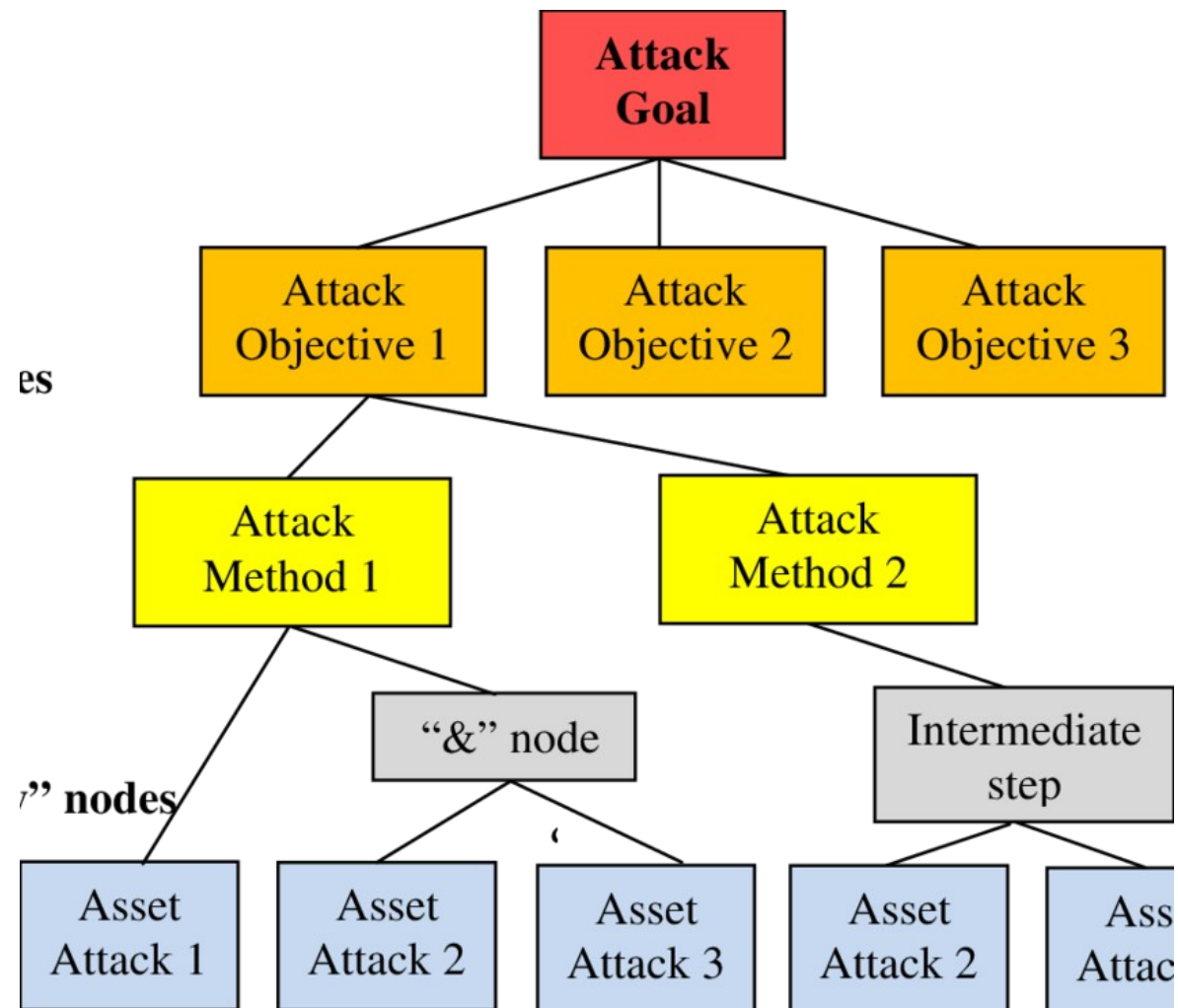
La norme NIST 800-30 pour l'évaluation des risques peut être utilisée pour bâtir son modèle de menace. Cette approche implique :

- Décomposition de l'application : Comprendre le fonctionnement de l'application, ses actifs, ses fonctionnalités et sa connectivité.
- Définir et classer les actifs : Classez les actifs en fonction de leur importance commerciale.
- Explorer les vulnérabilités potentielles - qu'elles soient techniques, opérationnelles ou de gestion.
- Explorer les menaces potentielles – développez une vision réaliste des vecteurs d'attaque potentiels du point de vue d'un attaquant.
- Créer des stratégies d'atténuation – développer des contrôles d'atténuation pour chacune des menaces jugées réalistes.

Le résultat de la modélisation est en général des graphes et des listes.

Des produits OSS ou commerciaux aide à bâtir ces modèles

Graphe de menace





Test de pénétration

Les tests de pénétration sont des test black-box qui concerne toute la stack.

Très efficace au niveau du réseau et de l'infrastructure, ils peuvent avoir être moins performant au niveau d'une application web développée sur mesure

En général, un test d'intrusion ne peut identifier qu'un petit échantillon représentatif de tous les risques de sécurité possibles dans un système



Analyse des résultats et reporting

La définition des objectifs des métriques et mesures des tests de sécurité est une condition préalable à l'utilisation des résultats des tests.

Le reporting doit inclure :

- une catégorisation de chaque vulnérabilité par type ;
- la menace de sécurité à laquelle chaque problème est exposé ;
- la cause première de chaque problème de sécurité, comme le bug ou la faille ;
- la technique de test utilisée pour trouver les problèmes ;
- la remédiation, ou contre-mesure, pour chaque vulnérabilité et le temps pour l'implémenter
- l'indice de gravité de chaque vulnérabilité (par exemple, élevé, moyen, faible ou score CVSS¹).



Outils de détection

Commercial : propose en général la gestion de vulnérabilité en plus

- Appspider : <https://docs.rapid7.com/appspider/>
- Invicti : <https://www.invicti.com/>
- Acunetix : <https://www.acunetix.com/>
- Burpsuite : <https://portswigger.net/burp>
- Codename SCNR

OpenSource

- Zed Attack Proxy (OWASP)
- WATOBO <https://watobo.sourceforge.net/index.html>



Tester la sécurité

Introduction

Techniques de reconnaissance

Test de pénétration

ZAP



Introduction

Que ce soit le hacker ou la personne qui teste la sécurité, le premier travail consiste à rassembler les informations sur l'application visée.

L'objectif étant de construire une cartographie de la structure, de l'organisation et des fonctionnalités de l'application.



Travail de reconnaissance

Un hacker effectue en général un travail de reconnaissance permettant d'identifier sa surface d'attaque.

