2024-2025

TASKSTEP : AUDIT D'OPTIMISATION DE SECURITE

1PEUDINSPI :

- Tristan DAL MOLIN
- Moulay-Wassim ALAOUI
- Jules DUTRION
- Matteo DE MARCO
- Wassim DIOURI

BUT2 Informatique

IUT DIJON AUXERRE NEVERS
UNIVERSITÉ DE BOURGOGNE
Révélateur de talents depuis 1968

UB
UNIVERSITÉ DE BOURGOGNE

I. Remise en contexte

Liste des types d'utilisateurs

Il n'existe qu'un type d'utilisateur : ceux qui verront leurs tâches être affichées dans Taskstep

Liste des composants de l'application

Nom du composant	Localisation	Restriction d'accès
BDD	Lieu d'hébergement du serveur	Administrateur du service et serveur
Serveur	Lieu d'hébergement du serveur	Administrateur du service
Client	Chez l'utilisateur	Public

À noter que l'application peut être utilisée localement, auquel cas tout se trouvera sur la même machine.

Liste des données manipulées

Données	**Accès en lecture**			**Accès en écriture**		
	Utilisateurs cibles	**Risques**	**Sensibilité**	**Utilisateurs cibles**	**Risques**	**Sensibilité**
Login de l'utilisateur	Utilisateur concerné	X	Pas sensible	X	Perte de fonctionnalité	Sensible
Mot de passe	Utilisateur concerné	Usurpation d'identité	Très sensible	Utilisateur concerné	Perte de l'usage du mot de passe	Très sensible
Item	Utilisateur concerné	X	Sensible	Utilisateur concerné	X	Aucune sensibilité
Paramètre	Utilisateur concerné	X	Pas de risque	Utilisateur concerné	X	Aucune sensibilité

II. Analyse des communications

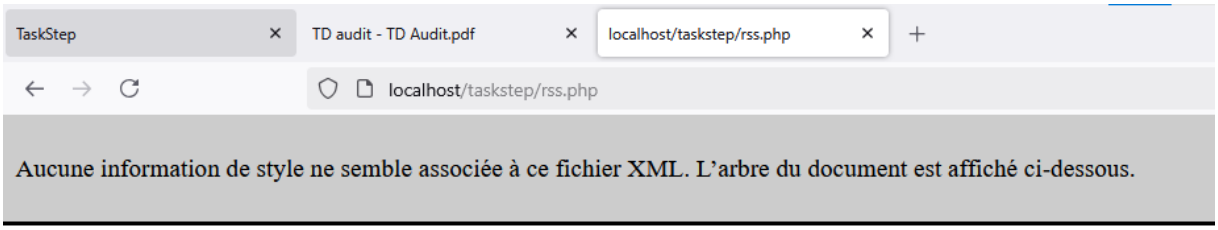
Liste des communications entre composants

Communication de ... vers ...	Type de communication	Données transmises	Risques	Sensibilité	Protocole utilisé
Client vers serveur	HTTP	Données web	Interception mot de passe / item	Très sensible	HTTP
Serveur vers BDD	SQL	Données SQL	Interception mot de passe / item	Très sensible	MySQL

Analyse des vulnérabilités

La communication se passe sans chiffrement donc les mots de passe se retrouve en clair et peuvent être intercepté lors de la communication. Il pourrait y avoir un problème pour intercepter les requêtes du côté serveur si le mot de passe protégeant la base.

De même, le serveur ne limite en rien les requêtes que l'utilisateur a le droit de faire ou non, impliquant qu'on peut ouvrir n'importe quel fichier sur le serveur. Cela n'est, pour le moment, pas critique car nous ne stockons rien de dangereux, mais si jamais nous venions à stocker des fichiers plus sensibles (base de données sous fichier db par exemple), l'utilisateur pourrait y avoir accès sans même être connecté en trouvant simplement le bon lien.




```
−<rss version="2.0">
  −<channel>
      <title>TaskStep</title>
      <link>http://localhost/taskstep/</link>
      <description>TaskStep Items Feed</description>
      <language>en-us</language>
      <generator>IceMelon RSS Feeder</generator>
    −<item>
        <title>Sample task</title>
        <link>http://localhost/taskstep/edit.php?id=1</link>
        <description>2007-08-27 | SampleProject | SampleContext | Notes</description>
      </item>
  </channel>
</rss>
```

Recommandations

Passer les communications entre le client et le serveur par https pourrait donc éviter les différentes failles, niveau de priorité critique.

