

Rapport d'audit ICT-183



David Dieperink, Stefan Petrovic – CID3A
CFPV

Table des matières

1	INTRODUCTION	3
2	AUTHENTIFICATION	3
2.1	DESCRIPTION.....	3
2.2	ÉLÉMENTS AUDITÉS.....	3
2.3	CORRECTIONS À APPORTER	4
3	XSS	5
3.1	DESCRIPTION.....	5
3.2	ÉLÉMENTS AUDITÉS.....	5
3.3	CORRECTIONS À APPORTER	5
4	CSRF	6
4.1	DESCRIPTION.....	6
4.2	ÉLÉMENTS AUDITÉS.....	6
4.3	CORRECTIONS À APPORTER	6
5	INJECTION SQL.....	7
5.1	DESCRIPTION.....	7
5.2	ÉLÉMENTS AUDITÉS.....	7
5.3	CORRECTIONS À APPORTER	7
6	PROTECTION DES INFORMATIONS	8
6.1	DESCRIPTION.....	8
6.2	ÉLÉMENTS AUDITÉS.....	8
6.3	CORRECTIONS À APPORTER	8
6.4	RECOMMANDATION OUTILS	10
6.4.1	OpenStego.....	10
7	WEBSINNERS	12
7.1	DESCRIPTION.....	12
7.2	ÉLÉMENTS AUDITÉS.....	12
7.2.1	Vulnérabilité High	12
7.2.2	Vulnérabilité Medium.....	12
7.3	CORRECTIONS À APPORTER	13
8	DDOS.....	14
8.1	DESCRIPTION.....	14
8.2	ÉLÉMENTS AUDITÉS.....	14
8.3	CORRECTIONS À APPORTER	14

1 INTRODUCTION

Ce document regroupe les différentes failles de sécurités trouvées sur le site d'e-commerce. Il décrit les failles de sécurité, explique leur fonctionnement et décrit différentes corrections mises en place avec leurs avantages et désavantages.

2 AUTHENTIFICATION

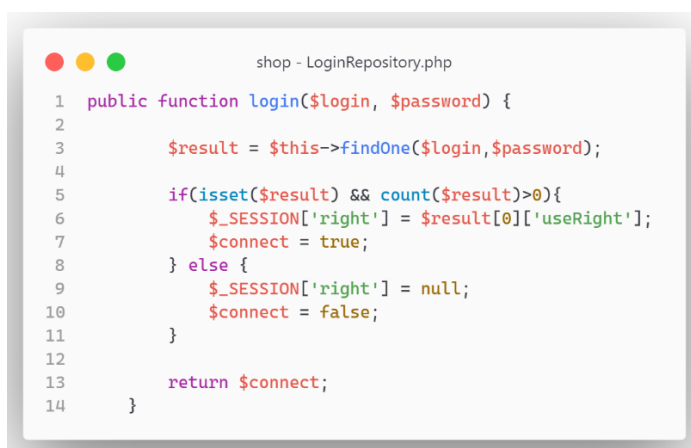
2.1 Description

L'authentification permet à un utilisateur de prouver son identité au site. Cela lui permet d'avoir des droits et des accès à des données du site.

2.2 Eléments audités

Voici les fonctions utilisées dans l'application pour l'authentification :

Figure 1



```
shop - LoginRepository.php
1 public function login($login, $password) {
2
3     $result = $this->findOne($login,$password);
4
5     if(isset($result) && count($result)>0){
6         $_SESSION['right'] = $result[0]['useRight'];
7         $connect = true;
8     } else {
9         $_SESSION['right'] = null;
10        $connect = false;
11    }
12
13    return $connect;
14 }
```

Figure 2



```
shop - LoginRepository.php
1 public function findOne($login,$password) {
2
3     $table = 't_user';
4     $columns = '*';
5     $where = "useLogin = '$login' AND usePassword = MD5('$password') ";
6
7     $request = new DataBaseQuery();
8
9     return $request->select($table, $columns, $where);
10
11 }
```

1. **Figure 1** : Un utilisateur est connecté à l'application quand il est trouvé directement dans la base de données en faisant une condition qui prend en compte le nom d'utilisateur et le mot de passe.
 - Il n'y a pas de vérification du mot de passe avec un sel, cela crée des failles. Une Rainbow table peut être utilisée.
 - Une vérification de la longueur du mot de passe n'existe pas, donc des **mots de passe par défaut** peuvent être utilisés (ex : admin, root).
2. **Figure 2** : La fonction utilisée pour effectuer la requête des informations de l'utilisateur dans la base de données fait un hachage avec le **MD5** mais ne prend en compte aucun sel.