A partir d'infos publics (1 site web par exemple), il essaye :

- D'identifier des sous-domaines
- Analyser les APIs
- Identifier les dépendances tierces utilisées
- Identifier les points faibles



Techniques de reconnaissance

Recherche de sous-domaine



Techniques de recherche de sous-domaine

Exploration du site web manuellement ou via des crawlers (zad par exemple)

Cache des moteurs de recherche

`site:mega-bank.com -inurl:www password`

Recherche dans les réseaux sociaux. Twitter API :

`https://api.twitter.com/1.1/search/tweets.json`

Zone Transfer Attacks :

- Identifier le DNS du domaine visé
`nslookup mega-bank.com`
- Lui demander de transférer ses enregistrements en se faisant passer par un autre DNS serveur
`host -l mega-bank.com ns1.mega-bank.com`

Brute force : Essayer des combinaisons au hasard

Dictionnary : Utiliser un dictionnaire avec le terme les plus utilisés

Utiliser dnscan : <https://github.com/rbsec/dnscan>



Générer une liste de sous-domaines

```
/* Un exemple générant une liste de domaines à tester*/
const generateSubdomains = function(length) {

  const charset = 'abcdefghijklmnopqrstuvwxyz'.split('');
  let subdomains = charset;
  let subdomain;
  let letter;
  let temp;
  /*
  * Time Complexity:  $O(n*m)$ , n:longueur de la chaîne
  */
  for (let i = 1; i < length; i++) {
    temp = [];
    for (let k = 0; k < subdomains.length; k++) {
      subdomain = subdomains[k];
      for (let m = 0; m < charset.length; m++) {
        letter = charset[m];
        temp.push(subdomain + letter);
      }
    }
    subdomains = temp
  }
  return subdomains;
}
const subdomains = generateSubdomains(4);
```



Brute force

```
const dns = require('dns');
const promises = [];
const subdomains = [];
/*
 * Iterer sur la liste des sous-domaine et effectuer une query DNS asynchrone
 */
subdomains.forEach((subdomain) => {
  promises.push(new Promise((resolve, reject) => {
    dns.resolve(`${subdomain}.mega-bank.com`, function (err, ip) {
      return resolve({ subdomain: subdomain, ip: ip });
    });
  }));
});

// Logger les résultats positifs
Promise.all(promises).then(function(results) {
  results.forEach((result) => {
    if (!!result.ip) {
      console.log(result);
    }
  });
});
```



Techniques de reconnaissance

Découverte de l'API



Découverte des endpoints

Identifier des appels REST ou SOAP :

```
GET /users/1234/payments HTTP/1.1
Host: api.mega-bank.com
Authorization: Bearer abc21323
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/1.0 (KHTML, like Gecko)
```

A partir d'un endpoint connu,

- Tenter le verbe OPTIONS

```
curl -i -X OPTIONS https://api.mega-bank.com/users/1234
```

Réponse :

```
200 OK
Allow: HEAD, GET, PUT, DELETE, OPTIONS
```

- Utiliser un script et faire de la brute discovery



Résultat de l'exploration

Le résultat de l'exploration permet de cartographier les requêtes et les réponses fournies par l'application

- Requêtes :
 - Distinction Verbe : GET/POST
 - Paramètres des POST, Paramètres cachés
 - Query String des GET
 - Entêtes personnalisées
- Réponses :
 - Cookie positionné
 - Code retour en particulier redirection
 - Entêtes

Ce travail est facilité par l'utilisation d'un proxy



Gray-box testing

Si l'on a accès aux sources, l'outil
OWASP Attack Surface Detector peut
faciliter le travail

```
java -jar attack-surface-detector-cli-  
1.3.5.jar <source-code-path> [flags]
```




Déterminer la charge utile

Après avoir identifié les endpoints, le hacker doit identifier la payload associé.

Certains sont standards : exemple demande de jeton *oAuth* à un serveur

https://discordapp.com/api/oauth2/authorize?response_type=code&client_id=157730590492196864&scope=identify%20guilds.%20join&state=15773059ghq9183habn&redirect_uri=https%3A%2F%2Fnicememe.website&prompt=consent



Payload non standard

Les payload non standard sont plus difficile à déterminer pour un hacker.

Il peut s'appuyer sur des messages d'erreur. Ex :
`400 auth_token not supplied`

=> Rester générique sur les messages d'erreur

Si le hacker est un utilisateur de l'appli, il peut analyser ses propres requêtes et essayer de les jouer pour un autre utilisateur.

Si le hacker connaît le nom d'une variable et certaines contraintes, il peut réduire son espace de solution utilisé par les techniques de brute force



Découvertes des applications

Recherche des applications hébergées sur le serveur web. En général, à partir d'une IP

- Différents contextes sur le même serveur

`http://www.example.com/url1`

`http://www.example.com/url2`

Tests manuels ou brute force

- Ports non standard

Utilisation de nmap pour les découvrir

- Hôtes virtuels

DNS Zone transfer

Search engine



Détection des mécanismes d'authentification

La détection du mécanisme d'authentification est généralement simple.

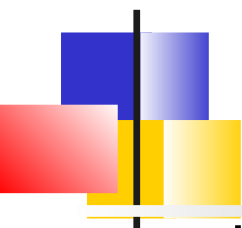
Accès à la page de login :

- Inspection des entêtes
Http authentication
- Inspection de l'URL
Form based authentication
OpenID



Techniques de reconnaissance

Identification des technologies



Identification du serveur

Le hacker/testeur cherche à récupérer le nom et la version du serveur

- Regarder l'entête Server d'une réponse
Server: Apache/2.4.41 (Unix)
- Regarder l'ordre des entêtes
- Envoyer de mauvaise requête et espérer récupérer la page par défaut

Il peut utiliser des outils automatisés :

- Netcraft.
- Nikto
- Nmap, avec sa GUI, Zenmap.

On peut rendre les choses plus difficiles :

–



Identification des dépendances

Toutes les applis utilisent des dépendances OpenSource (du code moins testé en termes de sécurité)

Des **Common Vulnerabilities and Exposures (CVE)** sont publiés dans des bases publiques.

Si le hacker parvient à savoir quelle dépendance OpenSource est utilisée, il peut exploiter une CVE



Dépendances clientes

Technologies clientes :

- Bibliothèques SPA (Angular, React, Vue, Ember)
- librairies de base (Lodash, JQuery)
- frameworks CSS (Bootstrap, Bulma).

Généralement simple à détecter en regardant le DOM

Si le hacker arrive à identifier le numéro de version, il peut trouver les vulnérabilités exposées sur le Web



Dépendances serveur

Plus difficiles à trouver. Cependant le hacker peut quelquefois s'appuyer :

- Sur les entêtes
 - X-Powered-By: ASP.NET
 - Server: Microsoft-IIS/4.5
 - X-AspNet-Version: 4.0.25
- Les pages d'erreur par défaut
- Le nom du cookie
- L'extension de l'URL (.jsp, .do, .php, etc..)
- Les erreurs Bds remontés directement au client

Il pourra cloner le dépôt github du framework OpenSource pour essayer d'identifier précisément le n° de version en fonction des réponses



Techniques de reconnaissance

Identification des points faibles



Identification des points faibles

La phase de reconnaissance a permis au hacker de consigner :

- Les technologies utilisées dans l'application Web
- La liste des points de terminaison d'API par verbe HTTP et leur payload
- Les fonctionnalités incluses dans l'application Web (par exemple, commentaires, authentification, notification)
- Les domaines utilisés par l'application Web
- Les configurations trouvées (par exemple, politique de sécurité du contenu ou CSP)
- Les systèmes d'authentification/gestion de session

L'étape d'après est alors d'identifier les points faibles



Identification des points faibles

Identifier si la sécurité est architectural ou implémenter point par point

Identifier si la sécurité est présente à toutes les couches

Identifier si l'appli a adopté des techniques standards ou a implémenter elle-même ces fonctionnalités de sécurisation (algorithme de hash par exemple)



Signaux d'un design Insecure

Les paramètres de requête sont utilisés pour identifier une méthode métier d'une classe

Les paramètres de requêtes permettent d'identifier un rôle

Les Uis non autorisées sont juste cachées

Sécurité implémentée au niveau de la vue mais pas de la couche métier



Tester la sécurité

Introduction
Techniques de reconnaissance
Étendue des tests
ZAP



Introduction

L'étendue des tests de sécurité est très vaste.