Limiter l’accès aux fichiers pour l’utilisateur à l’aide de fichier .htaccess. Faille pour le moment peu critique mais pouvant le devenir selon ce qu’on stocke sur le serveur et étant facilement réparable.

Optimisation

Par manque de temps nous n’avons pas pu remédier à ce problème, mais il aurait fallu utiliser le principe de chiffrement qu’offre le https par le chiffrement SSL ce qui aurait en cas d’interception des requêtes par exemple sur un serveur public. Actuellement l’utilisation d’un curl permet de lire le contenu du mot de passe.

III. Analyse de la gestion de permissions

Analyse de l'identification

La connexion est faite par un mot de passe en simple facteur. Le mot de passe n'a pas de règle détaillée (ni de taille imposée, de symboles ou majuscule, de durée de vie, etc.), et rien n'est mis en place dans le cas où l'utilisateur oublierait son mot de passe avant de se connecter.

Au moment de la connexion, on peut essayer autant de mots de passe que l'on veut, ce qui rend donc le site vulnérable aux brutes forces.

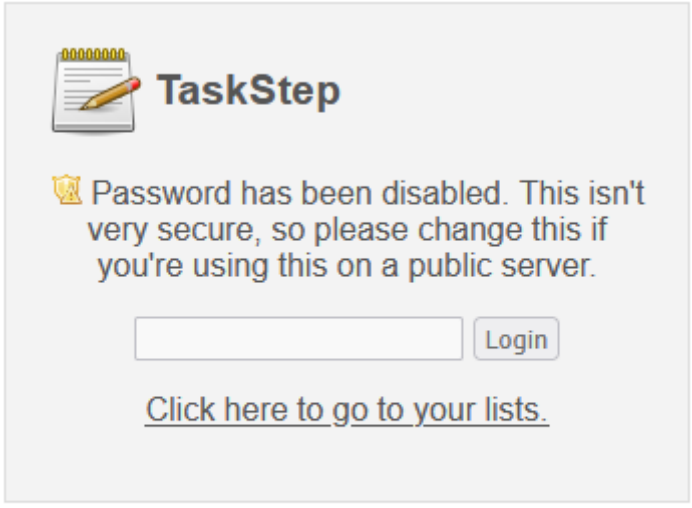
Exemple de brute force : on utilisera le logiciel Hydra via invite de commande sur WSL a été utilisé pour faire les tests liés aux brutes force avec une wordlist contenant le mot de passe pour permettre la connexion via le mot de passe :

```
jules@DESKTOP-II5QK5U:/mnt/c/Users/myste/Downloads$ hydra -l "" -P wordlist_fr_5d.txt -f 192.168.56.1 http-post-form "/taskstep-mas
ter/login.php:password=^PASS^&submit=Login:Incorrect password."
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for ille
gal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-16 09:26:58
[DATA] max 16 tasks per 1 server, overall 16 tasks, 11875 login tries (l:1/p:11875), ~743 tries per task
[DATA] attacking http-post-form://192.168.56.1:80/taskstep-master/login.php:password=^PASS^&submit=Login:Incorrect password.
[80][http-post-form] host: 192.168.56.1   password: taskstep
[STATUS] attack finished for 192.168.56.1 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-16 09:27:17
```

Le mot de passe "taskstep" a été placé dans les millièmes lignes et le mot de passe a donc fini par être cracké de cette façon.

Le mot de passe est stocké haché et salé en base de données.

De même, il existe un mode de fonctionnement du site qui ne nécessite aucun mot de passe, ce qui, dans le cadre d'une application multi-utilisateurs, est très dangereux.

