Joséphine VETU

SIO2

Cybersécurité

Activité 13 :
Défauts de configurations de chiffrement

Table des matières

Défi n°1 : scan des algorithmes de chiffrement d'une application web 2
Q1. Définir le protocole TLS. Expliquer la différence entre TLSv1.2 et TLSv1.3 en terme de performance et de sécurité. 2
Q2. Qu'est ce qu'un certificat de sécurité SSL ?. Quel est son rôle dans la sécurisation d'une application web ? 2
Q3. Qu'est ce qu'une autorité de certification ? Quelle différence entre une autorité privée et une autorité publique. Retrouver la liste des autorités publiques sur votre navigateur. 3
Q4. Tester la sécurité SSL/TLS des applications suivantes en utilisant les outils de scan : 3
Q5. A l'aide de vos recherches sur internet, expliquer ce qu'est la vulnérabilité Heartbleed. Quelles peuvent être les conséquences de cette vulnérabilité ? 5
Défi n°2: force brute d'un mot de passe haché en MD5 7
Q1. Rappeler la différence entre chiffrement symétrique et chiffrement asymétrique et proposer un schéma pour chacune de ces deux méthodes. 7
Q2. Recenser et décrire les principaux algorithmes de chiffrement symétrique et asymétrique. 7
Q3. Qu'appelle t-on une fonction de hachage, quel peut-être son rôle lors du développement d'une application. Lister les principales fonctions de hachage et leurs principales caractéristiques. 8
Q4. Expliquer ce que fait le code java suivant et quel est le problème de sécurité posé par ce code ? 9
Q5. Dans le code Symfony suivant, expliquer le rôle de 'auto'. Conclure sur l'intérêt d'utiliser un framework lors du développement d'une application. 10
Q6. Un attaquant récupère le haché suivant dans une base de données. 4138dd8a9c4ac256fcf4a42b633f9f88 11

Défi n°1 : scan des algorithmes de chiffrement d'une application web

Q1. Définir le protocole TLS. Expliquer la différence entre TLSv1.2 et TLSv1.3 en terme de performance et de sécurité.

Le **protocole TLS** (Transport Layer Security) est un protocole de sécurisation des communications sur Internet. Il est utilisé pour assurer la confidentialité, l'intégrité et l'authenticité des données échangées entre un client et un serveur, notamment via HTTPS.

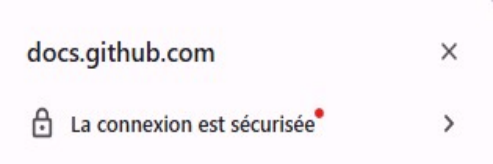
Différences entre **TLS 1.2 et TLS 1.3** en termes de performance et de sécurité :

1. **Sécurité** :
   - TLS 1.2 repose sur des suites de chiffrement qui incluent souvent des algorithmes vulnérables comme RSA et SHA-1.
   - TLS 1.3 supprime les anciens algorithmes cryptographiques obsolètes et impose l'utilisation de chiffres plus sécurisés comme ChaCha20-Poly1305 et AES-GCM. Il utilise des méthodes de chiffrement plus modernes et plus sûres.
2. **Performance** :
   - TLS 1.3 est plus rapide car il réduit le temps nécessaire pour établir une connexion sécurisée.
   - Il nécessite moins d'échanges entre le client et le serveur avant de commencer l'échange de données:
     - TLS 1.2 nécessite **deux allers-retours** (handshake) entre le client et le serveur avant de commencer l'échange de données.
     - TLS 1.3 n'a besoin que d'**un seul aller-retour**, ce qui réduit le temps d'établissement de connexion.

TLS 1.3 est plus sécurisé et plus performant que TLS 1.2. Il est recommandé d'utiliser TLS 1.3 si possible.

Q2. Qu'est ce qu'un certificat de sécurité SSL ?. Quel est son rôle dans la sécurisation d'une application web ?

Un **certificat SSL** (Secure Sockets Layer) est un fichier numérique émis par une autorité de certification (CA - Certificate Authority) qui permet d'authentifier l'identité d'un site web et de chiffrer les échanges entre le serveur et l'utilisateur.