Ils peuvent être exécutés en mode :

- Black Box : Se mettre à la place du hacker
- Gray Box : Etre guidé par l'implémentation

Ils peuvent être automatisés par des outils de pénétration spécialisés

Mais ce outils doivent être configurés et pilotés manuellement



Etendue des tests

Les différents types de test :

- Configuration : Réseau, Infra, Serveur
- Gestion des identités : Création/suppression de compte, ...
- L'authentification
- Les autorisations
- La gestion de session
- La validation des entrées
- La gestion des erreurs
- Le cryptage
- La sécurisation des fonctions métier
- Les tests client



Configuration de la plateforme

S'assurer que les configuration par défaut ne sont pas appliquées et les fichiers connus ont été supprimés.

Vérifier qu'aucun code de débogage ou de fichiers non nécessaires ne soient laissés sur la production.

Faire la chasse aux fichiers avec des extensions sensibles (scripts, données brutes, créidentiels), vérifier leur permission

Passez en revue les mécanismes de journalisation

Vérifier les méthodes HTTP autorisées, éliminer celles qui sont inutiles (TRACE, OPTIONS, ..)

Vérifier l'usage d'https : entête HTTP Strict Transport Security (HSTS)

Énumérer et Vérifier la configuration des sous-domaines

Vérifier la configuration des stockage en ligne (AWS, ...)



Gestion des identités

Processus d'inscription :

- Manuellement ou Http Proxy

Création, suppression de compte

- Manuellement ou Http Proxy

RBAC :

- Burp's Authorize extension, ZAP's Access Control Testing add-on

Collecte de username :

- Vérifier la généricité des messages d'erreur retournés par l'authentification ou la recovery de compte .
- ZAP, curl, JMeter



Authentication

Force des crédeniels et stratégie des mots de passe :

- Outils : Burp Intruder, THC Hydra, Nikto 2

Verrouillage de compte

- Manipulation de l'entête X-Forwarded
- Manuellement

Contournement de l'authentification

- Accès direct à des pages protégées, Modification de paramètre (Ex. authenticated=true), Prédiction de Session ID, Injection SQL
- Outils : ZAP

Vérifier que les crédeniels ne sont pas stockés côté client

Vérifier que les entêtes *Cache-control* sont bien positionnées pour les données sensibles.

Cache-Control: no-cache, no-store

Expires: 0

Pragma: no-cache

- Outil : ZAP



Autorisation

Contournement des autorisations horizontales ou verticales¹

- Outils : ZAP avec extension Access Control, Burp avec extensions Authorize et AuthMatrix

Elevation de privilèges

- Paramètre ayant une influence sur le rôle, sur l'autorisation d'une action ?, Faible Session ID

Référence directe d'objet

- Outils : Zap

Autorisation sur propriété de l'objet

- Manuellement

1. Un utilisateur peut avoir accès aux données d'un utilisateur du même rôle

Un utilisateur peut avoir accès aux données d'un rôle supérieur



Gestion de session

Force des cookies :

- Énumérer les cookies et leur moment de création, Analyse leur structure et leur côté aléatoire, Test de brute force
- Outils : ZAP, Burp Sequencer

Attributs des cookies

- Vérification des attributs : *Secure, Http-Only, Domain, Path, Expire, SameSite, Prefixes*
- Outils :
 - Proxies : ZAP, Web Proxy Burp Suite
 - Plug-in Navigateur : Tamper Data for FF Quantum, “FireSheep”, “Cookiebro - Cookie Manager” pour FireFox, “EditThisCookie” pour Chrome

Fixation de session

- Vérifier que le cookie est changé après authentification



Gestion de session (2)

Tests CSRF

- Outils : ZAP, CSRF Tester, Pinata-csrf-tool

Tests du logout :

- Vérifier que le cookie est bien invalidé après le logout
- Tiemout
- Plus compliqué si SSO
- Outils : BurpSuite Repeater



Validation des entrées

XSS¹ :

- Outils : Hack Vector, XSS Proxy, PHP Charset Encoder(PCE), Burp et ZAP

Pollution de paramètres

- consiste à fournir plusieurs fois le même paramètre HTTP
- Outils : ZAP active scanning

Injection SQL

- Outils : SQL Map, JSQL injection

Injection d'entête Host :

- Jouer avec les entêtes Host et X-Forwarded-Host et évaluer les impacts

Injection dans gabarit HTML

- Backslash Powered Scanner Burp / Bonne performance pour les injections en général

SSRF

- Identifier les points d'injection SSRF

EX : GET <https://example.com/page?page=about.phphttps://example.com/page?page=about.php>



Test du traitement d'erreur

Objectifs :

- Identifier et analyser les sorties retournées par les erreurs.

Tests :

- Serveurs Web : 404, 403, Requêtes qui ne respecte pas HTTP RFC : chemin trop long, mauvaise entête, mauvaise version
- Application : Fuzz chaque point d'entrée



Cryptographie

Configuration TLS

- Voir guide Mozilla Server Side TLS¹
- Outils : Nmap, OWASP O-Saft, sslscan, sslyze, SSL Labs, testssl.sh

Cryptage faible

- Algorithme dépassé, mauvaise génération d'un bon algorithme
- Outils :
 - Scanners faible pour les protocoles SNMP, TLS, SSH, SMTP, etc. : Nessus, Nmap, OpenVAS
 - Analyse statique de code : klocwork, Fortify, Coverity, CheckMarx

1. https://wiki.mozilla.org/Security/Server_Side_TLS

2. https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html



Sécurité métier

Exemple : Une opération métier nécessite les étapes 1,2,3. Que se passe t il si l'on passe directement de 1 à 3

Ce type de vulnérabilité ne peut pas être détecté par des scanners (Pas très bon pour les séquences)



Tests métier

Validation des données de la logique métier : Front-end, back-end

Tests des contrôles d'intégrité

Vérifier le timeout des opérations métiers

- Les temps d'exécution peuvent donner des indications à l'attaquant

Vérifier le throttling des opérations métier

- Défense contre l'utilisation abusive

Tests de contournement de flow

Test de l'upload



Tests clients

DOM Based XSS, en particulier celles qui permettent d'exécuter des scripts

- Plus facile en white box

Redirection d'URL ou manipulation de chemin de ressource

- Vérifier window.location
- La construction des URLs de lien

Injection de CSS, de hash

- Evaluer la présence de point d'injection

Vérification du CORS

- Entêtes

Vulnérabilité au clickjacking¹

1. https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html



Tests clients

Test des Web socket

- Outils : OWASP Zed Attack Proxy (ZAP), WebSocket Client, Google Chrome Simple WebSocket Client

Test de l'utilisation de postMessaging

Test du stockage navigateur

- Données sensible
- Outils : Plugins navigateur

Vérification des cross-scripts inclusion

- Vérification des scripts importés



Tester la sécurité

Introduction
Techniques de reconnaissance
Étendue des tests
ZAP



Introduction

Zed Attack Proxy (ZAP) est un outil de test d'intrusion gratuit conçu pour tester des applications Web.

Il est à la fois flexible et extensible grâce à des plugins.

