Sécurisation des applications réparties

Olivier Flauzac & Cyril Rabat

Licence 3 Info - Info0503 - Introduction à la programmation client/serveur

2020-2021

Cours n°6

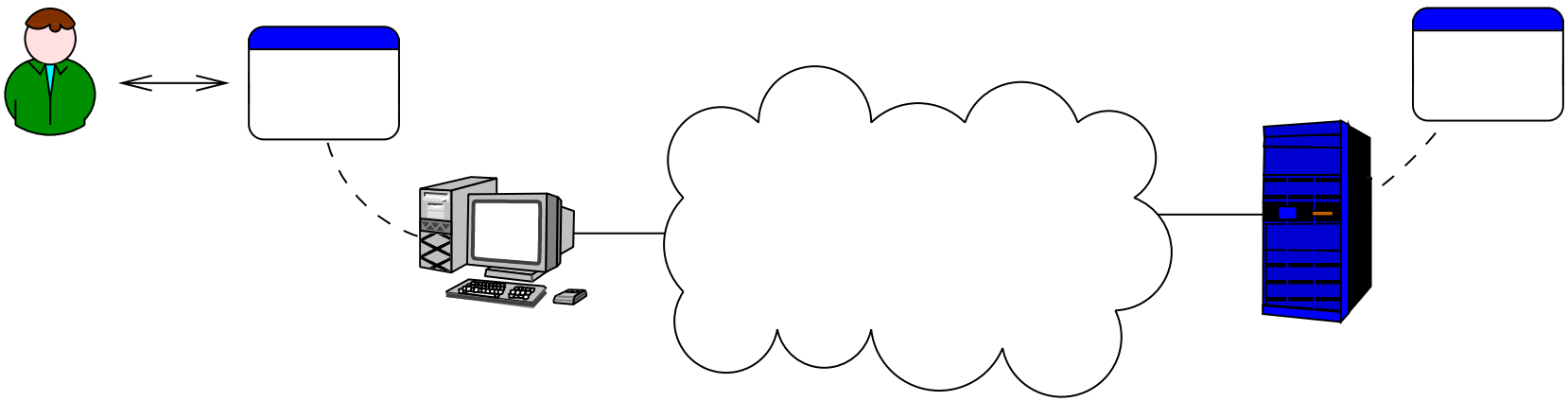
Problématiques liées à la sécurité dans les applications réparties

Présentations de solutions en Java

Version 6 octobre 2020

Table des matières

Les systèmes communicants



- Les différentes entités sont situées sur des sites indépendants
- Le système réparti n'est pas totalement sous contrôle

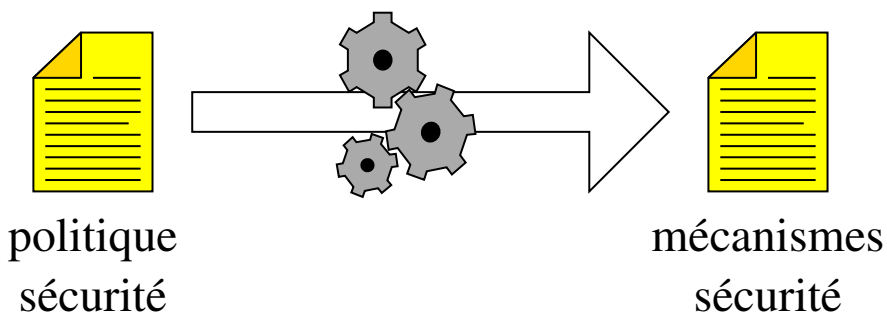
Problèmes de sécurité

À quels niveaux ?

- **Extérieur** :
  ↪ Le système réparti n'est pas dans un réseau privé !
- **Intérieur** :
  - Non contrôlable : panne de matériel, du système d'exploitation
  - Sous contrôle : application, données, ressources
- **Utilisateur** :
  - Identification
  - Autorisation d'accès dans le système

Politique et mécanismes de sécurité

- Entités : utilisateurs, services, données, machines. . .
- **Politique de sécurité** :
  - Actions possibles dans le système pour chaque entité
- **Mécanismes de sécurité** :
  - Moyens mis en place pour répondre à la politique de sécurité
    ↪ Exemples : chiffrement, authentification, etc.



Confidentialité, intégrité et authentification

Confidentialité

- Concerne toutes les parties du système : matériel, logiciels, données
- Accès uniquement aux parties autorisées
- Sécurisation des données

Intégrité

- Modifications suivant autorisation
- S'assurer de la non modification des données/messages

Authentification

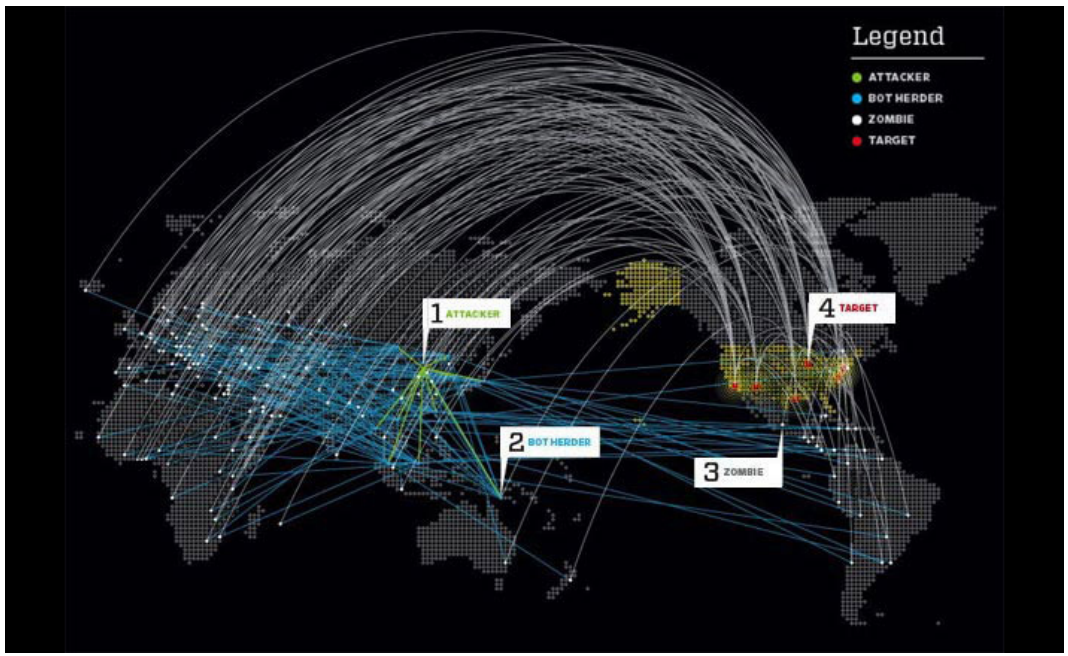
- S'assurer de l'identité de l'acteur
- Associée à des autorisations

