# Protocoles Internet
# TP 5 : HTTP sécurisé

Juliusz Chroboczek

29 octobre 2023

**Exercice 1** (Certificats X.509)**.** À l'aide de votre navigateur *web* favori, connectez-vous au site

`https://galene.org`

1. La communication est-elle chiffrée ? Cliquez sur le cadenas, ou, si vous utilisez une version récente de *Chrome*, sur l'icône incompréhensible à gauche de l'adresse.
2. Le serveur est-il authentifié au client ? Quel est le *Subject* déclaré dans le certificat ? Les *Subject Alternate Name* ?
3. Qui a authentifié le certificat ? Avec quel algorithme de signature ?
4. Obtenez les mêmes informations à l'aide de la commande `gnutls-cli`. Quel algorithme de chiffrage symétrique est utilisé ?
5. Que constatez-vous d'intéressant à propos du *Subject Alternate Name* du site `https://www.irif.fr` ?

**Exercice 2** (Un serveur HTTPS)**.**

1. Créez un certificat X.509 à l'aide de la commande suivante :

   ```
   openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365
               -out cert.pem
   ```

   Quels fichiers ont été crées ? Que contiennent-ils ? (Indication : utilisez la commande `file`.)
2. Écrivez un petit serveur *web* en Go qui présente une seule page (vous pouvez par exemple reprendre le serveur du TP 1). Transformez votre serveur en serveur HTTPS en remplaçant l'appel à la fonction `ListenAndServe` par un appel à `ListenAndServeTLS`.
3. Testez votre serveur à l'aide d'un navigateur *web*. Que constatez-vous ? Comme à l'exercice 1, examinez le certificat à l'aide d'un navigateur *web* puis à l'aide de `gnutls-cli`.
4. Pourquoi ai-je écrit le paquet `github.com/jech/cert` ? (Il a deux fonctionnalités intéressantes.)

**Exercice 3** (Authentification par mot de passe)**.**

1. Modifiez le gestionnaire du serveur de l'exercice précédent pour qu'il vérifie la présence d'un entête `Authorization` (testez la valeur de `r.Header.Get("Authorization")`), puis, s'il est absent ou vide, qu'il retourne une réponse de code 401. Dans le cas contraire, il affichera dans le terminal la valeur de l'entête :

```
auth := r.Header.Get("Authorization")
if auth == "" {
    w.Header().Set("www-authenticate", "basic realm=\"tp1\"")
    http.Error(w, "Haha!", http.StatusUnauthorized)
    return
}
log.Println("Authorization:", auth)
```

Testez votre serveur. Quel est le comportement du client ?

2. Modifiez votre serveur pour qu'il vérifie que le nom d'utilisateur envoyé par le client est `einstein` et que le mot de passe est `elsa`, et retourne une réponse de type 200 si c'est le cas, de type 401 dans le cas contraire. (Vous pouvez vous servir de la méthode `BasicAuth` de la structure `http.Request`.)
3. Comment s'appelle le mécanisme d'authentification implémenté dans cet exercice ? Authentifie-t-il le client ou le serveur ? À quelles attaques est-il vulnérable ? Pourquoi l'a-t-on implémenté au-dessus de HTTPS et pas HTTP ?

**Exercice 4** (Authentification tierce sans état)**.**

1. Écrivez un programme qui effectue une requête `POST` à l'URL

   `https://jch.irif.fr:8443/get-token`

   dont le corps contient un objet `JSON` ayant la forme suivante :

   ```
   {
       "username": "xxx",
       "password": "yyy"
   }
   ```

   Le nom d'utilisateur est arbitraire (mettez votre nom), le mot de passe est « `Rosebud` ».
2. Modifiez votre programme pour qu'il fasse ensuite une requête `GET` à l'URL

   `https://jch.irif.fr:8443/top-secret`

   avec un entête `Authorization` de type `Bearer` transmettant le *token* obtenu ci-dessus. (La syntaxe de l'entête est décrite dans la RFC 6750 paragraphe 2.1.)
3. On a donc dû faire deux requêtes pour finalement accéder aux données, avec deux mécanismes d'authentification différents. À quoi sert cette complexité ?
4. Le *token* consiste de trois parties séparées par des points « `.` ». À l'aide de la commande « `base64 -di` », examinez chacune des trois parties. Le client a-t-il besoin de connaître la structure du *token* ? A-t-il besoin de savoir si l'implémentation des *tokens* est avec ou sans état ?
5. Les deux interactions sont protégées par TLS. Pourquoi est-ce nécessaire ?
6. Qu'est-ce qui empêche un client malicieux de générer un faux *token* ?
7. Que se passe-t-il si vous rejouez le *token* ? Que se passe-t-il si vous rejouez le *token* avec 40 secondes de délai ?