Il se place entre le navigateur et le serveur afin d'intercepter les messages et éventuellement les modifier. Cela nécessite une configuration du navigateur ou de l'outil de test fonctionnel

Il s'exécute en standalone ou en daemon



Mode d'exécution

Zap agit comme une vraie attaque et peut donc causer des dommages sur le système.

Il peut s'exécuter en 4 modes:

- Safe : aucune opération potentiellement dangereuse n'est autorisée.
- Protected : Actions (potentiellement) dangereuses que sur les URL comprises dans le contexte.
- Standard : ne restreint rien.
- ATTAQUE : les nouveaux nœuds du contexte sont activement scannés dès qu'ils sont découverts.

Il est recommandé d'utiliser le mode protégé pour vous assurer que vous attaquez uniquement les sites que vous souhaitez.

Certaines opérations ne sont pas possibles dans les modes safe et protected en dehors du contexte : Active Scanning, Spridering, Fuzzing, ...



Scan automatique

Quick Start est un add-on Zap installé automatiquement.

Il permet d'explorer l'application cible avec un spider afin de faire un plan de l'appli.

Ensuite, le scanner actif attaque toutes les pages, fonctionnalités et paramètres découverts.

Limitations :

- N'accède pas aux pages nécessitant une authentification
- Pas de contrôle sur la séquence d'exploration ni sur les types d'attaque



Exploration manuelle

L'exploration manuelle est cependant nécessaire afin de saisir des données d'entrée permettant d'explorer l'entièreté du site.

Lors de l'exploration manuelle, Zad se comporte comme un proxy. Il scanne passivement les requêtes et réponses , continue à remplir l'arbre du site et enregistre les alertes de vulnérabilités

Une nouvelle fonctionnalité HUD permet d'accéder aux fonctionnalités de Zap directement dans le navigateur



Session

Le travail dans s'effectue dans le cadre d'une session

Les sessions peuvent être sauvegardées sur le disque

Une session a

- Des propriétés générales : description, règles d'exclusion d'URLs
- 1 ou plusieurs contextes qui groupent un ensemble d'URLs et qui configurent les propriétés de Zap



Contextes et scopes

Les **contextes** permettent de grouper un ensemble d'URL.

A priori un contexte correspond à une application web constituant le système que l'on teste.

Ex : 1 contexte SPA, 1 contexte backe-end

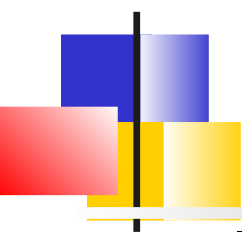
Un **scope** est l'ensemble des URLs que l'on teste. Il est défini à partir des contextes

Souvent 1 scope = 1 contexte

Dans un contexte, on peut définir des *custom page* permettant de classifier les réponses.

Ex : 200 mais avec le message

"Sorry, we can't seem to find what you were looking for"



Configuration des contextes

Pour chaque contexte, on peut configurer :

- Des règles d'inclusion/exclusion
- L'authentification
- La gestion de la session utilisateur
- Le spider Ajax
- Des utilisateurs
- Alert filters : Surcharger le niveau de risque des alertes
- ...



Utilisateurs

Pour chaque contexte, un ensemble d'utilisateurs peut être défini, qui peuvent ensuite être utilisés dans des actions liées au contexte.

- Exploration via Speeder
- Rejouer une requête
- ...

Ceci est étroitement lié aux concepts de gestion de session et d'authentification.

Lorsqu'un Utilisateur est utilisé pour la première fois quelque part dans ZAP, une authentification est effectuée (selon la Méthode d'Authentification définie pour le Contexte) et une Session est créée et configurée pour cet utilisateur (selon la Gestion de Session définie pour le Contexte).



Arbre du site

L'arborescence des sites est très importante car elle reflète la compréhension de ZAP de la structure de l'application.

Les URLs

<https://www.example.com/app/aaa?ddd=eee>

<https://www.example.com/app/aaa?ddd=fff>

<https://www.example.com/app/bbb?ddd=eee>

Produisent 2 entrée dans l'arborescence :

- <https://www.example.com>

- *app*

- *GET:aaa(ddd)*

- *GET:bbb(ddd)*

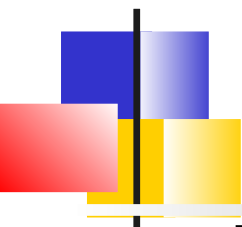
Des modificateurs structurels peuvent être définis dans un contexte, pour modifier l'arbre du site :

- **Data Driven Content** : Lorsque des données sont passées par le chemin de l'URL

- Ex API : /api/{companyName}/orders

- **Structural Parameters** : Les paramètres indiquent la structure de l'application et pas des données utilisateur.

- Ex SPA : /app/aaa?**page=p1**&ddd=eee



Scanning

Passive scanning : Par défaut tous les messages HTTP sont scannés en toile de fond.

Le scan passif génère des alertes et peut ajouter automatiquement des tags aux requêtes

Il existe un écran de configuration pour configurer le scan:

Options → Passive Scanner

Pour configurer les règles d'attributions des tags

Active Scanning : Recherche de vulnérabilités en utilisant des attaques connues.

Configurable via des scan policy qui définissent quelles règles doivent s'exécuter



Autres fonctionnalités

API : Sur localhost:8080

Access Control Testing : Tester l'accès non autorisé à des parties de l'application web

Automation Framework : Automatiser des jobs Zap

Client Side Integration : Extension Firefox ou Chrome permettant de surveiller les événements navigateur

Fuzzing : Permet d'envoyer beaucoup de données sur une URL avec des payload prédéfinies ou scripté

Breakpoints : Intercepter une requête et l'éditer



Scripts

Des scripts peuvent être embarqués. Support du JSR 223 => Javascript, Groovy, Kotlin, Python, ...

De nombreux exemples sont fournis dès que l'on installe les add-ons

Différents types de script :

- Standalone scripts : S'exécute à la demande et accède aux données de Zap (History, Arborescence, etc.)
- Scripts ciblés : S'exécute sur une URL. Par exemple extraire les commentaires HTML
- Http Sender et Proxy : Permet de modifier les messages transitant par le proxy. Il faut les activer pour qu'il s'exécute
- Active / Passive rules : Si activé s'exécute lors des scans
- Authentication : Invoqué lors de l'authentification



Reporting



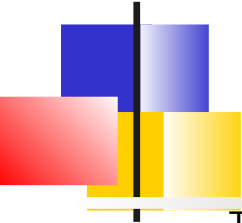
Automatisation

Il est possible démarre zap sans l'UI

- Avec l'option -daemon
- Avec l'option -cmd

On peut ensuite lui indiquer :

- -quickurl : L'application à attaquer
- -quickout : le fichier résultat (HTML/XML/Json)
- -quickprogress : Afficher la barre de progression
- -session : Lui indiquer une session
- -config : Indiquer un fichier de configuration
- Plein d'options sont disponibles voir ./zap.sh -h