Ce que l'on veut éviter

- **Interception** :
  - Accès à des parties non autorisées
  - Récupération de données (écoute, copie de données)
- **Interruption** :
  - Service inaccessible : déni de service (distribué)
  - Données corrompues
- **Modification** : modification non autorisées
- **Ajout** : ajout d'informations non autorisées (injection)

Déni de service (*Denial Of Service*)

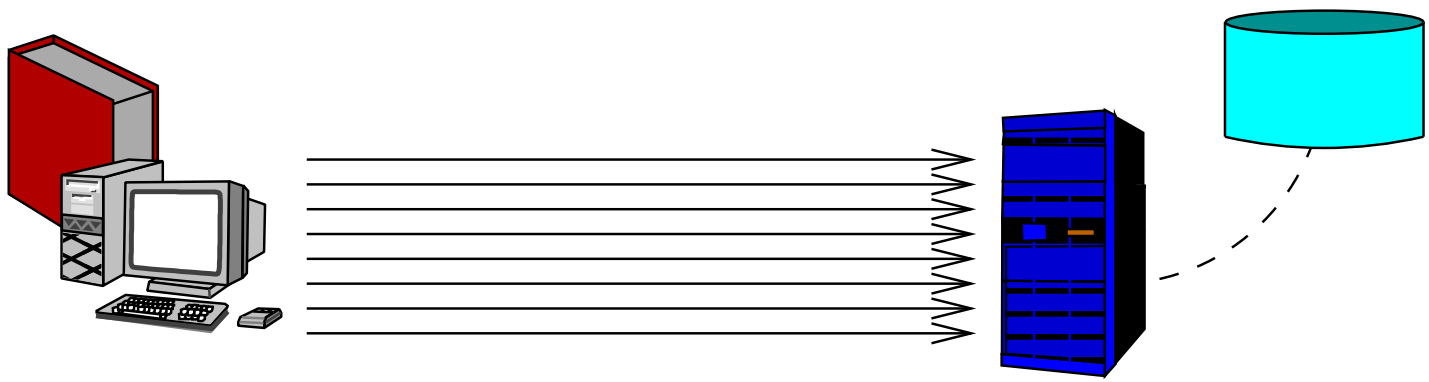
- Généralement distribué : DDoS
- Principe : saturer un serveur/service jusqu'à son interruption
- À l'aide d'un groupe de machines, envoyer des requêtes multiples



source `www.blacklotus.net`

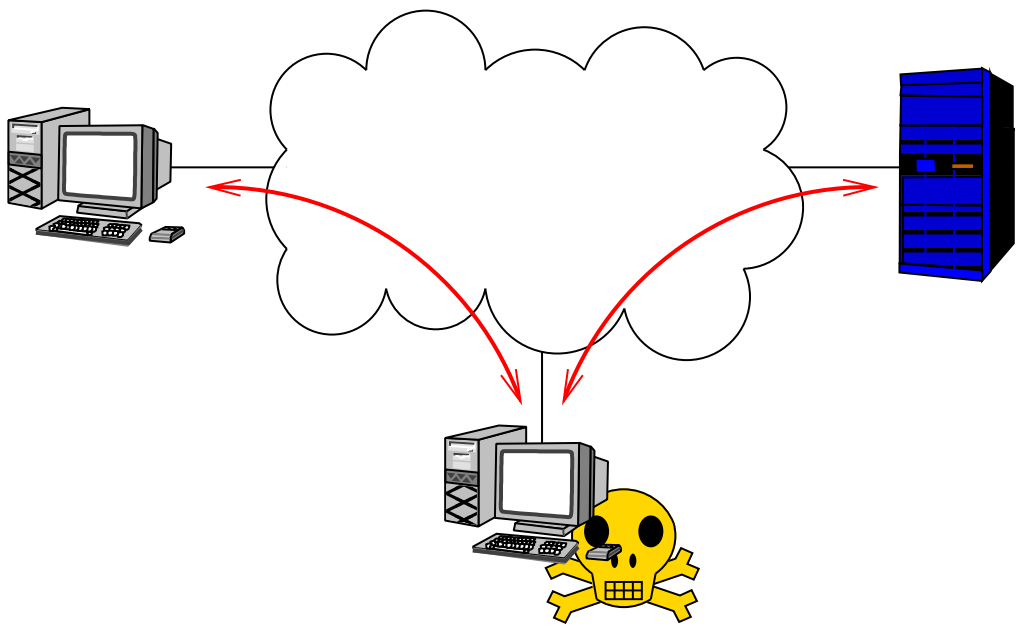
Attaque au dictionnaire

- Utilisation de la force brute pour forcer l'authentification
- À partir d'un *login* connu, tester tous les mots de passe :
  ↪ Utilisation d'un dictionnaire de mots
- Si le format du mot de passe est connu :
  ↪ Attaque encore plus simple