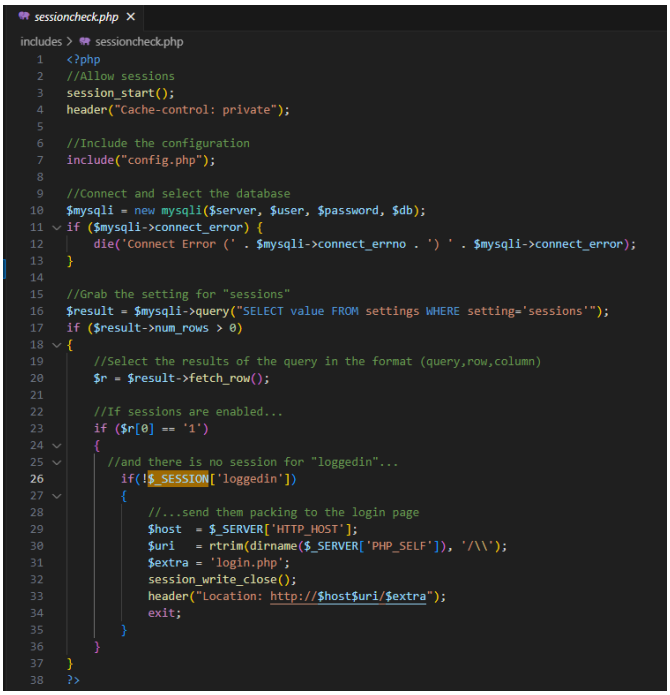


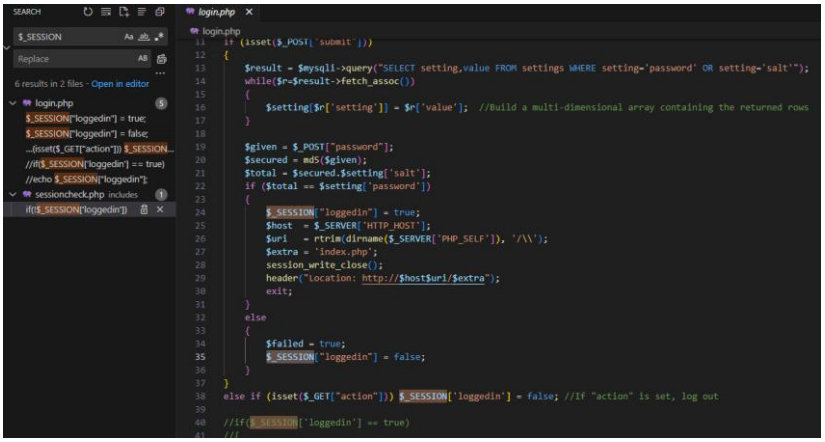
Analyse de l’authentification

L’utilisateur ne peut pas modifier ses propres droits dans l’application, uniquement son mot de passe.

L’application ne génère rien pour gérer l’authentification de l’utilisateur (ni token, ni session créée pour limiter dans le temps la connexion).

Il n’y a aucune expiration à la connexion de l’utilisateur : une fois connecté, on l’est pour toujours (voir figures ci-dessous pour preuves dans le code)





Recommandation

Concernant les attaques par brutes force, il faudrait mettre un compte à rebours entre chaque erreur de mot de passe pour rendre l’attaque lente et en définitive décourager les attaquants, en plus d’intégrer une limite d’un certain nombre d’essais par heure ou jour afin d’empêcher de telles attaques d’arriver. En effet actuellement le site peut facilement être attaqué par brute force

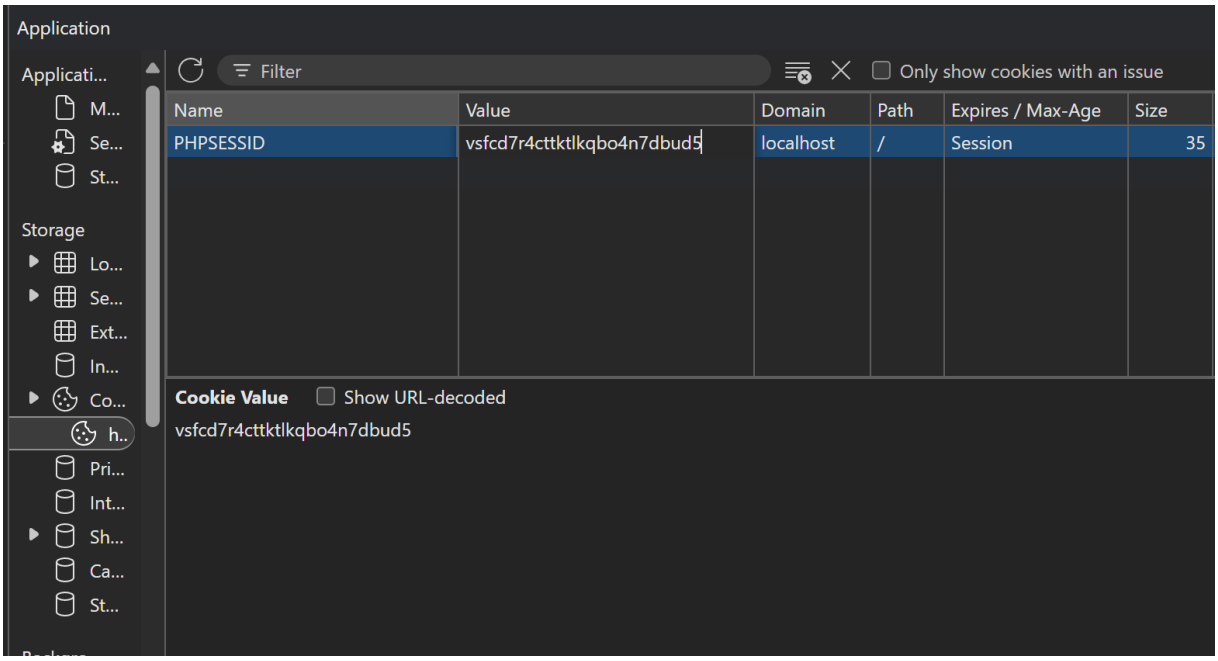
via n'importe quel logiciel, et l'efficacité peut être démultipliée par l'utilisation de "wordlist" complète.