Automatisation docker + OpenID

```
TOKEN=$(curl -ks -X POST \  
  ${URL}realms/${REALM}/protocol/openid-connect/token \  
  -H "Accept: application/json" \  
  -H "Content-Type: application/x-www-form-urlencoded" \  
  -H "cache-control: no-cache" \  
  -d "grant_type=password" \  
  -d "username=${USER_NAME}" \  
  -d "password=${USER_PASS}" \  
  -d "client_id=admin-cli"|jq -r .access_token|tr -d "\n"  
)  
  
docker run --net net_oia \  
  --name zap \  
  -v $(pwd):/zap/wrk/:rw \  
  -e ZAP_AUTH_HEADER="Authorization" \  
  -e ZAP_AUTH_HEADER_VALUE="Bearer $TOKEN" \  
  -t owasp/zap2docker-stable zap-full-scan.py -l -j -m 10 -T 60 \  
  -t http://apidb:8080/api/ \  
  -r report-$(date +%Y%m%d-%H%M%S).html
```



Résumé sur l'utilisation

Dans le cas d'une application web peut connue, on peut utiliser le spider pour la découverte des URLs

Sinon, exploration manuelle, configuration de l'authentification, définition d'utilisateur puis spider avec les utilisateurs

Une fois avoir effectué le spider, revoir l'arborescence du site et éventuellement, appliquer des modifcateurs structurels

Définir une configuration de règles et démarrer un active scanning



Modèles de sécurité

Introduction

SSL/TLS

Sécurité stateful

Sécurité stateless

oAuth2/OpenId Connect

JWT

Sécurisation des applications REST

Architecture micro-service



Rôles du protocole

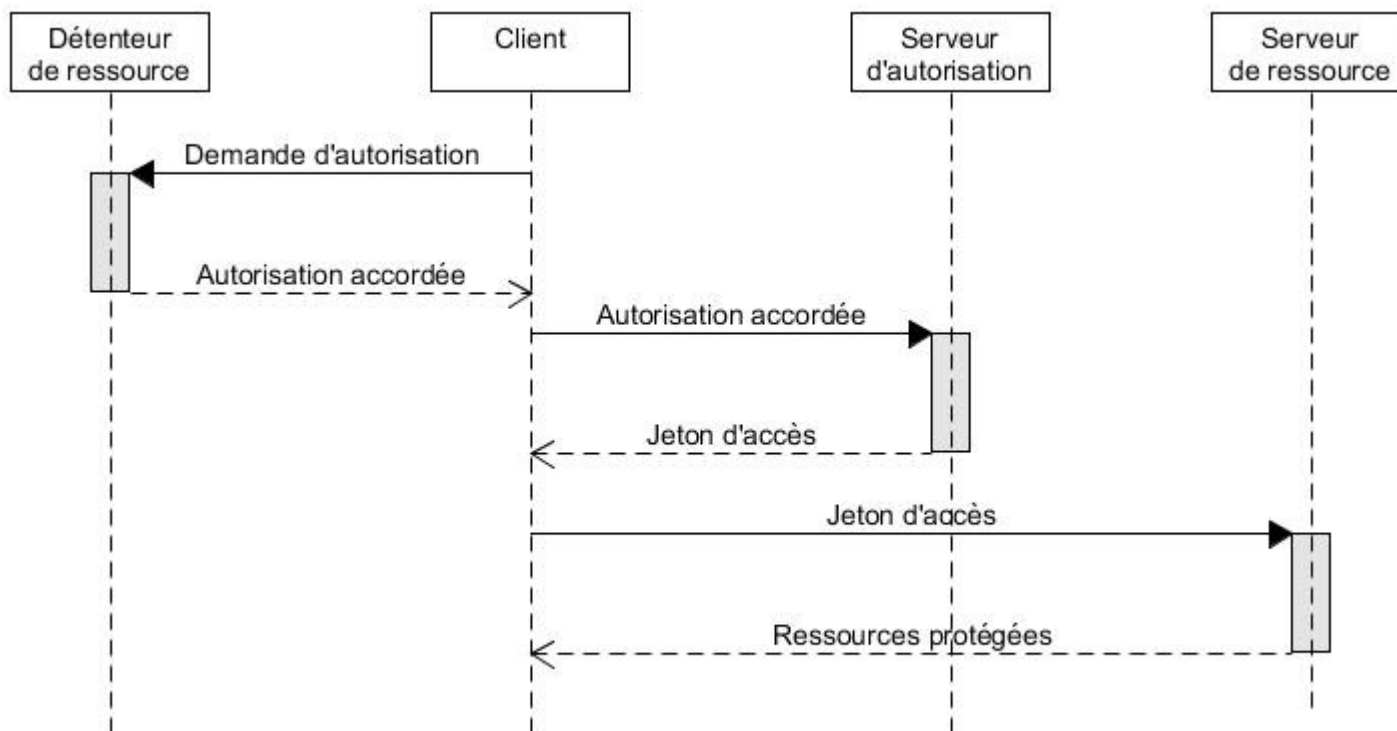
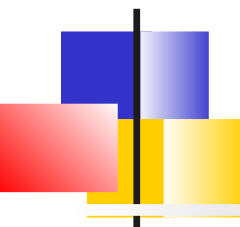
Le **Client** est l'application qui essaie d'accéder au compte utilisateur. Elle a besoin d'obtenir une permission de l'utilisateur pour le faire.

Le **serveur de ressources** est l'API utilisée pour accéder aux ressources protégées

Le **serveur d'autorisation** est le serveur qui autorise un client à accéder aux ressources en lui fournissant un jeton. Il peut demander l'approbation de l'utilisateur

L'utilisateur est la personne qui donne accès à certaines parties de son compte

Rq: Un participant du protocole peut jouer plusieurs rôles





Scénario

1. Pré-enregistrer le client auprès du service d'autorisation (=> client ID et un secret)
2. Obtenir l'autorisation de l'utilisateur.
(4 types de grant)
3. Obtention du token (date d'expiration)
4. Appel de l'API pour obtenir les informations voulues en utilisant le token
5. Validation du token par le serveur de ressource



Enregistrement du client

Le protocole ne définit pas comment l'enregistrement du client doit se faire mais définit les paramètres d'échange.

Le client doit fournir :

- **Application Name**: Le nom de l'application
- **Redirect URLs**: Les URLs du client pour recevoir le code d'autorisation et le jeton d'accès
- **Grant Types** : Les types d'autorisations utilisables par le client
- **Scopes** : paramètre utilisé pour limiter les droits d'accès d'un client
- **Javascript Origin** (optionnel): Le host autorisé à accéder aux ressources via *XMLHttpRequest*

Le serveur répond avec :