Rôle du certificat SSL dans la sécurisation d'une application web :

- Authentification : Il **prouve l'identité du site web** aux visiteurs.

- Chiffrement : Il protège les échanges de données entre le serveur et le client contre les interceptions (ex. attaques "Man-in-the-Middle").
- Intégrité : Il empêche la modification des données échangées.
- Confiance : Il permet au navigateur d'afficher un **cadenas sécurisé**, ce qui rassure les utilisateurs.

Q3. Qu'est ce qu'une autorité de certification ? Quelle différence entre une autorité privée et une autorité publique. Retrouver la liste des autorités publiques sur votre navigateur.

Une **autorité de certification** (CA) est une organisation qui délivre des certificats numériques en garantissant l'identité des entités (sites web, entreprises, serveurs, utilisateurs).

Différences entre une autorité privée et une autorité publique :

- **Autorité privée** : utilisée au sein d'une organisation pour sécuriser des communications internes (exemple : un réseau d'entreprise, VPN, Intranet).
- **Autorité publique** : reconnue mondialement, elle délivre des certificats aux sites web et services accessibles publiquement (exemple : Let's Encrypt, DigiCert, GlobalSign).

Liste des autorités publiques dans votre navigateur et comment les retrouver:

- Google Chrome / Edge :
  Paramètres > Sécurité et confidentialité > Gérer les certificats
- Mozilla Firefox :
  Options > Vie privée et sécurité > Afficher les certificats
- Windows :
  dans l'invite de commande : *certmgr.msc*

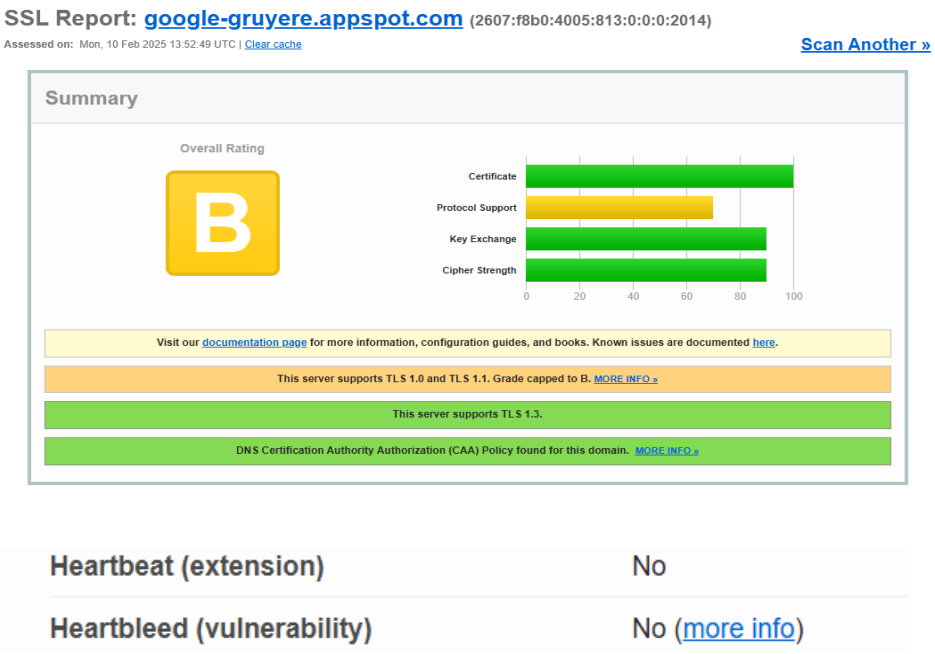
Ces autorités sont intégrées aux navigateurs et systèmes d'exploitation pour **garantir la légitimité des certificats SSL** des sites web.