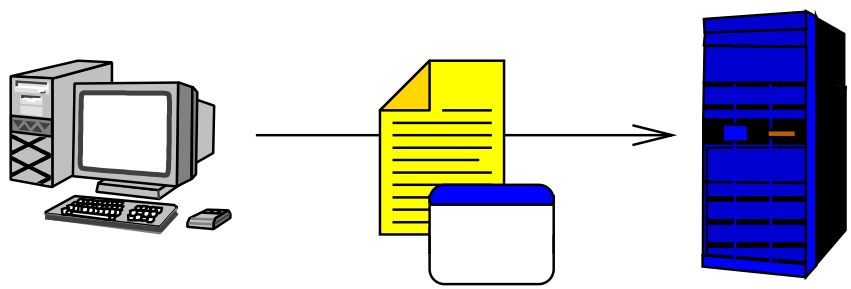
L'homme du milieu (*Man in the middle*)

- Intercepter les communications
- Technique du rejeu :
  - Données de l'émetteur captées
  - Réémission vers le destinataire

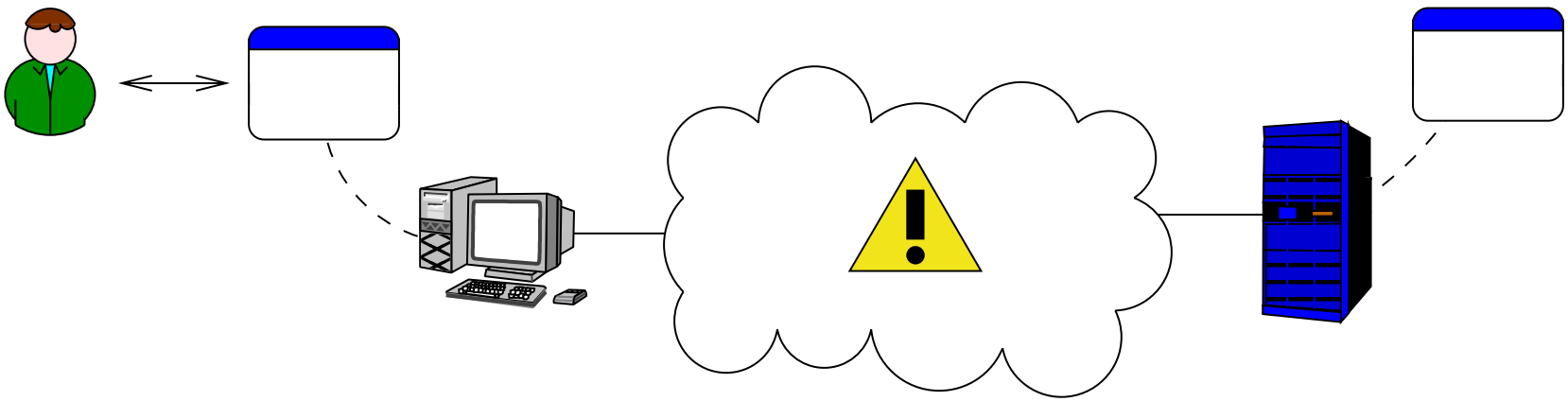


Injection de code

- Exploitation des failles du système (code, machine, logiciel, CMS...)
- En deux temps :
  1. Envoi de code sur la machine destination (exemple : *Buffer Overflow*)
  2. Obliger l'exécution du code sur la machine distante



Niveau 1 : les communications



- Les sites peuvent communiquer via un réseau publique :
  $\hookrightarrow$ Internet
- Contrôle possible uniquement à chaque bout !

Problèmes et solutions

Problèmes

- Interception des communications
- Modification des données envoyées. . .

Solutions

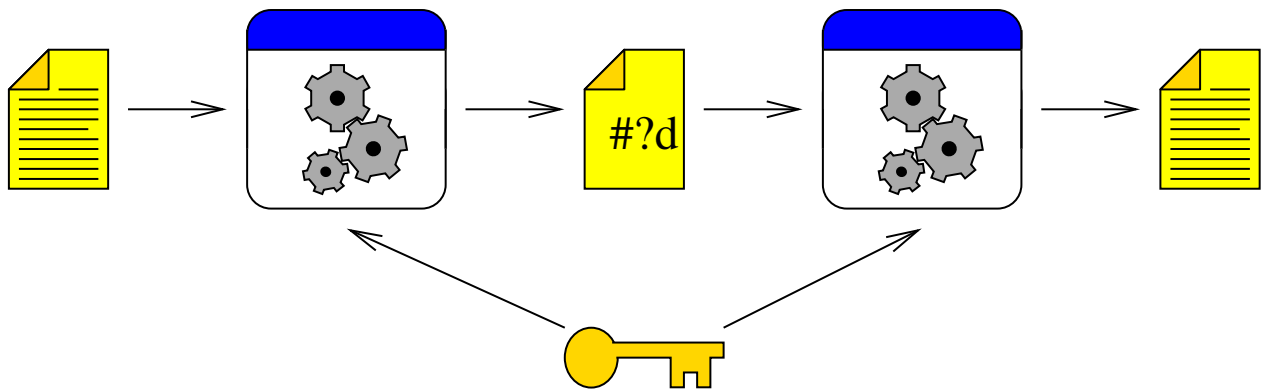
- Limiter les communications sensibles
- Louer un canal de communication sécurisé :
  ↪ Coûteux !
- Utiliser le chiffrement :
  - Exemple : VPN (*Virtual Private Network*)
  - Données rendues illisibles pour l'intercepteur . . .
    ↪ . . . pour combien de temps ?
  - N'empêche pas l'interception !