- **Client Id**:
- **Client Secret**: Clé devant rester confidentielle

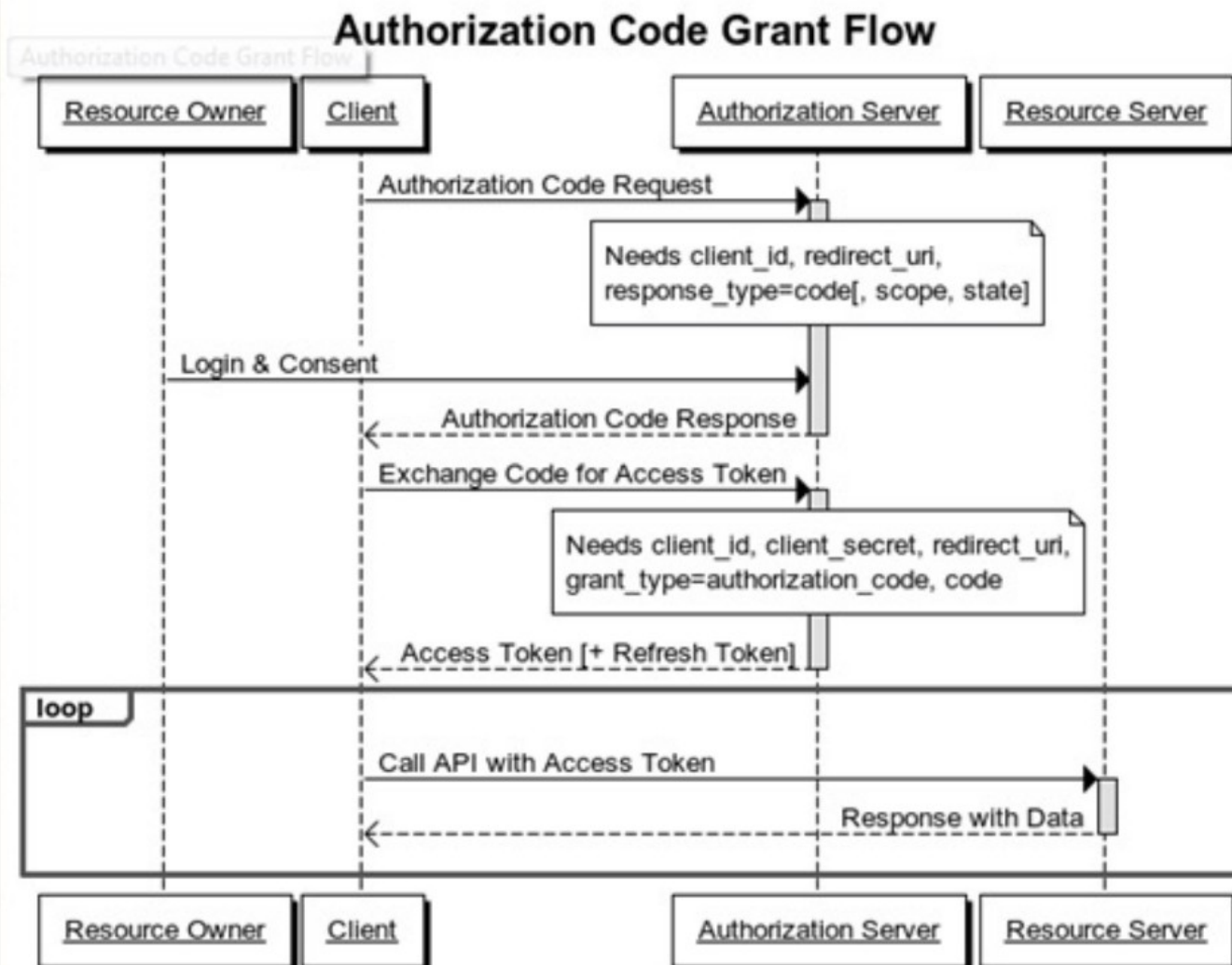


OAuth2 Grant Type

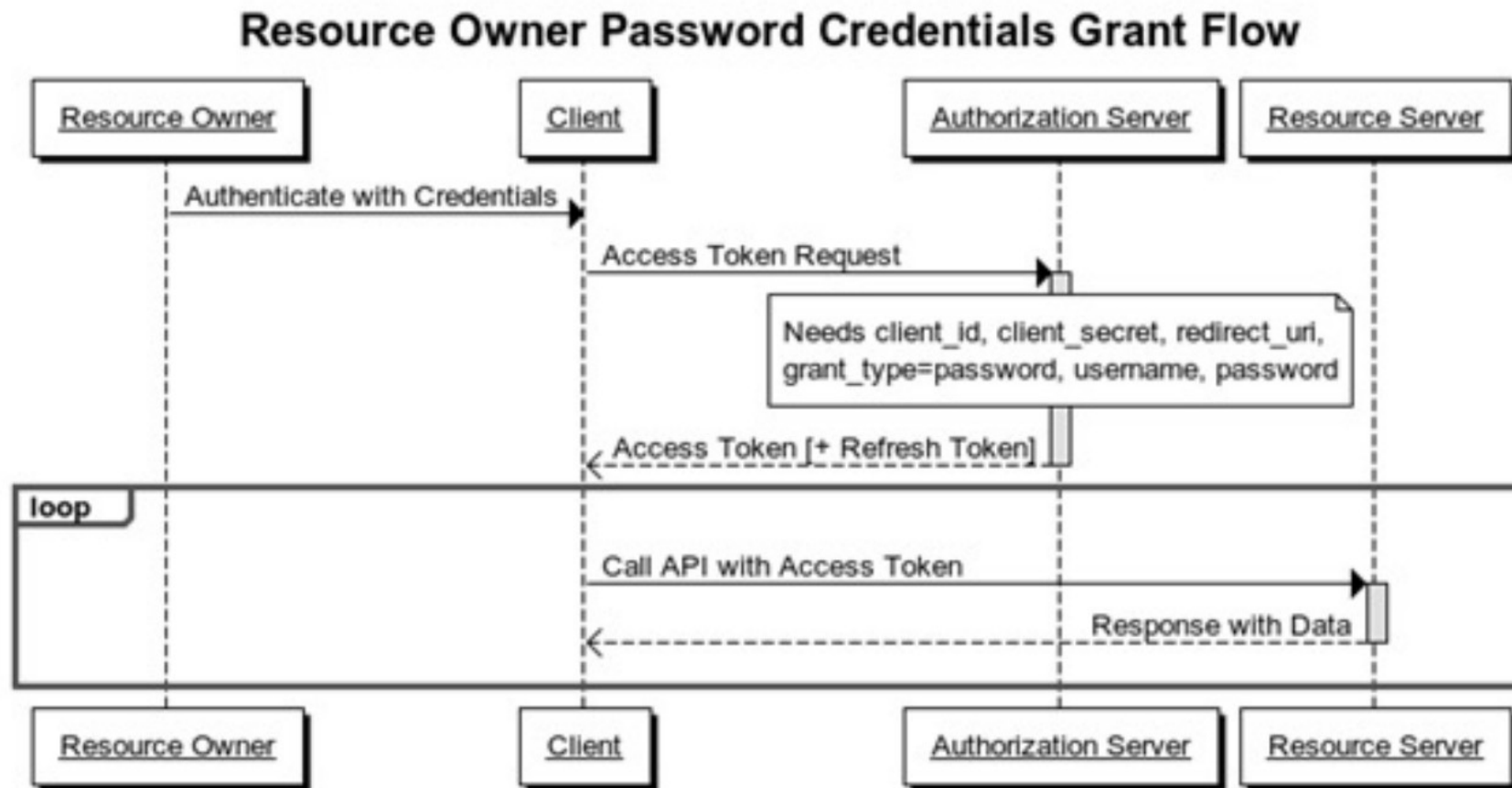
Différents moyens afin que l'utilisateur donne son accord : les **grant types**

- ***authorization code*** : Approbation de l'utilisateur sur le serveur d'autorisation et échange d'un code d'autorisation avec le client
- ***implicit*** : Jeton fourni directement. Certains serveurs interdisent de mode
- ***password*** : Le client fournit les créidentiels de l'utilisateur
- ***client credentials*** : Les créidentiels client suffise