De même, la connexion de l'utilisateur ne s'arrête jamais, il serait nécessaire d'intégrer une expiration à sa connexion afin d'éviter d'éventuelles intrusion sur le site à l'aide d'une simple connexion maintenue ouverte.

Le mode sans mot de passe du site web sera également à retirer pour renforcer sa sécurité.

Enfin, le mot de passe en lui-même pourrait nécessiter des ajustements tels qu'imposer majuscule, chiffre et caractère spécial voire une durée de vie au bout de laquelle il faudrait changer de mot de passe.

Le site rencontre aussi une nouvelle faille. Il y a un cookie sur le site qui permet de stocker l'identifiant d'une session PHP qui n'est pas supprimée. En modifiant la valeur du cookie par un id de session PHP qui est stockée, la page de connexion peut donc être directement contournée et les actions du site deviennent directement accessibles.



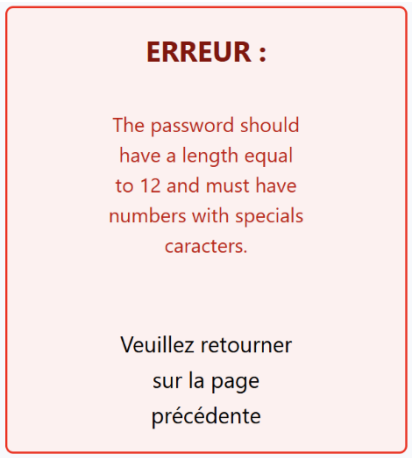
Optimisation

Pour la partie de la connexion se déroulant lors de la saisie d'un ID de connexion dans le cookie à l'attribut PHPSESSID, il aurait fallu mettre en place un système de validité de la session et que chaque session soit reliée par une id et donc effective pour une seule machine. Mais par manque de temps nous n'avons pas réussi à le mettre en place.

En revanche, concernant la partie du brute force nous avons pu mettre en place différentes vérifications pour retarder ce processus un maximum mais pas totalement le bloquer, nous aurions eu besoin de davantage de temps et de moyens pour limiter ces cas.

La première chose a été de mettre en place une norme de mot de passe donc celle qu'on a utilisée est une taille de 12 caractères avec au moins une minuscule, une majuscule, un chiffre

et un symbole. Ce qui fait donc passer le nombre de combinaison possible à minimum environ $1{,}32.10^{22}$ possibilités.



Ceci se faisait à l'aide d'une vérification par expression régulière comme décrit sur le code suivant :

```php
/**
 * Method to verify a password
 * @param string $password The password which must be verified
 * @return bool If a password is valid or not
 */
2 references | 0 overrides
function isValidPassword(string $password) : bool {
    return preg_match(pattern: '/^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{12,}$/', subject: $password);
}
```

Ce qui prendrait par une attaque par brute force avec un ordinateur traditionnel milieu de gamme environ 1 millier d'années à être craqué sans utiliser un système de wordlist qui pourrait donc éliminer énormément de combinaison rapidement si cette dernière est bien faite, ce qui sera énormément ralenti par le deuxième système que nous avons mis en place.