Le chiffrement

- Objectif : rendre les données illisibles
  ↪ En anglais *encryption*, francisé en *cryptage*
- Nécessite :
  - Une clé pour chiffrer
  - Une clé pour déchiffrer
  - Un algorithme de chiffrement
    ↪ Utilise la clé de chiffrement
  - Un algorithme de déchiffrement
    ↪ Utilise la clé de déchiffrement
- Deux principaux types de chiffrement :
  - Symétrique
  - Asymétrique

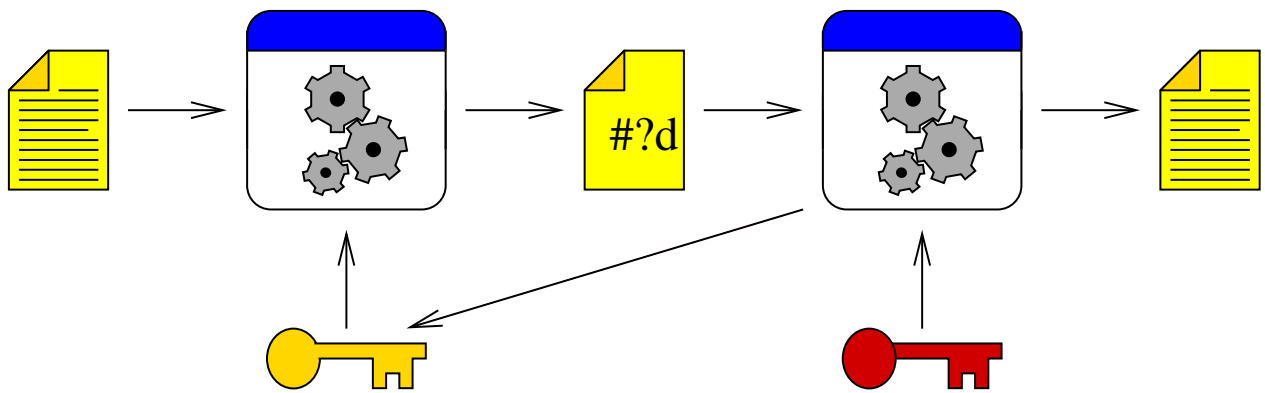
Chiffrement symétrique

- Utilisation d'une clé unique pour chiffrer et déchiffrer
- Différents algorithmes : DES, triple DES, AES...
- Avantages : rapides, petites clés
- Inconvénients : gestion des clés
  ↪ La clé doit être protégée tout en étant connue de tous



Chiffrement asymétrique

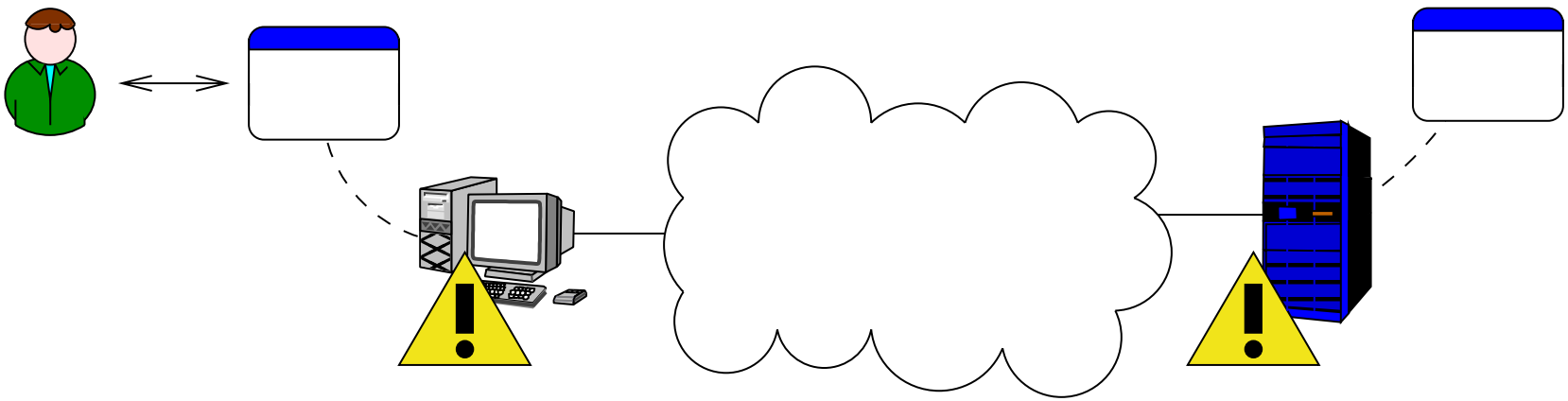
- Clés différentes pour chiffrer et déchiffrer
- Clé de chiffrement publique, clé de déchiffrement privée (ou l'inverse)
- Différents algorithmes : RSA, ElGamal, HFE...
- Avantages : clé privée locale, pas d'échange de celle-ci
- Inconvénients : longueur des clés, plus lent, gestion des clés publiques



Intégrité des données

- Objectif : s'assurer que les données n'ont pas été modifiées
- Utilisation d'un algorithme de hachage :
  ↪ MD5, SHA
- Génération d'une empreinte unique du message/des données
- Caractéristiques d'un algorithme de hachage :
  - Empreintes produites suffisamment petites
  - Éviter autant que possible les collisions :
    ↪ Mêmes empreintes pour des entrées différentes
  - Les modifications des données doivent produire des empreintes différentes
- Généralement utilisé conjointement au chiffrement

Niveau 2 : le matériel



- Les applications s'exécutent sur des machines :
  ↪ Pannes des machines !
  ↪ Accès aux machines
- Authentification : à qui je parle vraiment ?

Incidences des pannes

- Qu'est-ce que ça impacte ?
  - Les applications / services
  - Les données / messages
- La panne d'un site entraine :
  - Service(s) indisponible, voire de tout le système
  - Données inaccessibles voire perdues
- Question sur les données :
  - Où se trouvent les données ?
  - Comment sont-elles stockées ?

Tolérance aux pannes

- Services :
  - Réplication (attention aux données)
  - Migration à la volée vers un autre site :
    ↪ Avantage de la virtualisation
- Données :
  - Pas de donnée sensible en mémoire volatile
  - Stockage sur un tiers indépendant (SGBD, NAS)
  - Réplication (matérielle, logique)

Accès extérieur

- Côté client :
  - Données stockées accessibles hors-connexion
  - Accès depuis d'autres applications
    ↪ Mots de passe sauvegardés, cookies. . .
- Côté serveur :
  - Moins d'applications concurrentes
  - Accès possible physiquement :
    - Récupération des données stockées sur le disque dur
    - Récupération des données dans la base de données
      ↪ Protection des données privées !