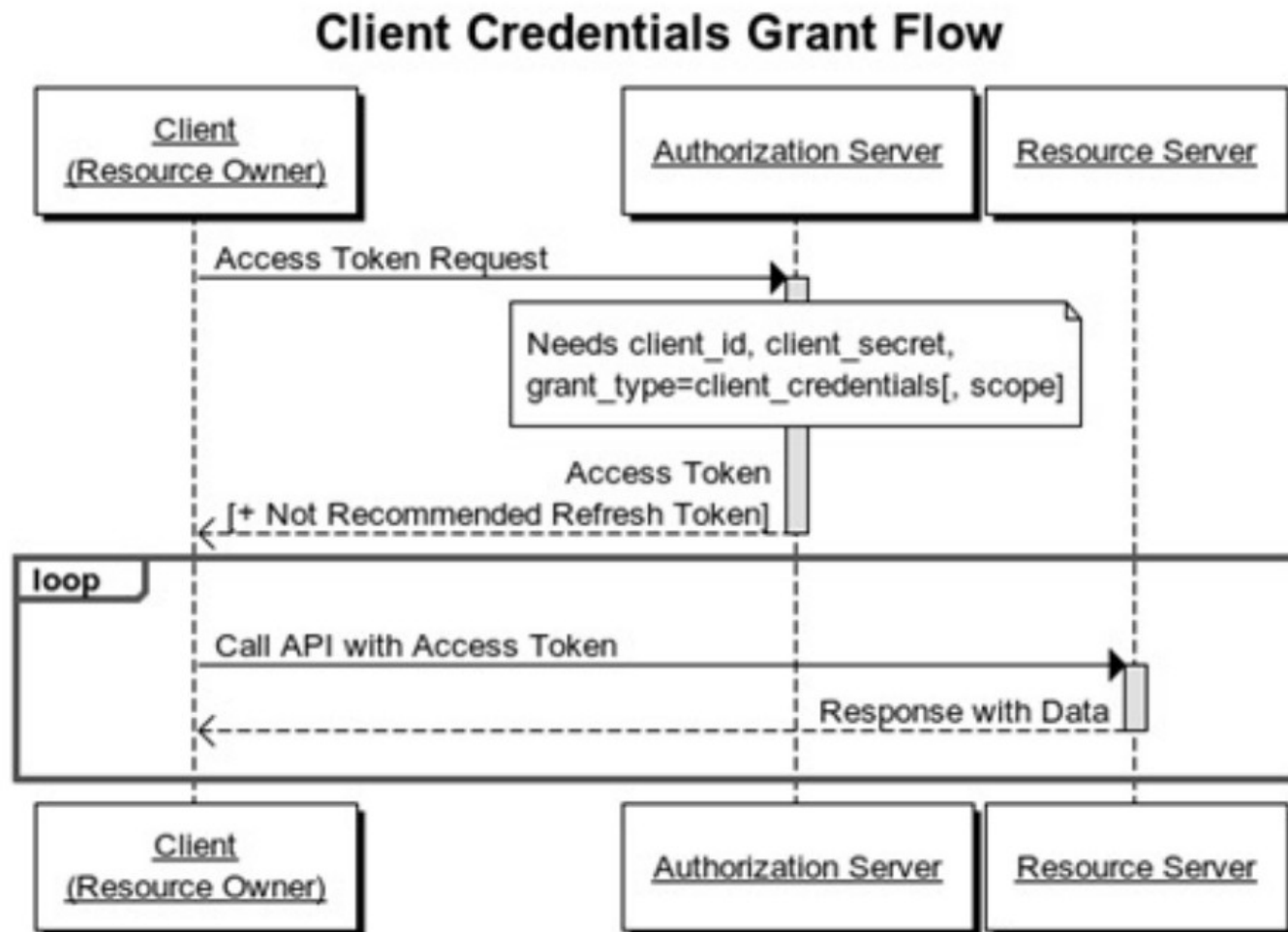
Authorization Code



Password Grant



Client Credentials





Tokens

Les Tokens sont des chaînes de caractères aléatoire générées par le serveur d'autorisation

Les jetons sont ensuite présents dans les requêtes HTTP et contiennent des informations sensibles => HTTPS

Il y a 2 types de token

- Le **jeton d'accès**: Il a une durée de vie limité.
- Le **Refresh Token**: Délivré avec le jeton d'accès. Il est renvoyer au serveur d'autorisation pour renouveler le jeton d'accès lorsqu'il a expiré



Usage du jeton

Le jeton est passé à travers 2 moyens :

- Les paramètres HTTP. (Les jetons apparaissent dans les traces du serveur)
- ***L'entête d'Authorization***

```
GET /profile HTTP/1.1
```

```
Host: api.example.com
```

```
Authorization: Bearer MzJmNDc3M2VjMmQzN
```

<http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/>



Validation du jeton

Lors de la réception du jeton, le serveur de ressource doit valider l'authenticité du jeton et extraire ses informations différentes techniques sont possibles

- Appel REST vers le serveur d'autorisation
- Utilisation d'un support persistant partagé (ex. JdbcStore)
- Utilisation de JWT et validation via clé privé ou clé publique



Modèles de sécurité

Introduction

SSL/TLS

Sécurité stateful

Sécurité stateless

oAuth2/OpenId Connect

JWT

Sécurisation des applications REST

Architecture micro-service



Bonnes pratiques

Stockage des mots de passe
Authentication



Authentification biométrique

Empreintes digitales, rétine, reconnaissance faciale, ...

Avantage : difficile à oublier

Problèmes:

La biométrie n'est généralement pas secrète

Non modifiable, contrairement aux mots de passe

=> Doit principalement être utilisé comme authentification à deux facteurs



2nd

facteur : OneTime Password

Mise en place :

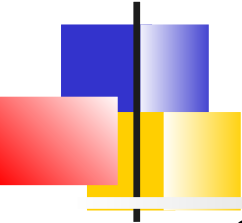
- Sur un device, téléphone ou autre, on installe une clé partagée avec le serveur
- Lors d'une connexion, tous les x temps le serveur génère un code à 6 chiffres en utilisant la clé partagée et un compteur
- Si la device saisi le bon code, l'authentification est validée



Que se passe t il si on perd son
téléphone

How to handle lock-out ?

Si la clé est compromise sur le serveur



Duo (also FIDO U2F)

Signature base challenge : La clé privée n'est pas stocké côté serveur mais une clé publique