2.3 Corrections à apporter

- Une vérification plus complète peut être effectuée en vérifiant si l'utilisateur existe en faisant une condition avec son nom d'utilisateur dans la requête SQL (fonction **findOne**) puis en utilisant la fonction PHP **password_verify** qui vérifie le hachage d'un mot de passe avec un sel. Cela permet de contourner la Rainbow table.

```
shop - LoginRepository.php
1  $result = $this->findOne($login);
2
3  if(isset($result) && count($result)>0){
4      if(password_verify($password, $result[0]['usePassword'])){
5          $_SESSION['right'] = $result[0]['useRight'];
6          $connect = true;
7      } else {
8          $_SESSION['right'] = null;
9          $connect = false;
10 }
```

- Une vérification sur la longueur du mot de passe doit également être implémentée pour éviter les mots de passe par défaut.

```
shop - LoginRepository.php
1  if(strlen($password) < 8){
2      return false;
3  }
```

- Pour contourner une attaque de **force brute**, la fonction PHP **sleep()** peut être utilisée pour mettre un délai lors de chaque tentative de connexion.

```
shop - LoginRepository.php
1  sleep(1);
2
3  $result = $this->findOne($login);
```

- Pour forcer HTTPS dans le serveur, un fichier **.htaccess** peut être ajouté dans la racine de l'application avec le contenu suivant :

```
www - .htaccess
1  RewriteEngine On
2  RewriteCond %{HTTPS} off
3  RewriteRule (.*?) https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

3 XSS

3.1 Description

XSS veut dire Cross-Site-Scripting. Une attaque XSS consiste à insérer un code malveillant dans un champ de formulaire par exemple. Si un code malveillant est inséré dans un champ de formulaire, le site va exécuter le code.

3.2 Éléments audités

Le site possède plusieurs formulaires. Ces formulaires sont sensibles aux attaques XSS. Lorsque cette ligne (<script>alert('XSS')</script>) est rentrée dans un champ de formulaire et que le formulaire est envoyé le script s'exécutera.

3.3 Corrections à apporter

- Pour éviter que cela se produise, une des solutions est de filtrer les informations reçues lors de l'envoi du formulaire. Si les informations reçues ne sont pas conformes à nos attentes, l'envoi du formulaire est refusé.

```
shop - OrderController.php

1  $reg = "/^[A-Za-zÀ-ÿ' ]*$/";
2  $regNPA = "/^[0-9]{4}$/";
3  $regStreetNB = "/^[0-9]{2,3}[A-z]?$/";
4
5  $errors = array();
6
7  if($_POST["title"] != "Monsieur" && $_POST["title"] != "Madame")
8      array_push($errors, "Titre");
9  if(!preg_match($reg, $_POST["name"]))
10     array_push($errors, "Nom");
11  if(!preg_match($reg, $_POST["firstname"]))
12     array_push($errors, "Prénom");
```

4 CSRF

4.1 Description

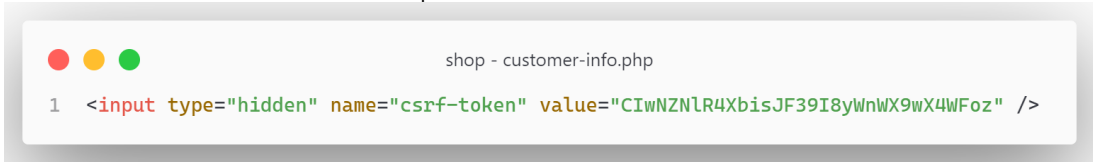
CSRF veut dire Cross-Site request forgery. Une attaque CSRF consiste à prendre le contrôle d'une session autorisée par l'utilisateur et grâce à cette session, les attaquants peuvent exécuter des actions malveillantes. Cette attaque par le biais de requête HTTP.

4.2 Éléments audités

Le site possède plusieurs formulaires qui sont sensible. Aucun champ ne permet réellement de prouver que c'est la bonne personne qui rentre les informations et qui exécute les requêtes HTTPS.

4.3 Corrections à apporter

- Pour éviter que cela se produise, une des solutions est d'utiliser un framework comme Laravel qui prend en charge automatiquement les attaque CSRF. Il existe également une autre option, celle-ci consiste à générer un token aléatoire avec un délai d'expiration.