Authentification

- Communication avec un site distant :
  $\hookrightarrow$ Comment être sûr de son identité ?
- Plusieurs solutions :
  - Mot de passe
  - Challenge
  - Signature...

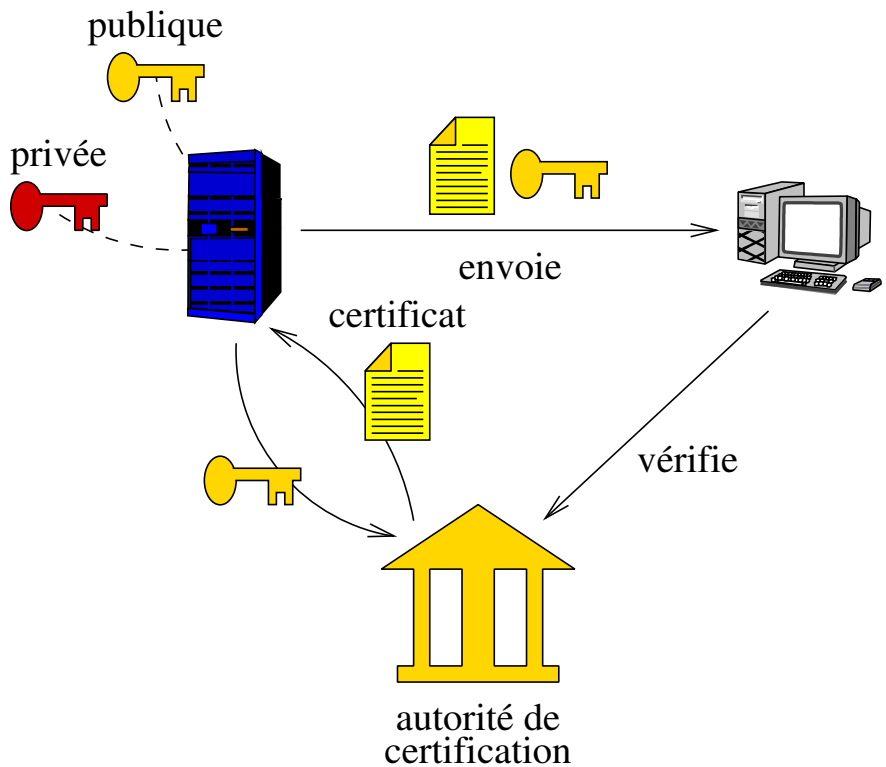
Authentification

- Communication avec un site distant :
  $\hookrightarrow$ Comment être sûr de son identité ?
- Plusieurs solutions :
  - Mot de passe
  - Challenge
  - Signature...

On est jamais sûr de rien à 100% !

Certificats numériques

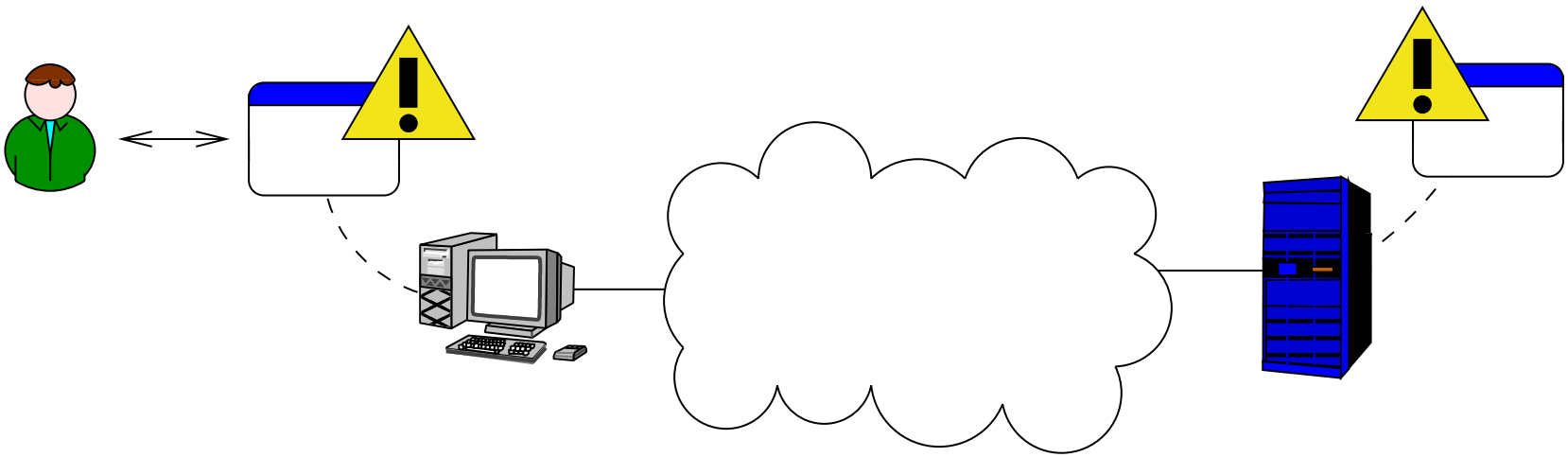
- Récupération d'un certificat auprès de l'autorité
- Le serveur fournit au client le certificat + la clé publique
- Le client peut vérifier le certificat auprès de l'autorité



Signature des messages

- Vérifier l'intégrité d'un message n'est pas suffisant
- Ajouter une information pour s'assurer du bon émetteur :
  ↪ Sa signature
- Problèmes :
  ↪ La signature ne doit pas être imitable
  ↪ Le destinataire du message doit pouvoir vérifier
- Utilisation du principe de clé privée/publique :
  1. L'émetteur chiffre le message avec sa clé privée
  2. Le destinataire reçoit le message et déchiffre avec la clé publique
  3. Vérification du résultat