Q4. Tester la sécurité SSL/TLS des applications suivantes en utilisant les outils de scan :

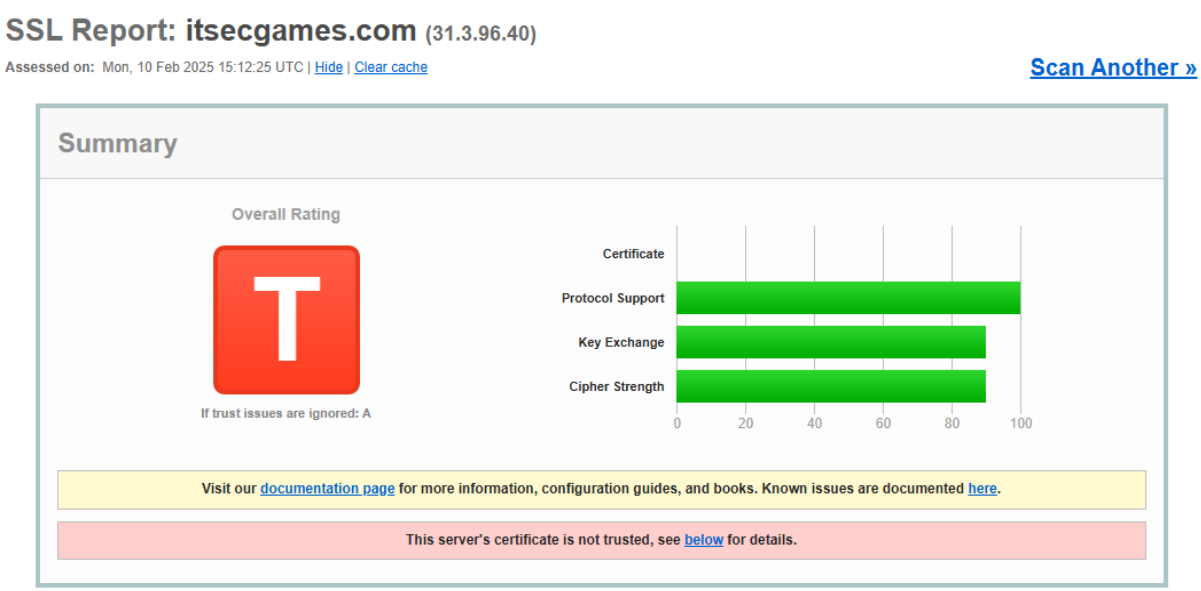
J'ai utilisé le site web https://www.ssllabs.com/ pour tester ces applications:

- Google Gruyere https://google-gruyere.appspot.com/start

Activité 13 : Défauts de configurations de chiffrement



– BWAP http://itsecgames.com



Extended Validation	No
Certificate Transparency	No
OCSP Must Staple	No
Revocation information	None
DNS CAA	No (more info)
Trusted	No **NOT TRUSTED** (Why?) Mozilla Apple Android Java Windows

Q5. A l'aide de vos recherches sur internet, expliquer ce qu'est la vulnérabilité Heartbleed. Quelles peuvent être les conséquences de cette vulnérabilité ?

La vulnérabilité **Heartbleed** est une faille de sécurité découverte en avril 2014 dans la bibliothèque cryptographique OpenSSL, largement utilisée pour sécuriser les communications sur Internet via les protocoles SSL/TLS.

Cette faille permettait à un attaquant d'exploiter une erreur dans l'implémentation de l'extension "Heartbeat" de TLS, lui offrant la possibilité de **lire la mémoire du serveur ou du client vulnérable**. Ainsi, des informations sensibles telles que des clés privées, des noms d'utilisateur, des mots de passe ou d'autres données confidentielles pouvaient être exposées.

Conséquences potentielles de la vulnérabilité Heartbleed :

- **Compromission des clés privées** : Les attaquants pouvaient accéder aux clés privées des certificats SSL/TLS, compromettant ainsi la confidentialité des communications chiffrées et permettant des attaques de type *man-in-the-middle*.
- **Vol de données sensibles** : Des informations telles que des mots de passe, des numéros de cartes de crédit ou des messages instantanés pouvaient être dérobées sans laisser de trace, mettant en danger la confidentialité des utilisateurs.
- **Usurpation d'identité** : Avec les informations obtenues, les attaquants pouvaient se faire passer pour des utilisateurs légitimes.
- **Atteinte à la réputation des entreprises** : Les organisations touchées par Heartbleed ont dû faire face à une perte de confiance de la part de leurs clients et partenaires, ainsi qu'à des coûts liés à la correction de la faille et à la gestion de la crise.