```
shop - customer-info.php
1 <input type="hidden" name="csrf-token" value="CIwNZNlR4XbisJF39I8yWnWx9wX4wFoz" />
```

5 INJECTION SQL

5.1 Description

Une injection SQL est un type d'attaque assez courant. Cette attaque consiste à faire passer une requête SQL dans un champ de formulaire. Par exemple dans un formulaire de connexion, cela permet de bypass l'authentification si le formulaire est mal protégé.

5.2 Eléments audités

Dans plusieurs formulaires du site, les champs n'étaient pas vérifiés donc les injection SQL passaient. Ces formulaires sont celui du login, de l'insertion d'un article et de l'update d'un article.

5.3 Corrections à apporter

- Pour éviter que des injections SQL se produisent, pour la vérification du login nous allons check si les informations reçues contiennent des simple guillemets « ' ». Si le champ « login » et le champ « password » ne contiennent pas de simples guillemets, la personne arrive à se connecter.

```
shop - LoginController.php
1  if(!str_contains($_POST['login'], "'") && !str_contains($_POST['password'], "'")){
2
3      $login = $_POST['login'];
4      $password = $_POST['password'];
5
6      $loginRepository = new LoginRepository();
7      $result = $loginRepository->login($login, $password);
8  }
```

- Nous effectuons la même vérification sur le formulaire d'insertion des articles. On vérifie que les champs « name », « description », « prix » et « quantité » ne possède pas de simples guillemets.

```
shop - AdminRepository.php
1  if(!str_contains($name, "'") && !str_contains($description, "'") && !str_contains($price, "'") && !str_contains($quantity, "'"))
```

6 PROTECTION DES INFORMATIONS

6.1 Description

Les informations que possède un site web sont très peu sécuriser, par exemple les images, les fichiers que possède le site, etc. Les images sont très peu sécurisées car avec un simple screenshot l'image peut être volée.

6.2 Eléments audités

Les images ne possèdent pas de filigrane (watermark en anglais), ce qui permet à des utilisateurs mal intentionnés de potentiellement voler nos images à l'aide de screenshot.

De plus certains formulaires ne sont pas immunisés contre les uploads de fichiers malveillants. Apache exécute tous les fichiers qui possèdent l'extension « .php ». Donc si une image s'appelle « image.php.png » Apache va exécuter le code php que possède l'image.

Il y a également un autre problème dans la page de contact, l'adresse mail est écrite en dur, cela permet à des scrappeurs web de récupérer l'adresse mail et d'envoyer des spams.

6.3 Corrections à apporter

- Tout d'abord il faut mettre un filigrane sur nos images. Cela se fait à l'aide d'un fichier PHP qui génère du texte et qui le met sur l'image de base. On récupère l'image, on définit une couleur pour le texte du filigrane et ensuite on définit sa position et son texte. Si l'image est valide on l'affiche.

```
shop - watermark.php
1  <?php
2  error_reporting(E_ALL);
3
4  ini_set('display_errors', TRUE);
5
6  $img=imagecreatefrompng($_GET['jpg']);
7  $red = imagecolorallocatealpha($img,255,0,0,0);
8  imagefttext($img, 20, -40, 40, 60, $red, "C:\Windows\Fonts\Calibri.ttf", "COPYRIGHT");
9
10 if(!$img){
11     echo "erreur";
12 }
13 else{
14     imagepng($img);
15 }
16 ?>
```

- Pour l'affichage on doit mettre ce lien, on transmet le lien de l'image au fichier qui va générer le filigrane.