Niveau 3 : les applications

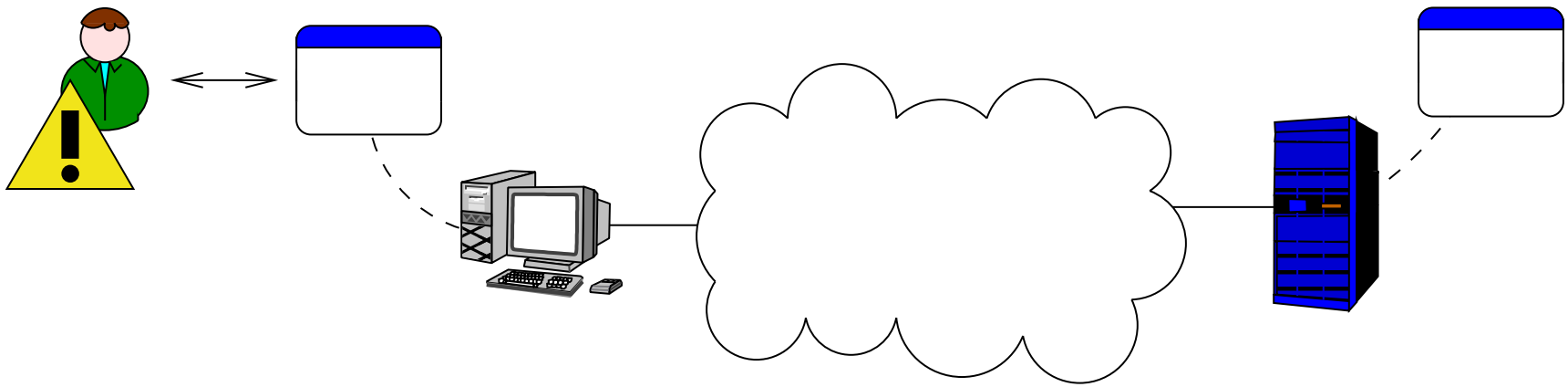


- Applications exécutées sur les sites
- Ressources locales accessibles

Problèmes de sécurité

- L'application s'exécute sur la machine :
  ↪ Accède potentiellement aux ressources systèmes
- Plusieurs solutions :
  - Le code/script est connu :
    ↪ Pirate : manipulation pour exécution arbitraire
  - Le code/script n'est pas connu (code mobile) :
    ↪ Serveur/client : exécution contrôlée
- Notion de bac à sable :
  - Restriction des droits pour l'exécution
  - Indiquer les opérations possibles

Niveau 4 : les utilisateurs



- Identification de l'utilisateur
- Une fois identifié, l'utilisateur "est dans le système"

Problèmes de sécurité

- Authentification (comme vu précédemment) :
  ↪ Une fois authentifié, l'utilisateur devient partie intégrante du système
- Utilisateur conscient (pirate) :
  - Récupération du mot de passe
  - Porte dérobée (cheval de Troie)
- Utilisateur inconscient :
  - Mot de passe trop simple ("toto")
  - Sécurisation du poste de travail :
    ↪ Problème du nomadisme

Nombreuses problématiques

Quelques solutions en *Java*

- Niveau 1 : les communications
  - ↪ Intégrité des données : SHA
  - ↪ Chiffrement : AES
  - ↪ Signature : DSA
- Niveau 2 : le matériel
  - ↪ Accès extérieur : stockage des données
  - ↪ Authentification : mot de passe, challenge
- Niveau 3 : les applications
  - ↪ Contrôle de l'exécution : gestionnaire de sécurité de *Java*
- Niveau 4 : les utilisateurs
  - ↪ Accès : définition des droits

Protection de mot de passe

- Pour se loguer, utilisation d'une requête POST/GET HTTP
- Si la requête est interceptée :
  - *Login* et mot de passe "en clair"
  - Possible de se faire passer pour l'utilisateur !

Protection de mot de passe

- Pour se loguer, utilisation d'une requête POST/GET HTTP
- Si la requête est interceptée :
  - *Login* et mot de passe "en clair"
  - Possible de se faire passer pour l'utilisateur !

Solution «naïve»

- Chiffrement du mot de passe avant l'envoi dans le réseau :
  $\hookrightarrow$ Utilisation de MD5/SHA
- Le pirate ne peut plus récupérer le mot de passe (en clair)

Protection de mot de passe

- Pour se loguer, utilisation d'une requête POST/GET HTTP
- Si la requête est interceptée :
  - *Login* et mot de passe "en clair"
  - Possible de se faire passer pour l'utilisateur !

Solution «naïve»

- Chiffrement du mot de passe avant l'envoi dans le réseau :
  ↪ Utilisation de MD5/SHA
- Le pirate ne peut plus récupérer le mot de passe (en clair)

Problème

- Le pirate n'a pas besoin du mot de passe pour se loguer !
  ↪ Le rejeu de l'empreinte suffit
- Possible de récupérer le mot de passe à l'aide d'un inverseur
  ↪ Si le mot de passe est trop simple

Des solutions

Avec HTTPs (classes `SSLSocket`, `SSLSocketFactory`...)