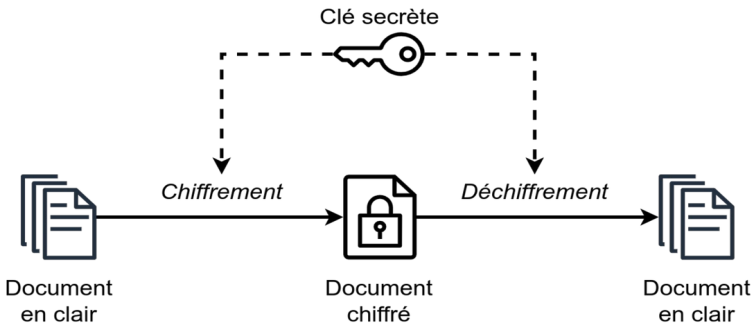
La vulnérabilité Heartbleed pouvait être exploitée sans authentification préalable, rendant toute application utilisant une version vulnérable d'OpenSSL susceptible d'être attaquée.

Depuis sa découverte, des correctifs ont été développés et il est essentiel pour les administrateurs système de s'assurer que leurs infrastructures utilisent des versions à jour d'OpenSSL pour prévenir de telles failles.

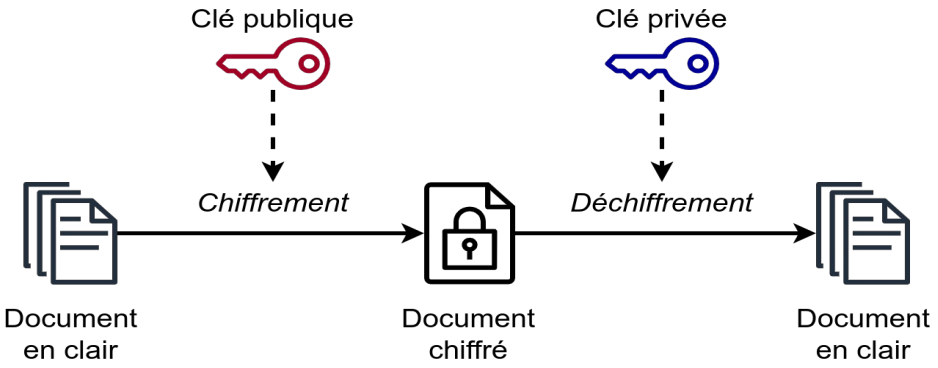
Défi n°2: force brute d'un mot de passe haché en MD5

Q1. Rappeler la différence entre chiffrement symétrique et chiffrement asymétrique et proposer un schéma pour chacune de ces deux méthodes.

- **Chiffrement symétrique** : une clé unique est utilisée pour chiffrer et déchiffrer le message. Avec cette méthode, le chiffrement est simple. Sa faiblesse réside dans la transmission de la clé à un correspondant.



- **Chiffrement asymétrique** :
  - Une **clé publique** : utilisée pour chiffrer les données et pouvant être partagée librement.
  - Une **clé privée** : utilisée pour déchiffrer les données et qui doit rester secrète.



Q2. Recenser et décrire les principaux algorithmes de chiffrement symétrique et asymétrique.

Algorithmes de chiffrement symétrique :

- **AES** (Advanced Encryption Standard) : Standard de chiffrement adopté mondialement,

offrant des longueurs de clé de 128, 192 ou 256 bits.
- **DES** (Data Encryption Standard) : Ancien standard avec une clé de 56 bits, considéré aujourd'hui comme obsolète en raison de sa vulnérabilité aux attaques par force brute.
- **3DES** (Triple DES) : Amélioration du DES en appliquant le chiffrement trois fois, offrant une sécurité accrue mais avec une performance réduite.
- **RC4** : Algorithme de flux rapide, mais désormais considéré comme non sécurisé en raison de plusieurs vulnérabilités découvertes.