```
shop - list.php
1  <img src="view/page/shop/watermark.php?jpg=../../resources/image/' . $product['proImage']"
```

- Pour l'upload d'image, il faut supprimer le « .php » du nom du fichier, cela permet d'éviter l'exécution de ce fichier par Apache. Pour le supprimer, une des manières est la suivante, je remplace le « .php » par une chaîne de caractère vide :

```
shop - AdminController.php
1  $_FILES['productFile']['name'] = str_replace(".php", "", $_FILES['productFile']['name']);
```


- Pour contrer les spam mail à cause d'un scrappeur web, il suffit d'utiliser un bouton qui lorsqu'il est appuyé, il affiche l'adresse mail bout par bout grâce à un script JavaScript.

```
shop - contact.php
1 <button onclick="displayMail(this)">Afficher l'email</button>
```

```
shop - mail.js
1 function displayMail(btn){
2     document.getElementById("mail").innerText += "info";
3     document.getElementById("mail").innerText += "@";
4     document.getElementById("mail").innerText += "shop";
5     document.getElementById("mail").innerText += ".shop";
6     btn.remove();
7 }
```

- Une autre solution peut être envisageable, il s'agirait d'implémenter un formulaire de contact qui serait envoyé que lorsque le captcha de vérification est vérifié, cela évite aussi les spam car validé les captchas avec un BOT est compliqué.

6.4 Recommandation outils

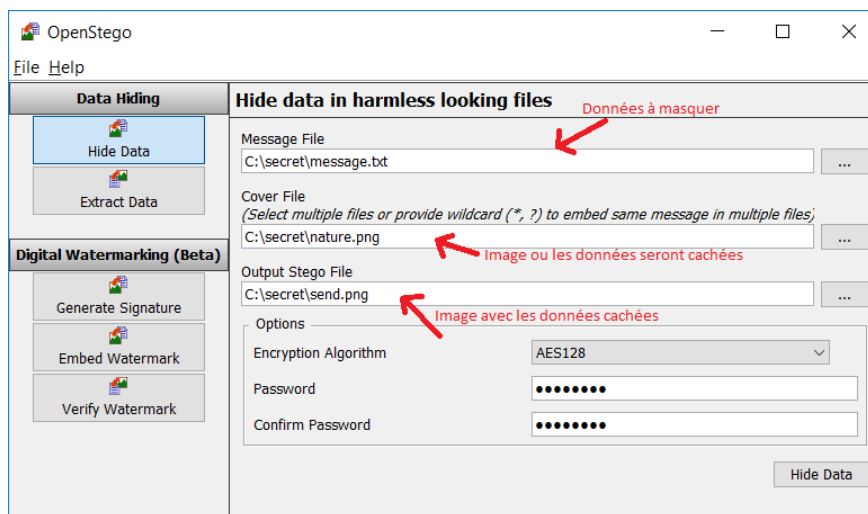