- Communications sécurisées :
  ↪ Avec le chiffrement, pas possible de récupérer le mot de passe
- Dépend de l'hébergement :
  ↪ HTTPs peut ne pas être disponible

Avec MD5/SHA, toujours

- Problème : l'empreinte correspond au mot de passe
- En combinant plusieurs informations non statiques :
  ↪ Plus difficile à mettre en œuvre
- Utilisation d'un captcha dont la réponse est couplée au mot de passe
  ↪ Empêche l'utilisation de robots
  ↪ Mais le captcha peut être aussi intercepté
- D'autres fonctions de hachage plus performantes peuvent être utilisées

Hachage avec MD5/SHA

- Utilisation de la classe `MessageDigest`
  ↪ Correspond à une empreinte de message
- Instanciation à l'aide de la méthode `getInstance` :
  ↪ On spécifie l'algorithme utilisé (MD5, SHA-1, SHA-256)
- Deux solutions :
  - Messages longs (fichier) :
    ↪ Utilisation de la méthode `update`
    ↪ puis `digest`
  - Messages courts :
    ↪ Directement méthode `digest`
- Le résultat est un tableau d'octets (`byte[]`)

Exemple

```
// Mot de passe
String motDePasse = "admin";

// Préparation de l'empreinte
MessageDigest empreinte = MessageDigest.getInstance("SHA-256");

// Calcul de l'empreinte
byte[] bytes = empreinte.digest(motDePasse.getBytes());
```

Chiffrement avec AES

- Utilisation de la classe `Cipher` qui permet de chiffrer :
  ↪ Indépendante de l'algorithme utilisé
- Nécessité de spécifier une clé :
  ↪ Utilisation de `SecretKeySpec`
- Pour AES, on utilise une clé de 128 bits (16 caractères)
  ↪ Attention à l'encodage !

Exemples d'utilisation

Chiffrement AES

```
String message = "Le message que je veux chiffrer";
SecretKeySpec specification =
  new SecretKeySpec(motDePasse.getBytes(), "AES");
Cipher chiffreur = Cipher.getInstance("AES");
chiffreur.init(Cipher.ENCRYPT_MODE, specification);
byte[] messageChiffre = chiffreur.doFinal(message.getBytes());
```

Déchiffrement AES

```
SecretKeySpec specification =
  new SecretKeySpec(motDePasse.getBytes(), "AES");
Cipher dechiffreur = Cipher.getInstance("AES");
dechiffreur.init(Cipher.DECRYPT_MODE, specification);
byte[] bytes = dechiffreur.doFinal(messageChiffre);
String messageClair = new String(bytes);
```

Signature d'un document avec DSA

- Génération d'une paire de clés privée/publique :
  ↪ Utilisation de l'algorithme DSA (par exemple)
- Généralement, cette étape est inutile :
  ↪ Les hôtes possèdent déjà des clés
  ↪ Possible de les importer dans une application *Java*
- Utilisation de la classe `Signature` :
  ↪ Génération de la signature : utilisation de la clé privée
  ↪ Vérification de la signature : utilisation de la clé publique

Création des clés privée/publique

Extrait de code

```
// Générateur de clés
KeyPairGenerator generateurCles =
  KeyPairGenerator.getInstance("DSA");
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
generateurCles.initialize(1024, random);

// Génération de la paire de clés
KeyPair paireCles = generateurCles.generateKeyPair();
PrivateKey clePrivee = paireCles.getPrivate();
PrivateKey clePublique = paireCles.getPublic();
```

Génération de la signature

Extrait de code

```
// Initialisation de la signature
Signature signature = Signature.getInstance("SHA1withDSA");
signature.initSign(clePrivee);

// Mise a jour de la signature par rapport au contenu du fichier
String nomFichierASigner = "monFichier.bin";
BufferedInputStream fichier =
  new BufferedInputStream(new FileInputStream(monFichierASigner));
byte[] tampon = new byte[1024];
int n;
while (fichier.available() != 0) {
  n = fichier.read(tampon);
  signature.update(tampon, 0, n);
}
fichier.close();

// Sauvegarde de la signature du fichier
String fichierSignature = "signature.bin";
FileOutputStream fichierSignature = new FileOutputStream(fichierSignature);
fichierSignature.write(signature.sign());
fichierSignature.close();
```

Vérification de la signature

Extrait de code

```
// Création de la signature
Signature signature = Signature.getInstance("SHA1withDSA");

// Initialisation de la signature
signature.initVerify(clePublique);

// Mise a jour de la signature par rapport au contenu du fichier
String nomFichierASigner = "monFichier.bin";
BufferedInputStream fichier =
  new BufferedInputStream(new FileInputStream(monFichierASigner));
byte[] tampon = new byte[1024];
int n;
while (fichier.available() != 0) {
  n = fichier.read(tampon);
  signature.update(tampon, 0, n);
}
fichier.close();

// Vérification de la signature
if(signature.verify(signatureFournie))
  System.out.println("Fichier OK");
else
  System.out.println("Fichier invalide");
```

Conclusion

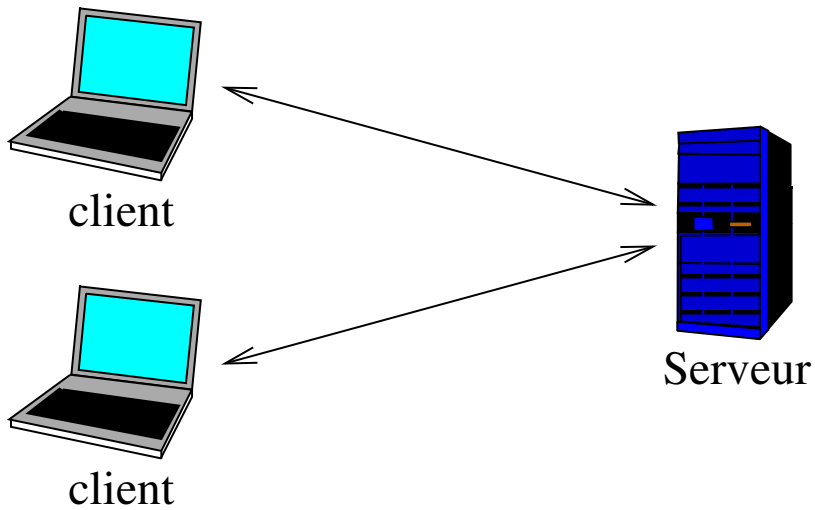
- Un système de sécurité doit rester simple : complexité $\neq$ sécurité
- Se reposer sur des solutions existantes :
  $\hookrightarrow$ Empêche les erreurs d'implémentation
  $\hookrightarrow$ Nécessite de connaître cette solutions (bugs, mises-à-jour, etc.)
- Sécurité à 100% : n'existe pas !
  $\hookrightarrow$ Ça n'empêche pas de prendre ses précautions...
- La sécurité : pas uniquement au niveau de l'application !
  $\hookrightarrow$ Communications, matériel, logiciels, etc...
- Ne jamais oublier les utilisateurs finaux :
  - Pas de contrôle sur leur poste de travail
  - Pas/peu de contrôle sur leurs actions

«Si vous pensez que la technique peut résoudre tous vos problèmes de sécurité, c'est que vous n'avez rien compris à la technique, ni à vos problèmes.»