Algorithmes de chiffrement asymétrique :

- **RSA** : L'un des premiers systèmes de chiffrement asymétrique, largement utilisé pour la sécurisation des données et les signatures numériques. Est désormais obsolète.
- **DSA (Digital Signature Algorithm)** : Utilisé principalement pour les signatures numériques, assurant l'authenticité et l'intégrité des messages.
- **ECC (Elliptic Curve Cryptography)** : Offre une sécurité équivalente à RSA avec des longueurs de clé plus courtes, ce qui le rend plus efficace en termes de calcul.

Q3. Qu'appelle t-on une fonction de hachage, quel peut-être son rôle lors du développement d'une application. Lister les principales fonctions de hachage et leurs principales caractéristiques.

Une **fonction de hachage** est une fonction mathématique qui prend en entrée des données de taille variable et produit une **empreinte**. Cette empreinte est unique pour chaque entrée distincte, et une petite modification de l'entrée entraîne une modification significative de l'empreinte.

Rôles dans le développement d'une application :

- **Stockage sécurisé des mots de passe** : Au lieu de stocker les mots de passe en clair, les applications stockent leur empreinte, ce qui protège les mots de passe en cas de compromission de la base de données.
- **Vérification de l'intégrité des données** : Les fonctions de hachage permettent de vérifier si les données ont été modifiées en comparant les empreintes avant et après la transmission ou le stockage.
- **Signatures numériques** : Elles sont utilisées pour créer des signatures numériques, assurant l'authenticité et l'intégrité des messages ou des documents.

Principales fonctions de hachage et leurs caractéristiques :

- **MD5** : Produit une empreinte de 128 bits, mais considéré comme non sécurisé en raison de vulnérabilités aux collisions.
- **SHA-1** : Produit une empreinte de 160 bits, également considéré comme obsolète en raison de faiblesses découvertes.
- **SHA-256** : Partie de la famille SHA-2, produit une empreinte de 256 bits, largement utilisé et considéré comme sécurisé.
- **SHA-3** : La plus récente de la famille SHA, offrant des tailles d'empreinte variables et basée sur une structure différente des précédentes.

Q4. Expliquer ce que fait le code java suivant et quel est le problème de sécurité posé par ce code ?

Le code Java est un test unitaire qui calcule et vérifie la somme de contrôle (checksum) MD5 d'un fichier nommé *test_md5.txt*.

```java
@Test
public void givenFile_generatingChecksum_thenVerifying()
  throws NoSuchAlgorithmException, IOException {
    String filename = "src/test/resources/test_md5.txt";
    String checksum = "5EB63BBBE01EEED093CB22BB8F5ACDC3";

    MessageDigest md = MessageDigest.getInstance("MD5");
    md.update(Files.readAllBytes(Paths.get(filename)));
    byte[] digest = md.digest();
    String myChecksum = DatatypeConverter
      .printHexBinary(digest).toUpperCase();

    assertThat(myChecksum.equals(checksum)).isTrue();
}
```

Explication du code :

1. **Lecture du fichier** : Le chemin du fichier est spécifié par la variable `filename`. Le fichier est lu en entier, et son contenu est passé à l'objet `MessageDigest` pour calculer le hash MD5.
2. **Calcul du hash MD5** : L'objet `MessageDigest` est initialisé avec l'algorithme MD5. Le contenu du fichier est mis à jour dans l'objet `MessageDigest` à l'aide de la méthode `update`, puis le hash (digest) est calculé avec la méthode `digest()`.
3. **Conversion en format hexadécimal** : Le tableau d'octets résultant (`digest`) est converti en une chaîne hexadécimale en majuscules à l'aide de `DatatypeConverter.printHexBinary(digest).toUpperCase()`.
4. **Vérification** : Le hash calculé (`myChecksum`) est comparé à une valeur de checksum attendue (`checksum`) pour vérifier l'intégrité du fichier.