Il existe plusieurs logiciels qui permettent d'ajouter des signatures dans des pixels invisibles sur le marché. Par exemple, Digimarc Guardian qui s'utilise avec Photoshop. L'outil que nous vous recommandons s'appelle [OpenStego](#).

6.4.1 OpenStego

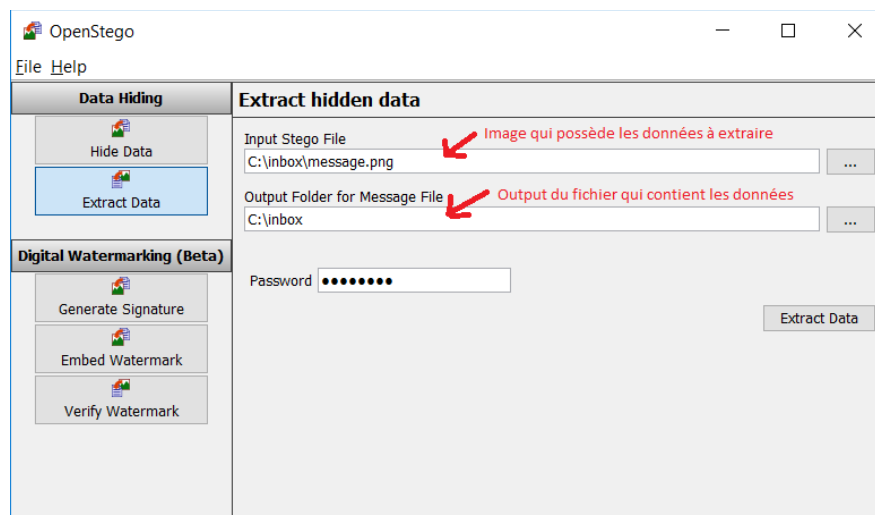
Cet outil propose 2 fonctionnalités principale, la première c'est qu'il permet de cacher toutes les données d'un fichier, la deuxième est un système de filigrane. Il ajoute une signature invisible sur le fichier.

Pour la fonctionnalité du « Masquage de données » vous pouvez soit masquer les données du fichier à l'intérieur d'une image, soit extraire les données de l'image.

- Masquer les données à l'intérieur d'une image :

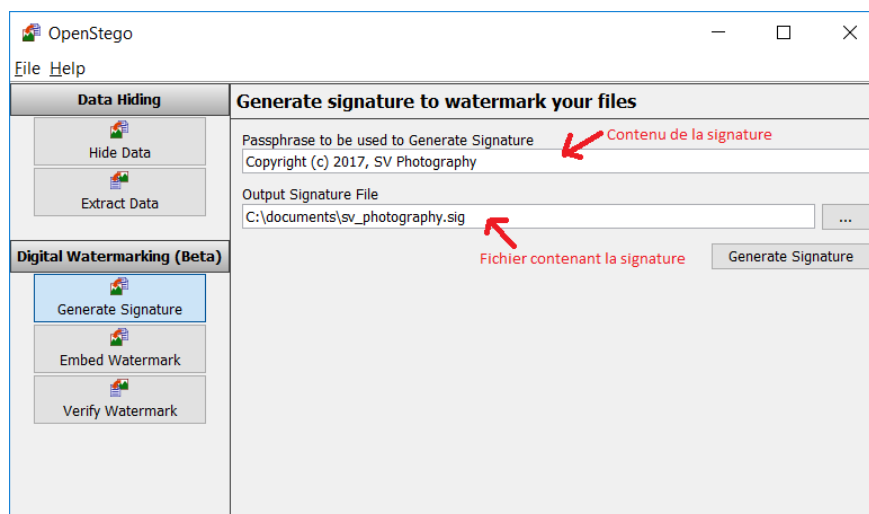


- Extraire les données d'une image :

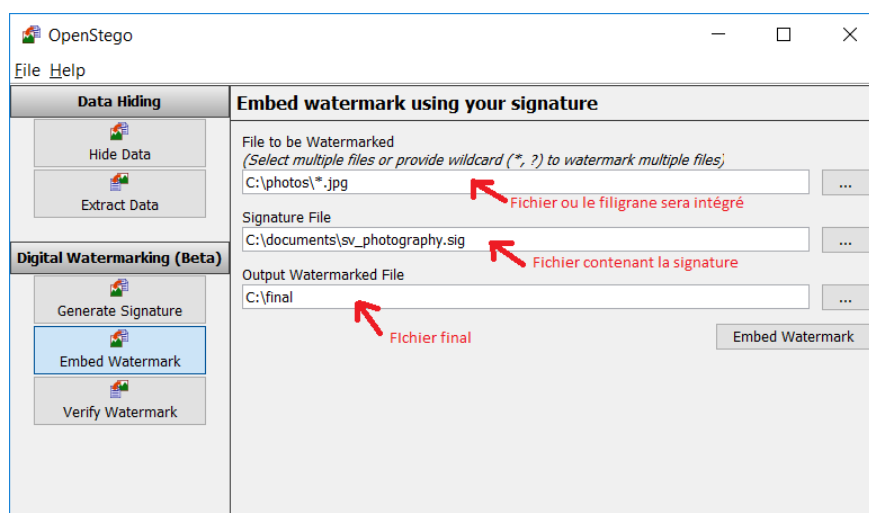


Pour la fonctionnalité du « Filigrane » vous pouvez générer une signature qui sera votre filigrane, ensuite vous pourrez l'insérer dans l'image et vous pourrez vérifier que votre image contient votre signature.

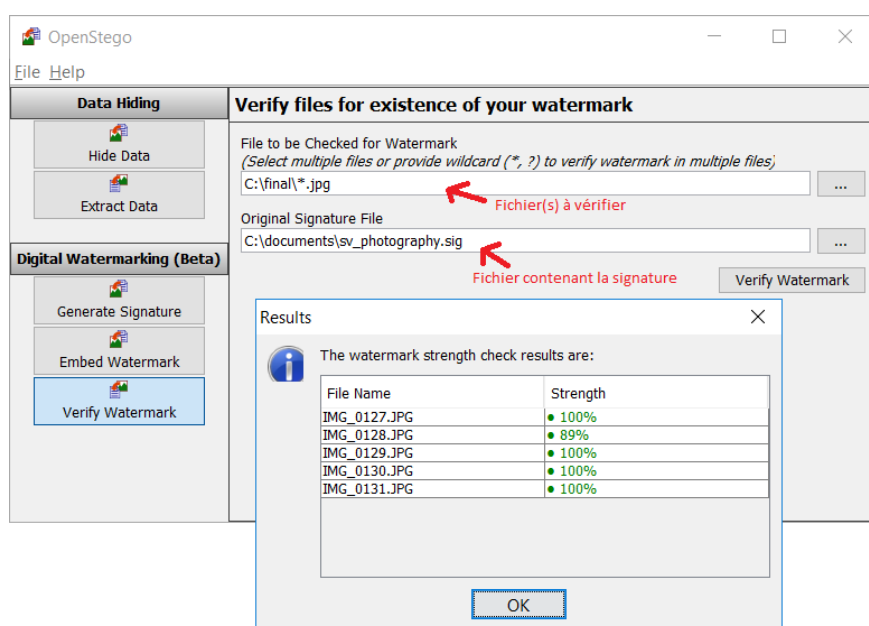
- Générer une signature :



- Intégrer le filigrane, le filigrane peut être ajouter à un ou plusieurs fichier :



- Vérifier le filigrane :



7 WEBCANNERS

7.1 Description

Les Webscanners permettent de scanner notre site web de A à Z. Il effectue un audit. Dans notre cas le webcanner VEGA a été utilisé.

7.2 Eléments audités

Le logiciel VEGA nous à retourner un rapport de son audit. Le voici :