Bruce Schneier

Présentation d'une application classique

Architecture



- Communications basées sur le modèle client/serveur
  ↪ Dialogue complet quelconque (plusieurs échanges)
- On souhaite :
  - Authentifier les entités (applications et utilisateurs)
  - Sécuriser les communications

Authentification des utilisateurs

- Que doit-on envoyer ?
  ↪ Exemple classique : login/mot de passe
- Comment vérifier du côté serveur ?
  ↪ Base de données, fichier (à plat, JSON, etc.)
- Quand réaliser l'authentification ?
  ↪ Dépend du/des protocoles utilisés

Développement avec des sockets

Avec TCP

- Établissement d'une connexion
- Authentification avant le début du dialogue

Avec UDP

- Pas de connexion : messages indépendants !
- Vérification à chaque message

Avec HTTP

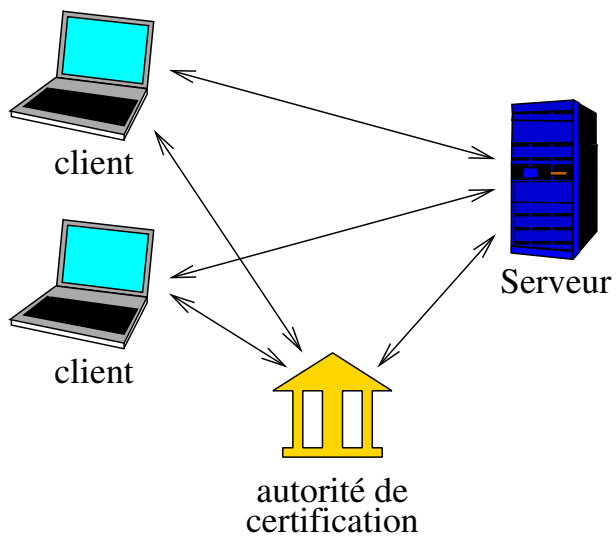
- Même problème qu'avec UDP : protocole sans état
  $\hookrightarrow$ Pourtant basé sur TCP !

Solution pour UDP et HTTP

Utilisation d'une session

Authentification des applications

- Indépendante des utilisateurs
  ↪ Doit être réalisée avant !
- Authentification mutuelle :
  ↪ Le serveur doit authentifier les clients
  ↪ Les clients doivent authentifier le serveur
- Solution :
  ↪ Utilisation de certificats



Mise en place des certificats

- Données :
  - Informations sur l'entité
  - Données diverses :
    ↪ Autorité de certification, date de validité
- Échange des certificats :
  - Avant tout autre échange
  - Vérification des certificats
- Nécessite une autorité de certification :
  ↪ Le tiers de confiance !

Autorité de certification

- On souhaite développer notre propre autorité de certification
  ↪ Le but est avant tout pédagogique !
- Quelle technologie utiliser ?
  - Sockets (mode connecté ou mode non connecté)
  - HTTP...
- Que placer dans le certificat ?
- Comment représenter/stocker le certificat ?
  ↪ INFO0503 : utilisation de JSON

Remarque

Des normes régissent les certificats (et leur contenu). Exemple : X.509

Création de certificats

- Nécessite des clés pour le chiffrement asymétrique
  ↪ Clients, serveur et autorité de certification
- Création du certificat pour l'autorité :
  ↪ Nom, adresse IP, port, clé publique
  ↪ Signature du certificat !
- Création du certificat pour les autres entités :
  - Récupération du certificat de l'autorité (faiblesse !)
  - Envoi des données à l'autorité (nom, adresse IP, port, clé publique)
    ↪ En clair ? Chiffrées ?
  - Challenge par l'autorité (pour s'assurer de l'identité de l'entité)
  - Envoi du certificat par l'autorité :
    ↪ Données du client
    ↪ Informations sur l'autorité de certification
    ↪ Signature du certificat

Un mot sur les certificats au format JSON (1/2)

- JSON : données au format texte
- Signature = flot binaire
- Nécessite d'utiliser des solutions d'encodage :
  ↪ Exemple : base64

Un mot sur les certificats au format JSON (2/2)

```
// Encodage en Base64
String chaine = Base64.getEncoder().encodeToString("Texte␣à␣encoder".
    getBytes("utf-8"));
System.out.println(chaine);

// Décodage
byte[] octets = Base64.getDecoder().decode(chaine);
System.out.println(new String(octects, "utf-8"));
```

Sécurisation des communications

- Chiffrement des communications :
  ↪ Utilisation du chiffrement symétrique
- Nécessite d'avoir au préalable un échange de la clé :
  ↪ Utilisation du chiffrement asymétrique

Remarque

- Sans sécurisation des communications :
  ↪ Échanges de données au format JSON ; format texte
- Avec sécurisation des communications :
  ↪ Messages au format binaire

Données nécessaires pour chaque entité

- Fichier de configuration pour chaque entité
  ↪ Configuration de l'application
  ↪ Clés privées/publiques (noms de fichier)
  ↪ Certificat de l'autorité (nom de fichier)
  ↪ Certificat de l'entité (nom de fichier)
- Clés privée/publique
- Certificats (un seul pour l'autorité)
- Par défaut : rien !
  ↪ Génération/vérification à chaque exécution