Problème de sécurité :

Le code contient le checksum en dur, il est facilement récupérable et fait face aux vulnérabilitées de l'algorithme MD5 suivantes:

- **Collisions** : MD5 est vulnérable aux attaques par collision, où deux entrées différentes peuvent produire le même hash. Cela signifie qu'un attaquant pourrait potentiellement modifier le contenu d'un fichier sans changer sa somme de contrôle MD5, compromettant ainsi l'intégrité des données.

- **Faible résistance aux préimages** : Il est possible de trouver une entrée qui correspond à un hash MD5 donné dans une rainbow table (table arc-en-ciel), ce qui compromet la fiabilité de

MD5 pour les signatures numériques et autres applications de sécurité.

Recommandation :

Il faudrait remplacer MD5 par des algorithmes de hachage plus sécurisés, tels que **SHA-256 ou SHA-3**, qui offrent une meilleure résistance aux collisions et aux attaques de préimage.

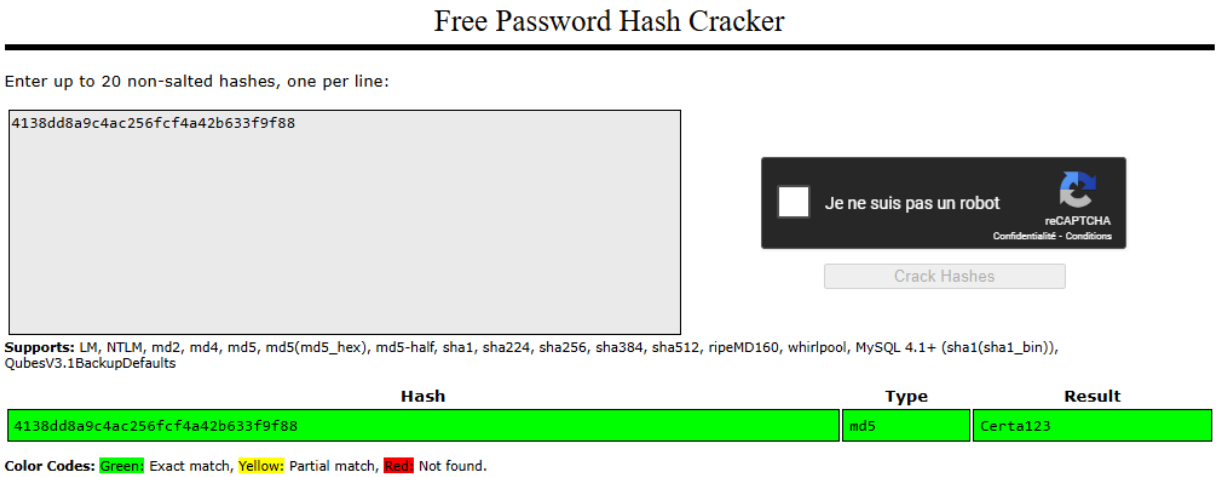
Q5. Dans le code Symfony suivant, expliquer le rôle de 'auto'. Conclure sur l'intérêt d'utiliser un framework lors du développement d'une application.

```yaml
# config/packages/security.yaml
security:
    # ...

    password_hashers:
        # auto hasher with default options for the User class (and children)
        App\Entity\User: 'auto'

        # auto hasher with custom options for all PasswordAuthenticatedUserInterface instanc
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
            algorithm: 'auto'
            cost:      15
```

Le "**auto**" est un hashage qui sélectionne automatiquement l'algorithme le plus sécurisé disponible sur votre système .

Q6. Un attaquant récupère le haché suivant dans une base de données. 4138dd8a9c4ac256fcf4a42b633f9f88

J'ai utilisé le site https://crackstation.net/ pour décrypter le mot de passe recupéré.


Free Password Hash Cracker
Enter up to 20 non-salted hashes, one per line:
4138dd8a9c4ac256fcf4a42b633f9f88
Je ne suis pas un robot
reCAPTCHA
Confidentialité - Conditions
Crack Hashes
Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults
Hash
Type
Result
4138dd8a9c4ac256fcf4a42b633f9f88
md5
Certa123
Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.