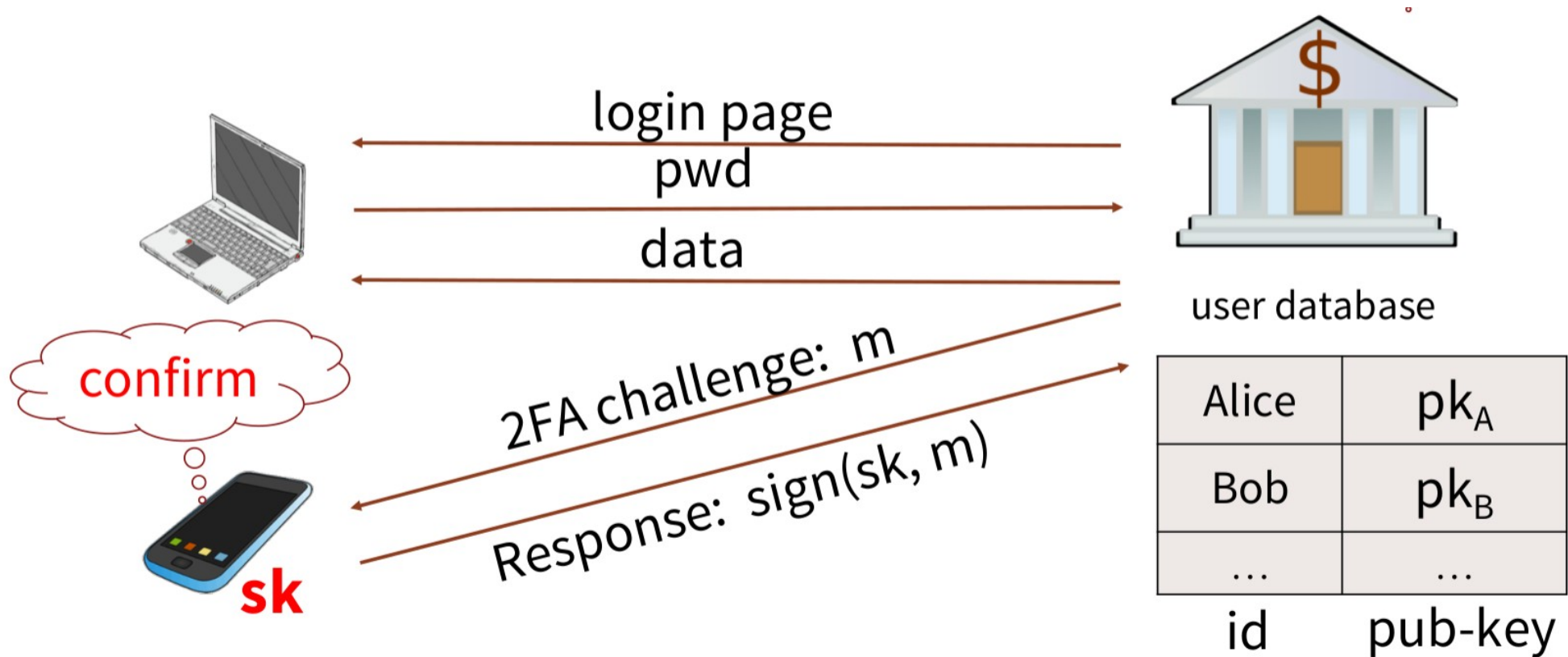
1. A la réception du mot de passe de l'utilisateur.
Le serveur envoie un challenger sur le device de l'utilisateur :

Voulez-vous me connecter ?

2. La réponse au challenge est crypté avec la clé privé du device

3. Le serveur vérifie le challenge grâce à sa clé publique.

2 facteur authentication





Bonnes pratiques

Stockage des mots de passe
Authentification
Certificats



Problèmes de délivrance de certificats

Mauvaise émission :

Problèmes de délivrance de certificats

2011 : Comodo et DigiNotar RA piratés, émettent des certificats pour Gmail, Yahoo !
Courrier

2013 : TurkTrust a délivré le certificat. pour gmail.com (découvert en épinglant)

2014 : NIC indien (autorité de certification intermédiaire approuvée par l'autorité de certification racine IndiaCCA) délivre des certificats

pour Google et Yahoo! domaines

Résultat : (1) Inde CCA a révoqué le certificat intermédiaire de NIC

(2) Chrome limite la racine India CCA à seulement sept domaines indiens

2015 : MCS (certificat d'autorité de certification intermédiaire délivré par le CNNIC) délivre des certificats pour Google

domaines

Résultat : la racine CNNIC actuelle n'est plus reconnue par Chrome

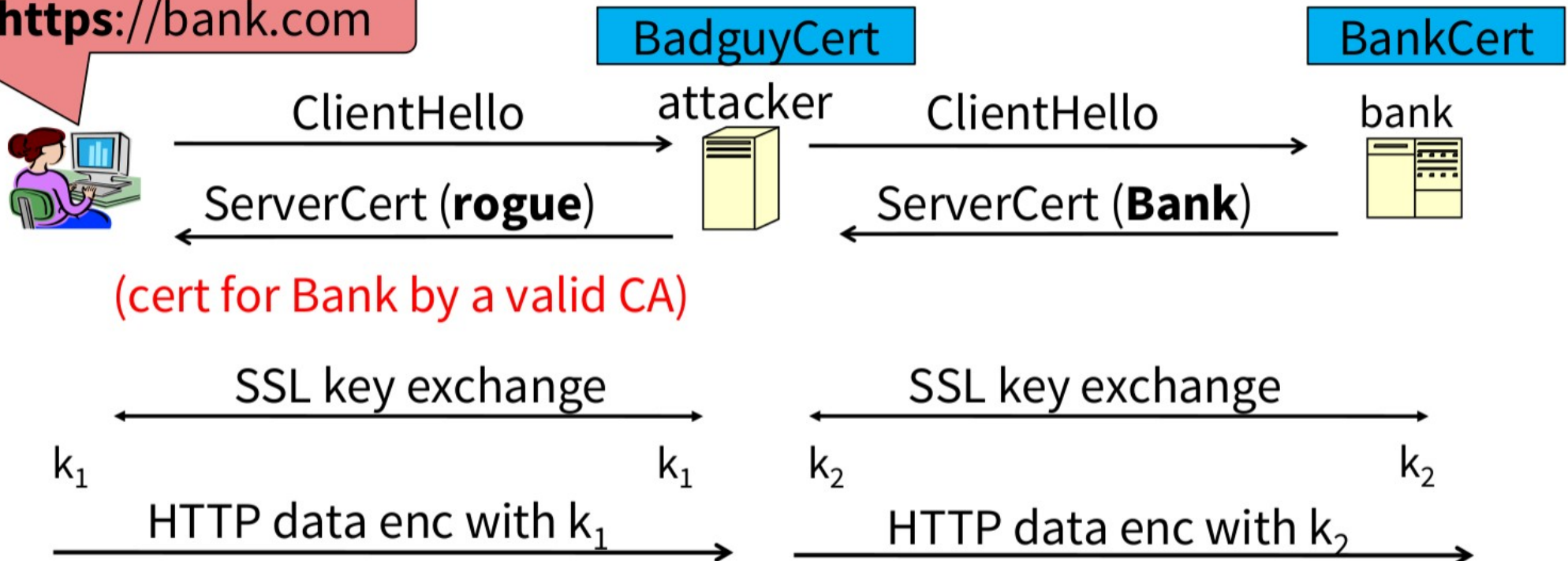
⇒


permet l'écoute sans avertissement sur la session de l'utilisateur

Man in the middle Attack

Gateway d'un pays, d'un hôtel

GET **https://bank.com**





HTTP public-key pinning

HPKP : HTTP public-key pinning

- En-tête HTTP qui permet à un site de déclarer des autorités de certification pouvant signer son certificat

Public-Key-Pins: pin-sha256="cUPcTAZWKaASuYWhhneDttWpY3oBAkE3h2+soZS7sWs=" ;

- sur les requêtes HTTPS suivantes, le navigateur rejette les certificats émis par d'autres autorités de certification

Cela fonctionne seulement si la première connexion est digne de confiance : TOFU (Trust on First Use)



Certificate Transparency (CT)

Les autorités de certification doivent publier un journal de tous les certificats qu'ils ont émis

Le navigateur n'utilisera un certificat que s'il se trouve dans le journal CT

- Mise en œuvre efficace à l'aide des arbres de hachage Merkle
- Les entreprises peuvent analyser les journaux pour rechercher une émission non valide