Scan Alert Summary

High	(4 found)
Session Cookie Without Secure Flag	1
Session Cookie Without HttpOnly Flag	1
MySQL Error Detected - Possible SQL Injection	1
SQL Injection	1
Medium	(35 found)
Certificate signed using SHA-1	1
Client Ciphersuite Preference	1
HTTP Trace Support Detected	1
Local Filesystem Paths Found	2
PHP Error Detected	30
Low	(18 found)
Directory Listing Detected	16
Form Password Field with Autocomplete Enabled	2
Info	(5 found)
Self-Signed Certificate	1
Interesting Meta Tags Detected	3
Character Set Not Specified	1

7.2.1 Vulnérabilité High

VEGA nous signale qu'il y'a 4 vulnérabilités majeure.

- La première et la deuxième sont liée au cookie de session. Le cookie à été set (défini) sans être sécurisé.
- La troisième est une erreur MySQL, il à potentiellement détecté une injection SQL.
- La dernière c'est indiqué que c'est une injection SQL mais en réalité c'est un faux positif. Les erreurs ne sont pas liées é une injection SQL.

7.2.2 Vulnérabilité Medium

VEGA nous signale qu'il y'a 35 vulnérabilités medium.

- Les erreurs PHP sont liées à des variables pas définie ou à des tentatives d'injection SQL. Les injections SQL génèrent des erreurs car dans la balise « src » il ne comprend pas les « > » qui remplacent les « > ».
- Les erreurs liées aux variables sont dues au fait que les fonctions ne connaissent pas les variables.
- L'erreur liée au certificat signé avec le SHA-1 est en rapport avec la configuration Apache.

```
<b>Warning</b>: Undefined global variable $_SESSION in <b>D:\CFC\Outils\Uw&www\etml-183-secu\shop\view\page\order\order-confirmed.php</b> on line <b>
```

7.3 Corrections à apporter

- Pour le cookie de session il faut ajouter 2 lignes dans le fichier .htaccess. L'ajout de ces 2 lignes permet de sécuriser le cookie. Suite à cela, le cookie sera utilisé seulement avec le protocole HTTPS et non le HTTP.

```

● ● ● etml-183-secu - .htaccess

1  php_value session.cookie_httponly 1

2  php_value session.cookie_secure 1

```

- L'audit effectué après ce patch par VEGA :



Scan Alert Summary

High		(2 found)
MySQL Error Detected - Possible SQL Injection	1	
SQL Injection	1	
Medium		(35 found)
Certificate signed using SHA-1	1	
Client Ciphersuite Preference	1	
HTTP Trace Support Detected	1	
Local Filesystem Paths Found	2	
PHP Error Detected	30	
Low		(18 found)
Directory Listing Detected	16	
Form Password Field with Autocomplete Enabled	2	
Info		(5 found)
Self-Signed Certificate	1	
Interesting Meta Tags Detected	3	
Character Set Not Specified	1	

8 DDOS

8.1 Description

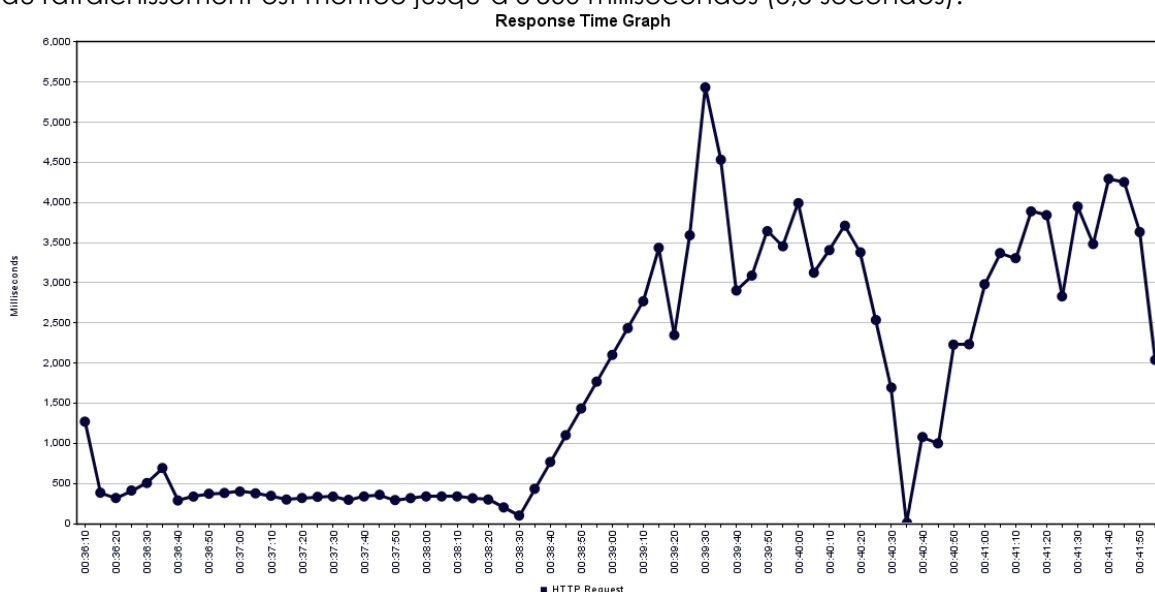
Tout d'abord DDOS veut dire Distributed Denial of Service, en français Dénî de Service Distribué. Cette attaque consiste à envoyer des multiples requêtes à un site web dans le but de le mettre hors d'utilisation. Cela veut dire que le site recevra tellement de requêtes qu'il ne pourra plus les supporter.

8.2 Eléments audités

Les paramètres mis sur Jmeter sont les suivants : 10'000 utilisateurs qui envoient des requêtes HTTP à l'infini sur le site web. Le site est tombé plusieurs fois lors du test.

Thread Properties	
Number of Threads (users):	10000
Ramp-up period (seconds):	30
Loop Count:	<input checked="" type="checkbox"/> Infinite

Ce graphique représente le temps de réaction du site lors du DDOS. On voit que la courbe fait des hauts et des bas, on remarque que à un moment donné la durée du rafraîchissement est montée jusqu'à 5'500 millisecondes (5,5 secondes).



8.3 Corrections à apporter

Pour contrer le DDOS, l'entreprise Cloudflare propose un service gratuit qui permet de contrer ce type d'attaque.

Leur solution anti-DDOS sécurise les sites web, les applications et les réseaux. Ils sont leader dans le domaine.

Lien sur l'article de Cloudflare : <https://www.cloudflare.com/fr-fr/ddos/>