Le deuxième système qui a été mis en place et un système de pause lors de la saisie d'un mot de passe. La pause ayant été définie sur 2 secondes à chaque tentative pour ne pas nuire au confort utilisateur. L'utilisation de cette mesure permet donc de limiter les essais à donc un essai par seconde, du fait que la réponse indiquant si le test est passé ou pas met 2 secondes à apparaître en cas d'échec de connexion.

Pour obtenir une preuve d'efficacité de cette mesure, des tests ont été réalisé avec le logiciel Hydra avec les mêmes conditions que celle pour l'audit, c'est-à-dire que le mot de passe a été glissé dans une wordlist d'environ 11 000 éléments. Afin d'obtenir des chiffres assez parlant le mot de passe validant la connexion a été placé à la toute fin de la wordlist.

```
jules@DESKTOP-II5QK5U:/mnt/c/Users/myste/Downloads$ hydra -I -l julesDutrion -P wordlist_fr_5d.txt -f -vV 192.168.56.1 http-post-fo
rm "/TaskStepV2/index.php?action=ConnexionPage:login=^USER^&password=^PASS^&validate=:S=Bienvenue"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for ille
gal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-05-15 15:06:40
[WARNING] Restorefile (ignored ...) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 11874 login tries (l:1/p:11874), ~743 tries per task
[DATA] attacking http-post-form://192.168.56.1:80/TaskStepV2/index.php?action=ConnexionPage:login=^USER^&password=^PASS^&validate=:
S=Bienvenue
```

Comme affiché sur cette capture, le lancement de l'attaque par brute force a été lancé à 15h 06min 40s et a été fini comme l'est indiqué dans la capture d'écran à 15h 11min 12s.

```
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "6364AbAc/+/-" - 11874 of 11874 [child 8] (0/0)
[STATUS] attack finished for 192.168.56.1 (waiting for children to complete tests)
[80][http-post-form] host: 192.168.56.1   login: julesDutrion   password: 6364AbAc/+/-
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-05-15 15:11:12
```

Ce qui donne en définitive un temps pour les 11 874 essais de 4 min et 32 secondes.

Or en rajoutant ladite pause de deux secondes entre chaque essai, le temps est passé d'environ 4 minutes à plus de 31 minutes.

```
jules@DESKTOP-II5QK5U:/mnt/c/Users/myste/Downloads$ hydra -I -l julesDutrion -P wordlist_fr_5d.txt -f -vV 192.168.56.1 http-post-fo
rm "/TaskStepV2/index.php?action=ConnexionPage:login=^USER^&password=^PASS^&validate=:S=Bienvenue"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for ille
gal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-05-15 16:07:44
[WARNING] Restorefile (ignored ...) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 11874 login tries (l:1/p:11874), ~743 tries per task
[DATA] attacking http-post-form://192.168.56.1:80/TaskStepV2/index.php?action=ConnexionPage:login=^USER^&password=^PASS^&validate=:
S=Bienvenue
```

```
[STATUS] 382.39 tries/min, 11854 tries in 00:31h, 20 to do in 00:01h, 16 active
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "voyons" - 11855 of 11874 [child 12] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "voyou" - 11856 of 11874 [child 2] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "vrac" - 11857 of 11874 [child 7] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "vrai" - 11858 of 11874 [child 0] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "vue" - 11859 of 11874 [child 13] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "wagon" - 11860 of 11874 [child 14] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "western" - 11861 of 11874 [child 5] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "whisky" - 11862 of 11874 [child 15] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "yacht" - 11863 of 11874 [child 8] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "yankee" - 11864 of 11874 [child 6] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "yaourt" - 11865 of 11874 [child 3] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "yeux" - 11866 of 11874 [child 11] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "yoga" - 11867 of 11874 [child 4] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "zen" - 11868 of 11874 [child 10] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "zinc" - 11869 of 11874 [child 9] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "zingaro" - 11870 of 11874 [child 1] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "zizi" - 11871 of 11874 [child 12] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "zone" - 11872 of 11874 [child 2] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "zoo" - 11873 of 11874 [child 7] (0/0)
[ATTEMPT] target 192.168.56.1 - login "julesDutrion" - pass "6364AbAc/+/-" - 11874 of 11874 [child 0] (0/0)
[STATUS] attack finished for 192.168.56.1 (waiting for children to complete tests)
[80][http-post-form] host: 192.168.56.1   login: julesDutrion   password: 6364AbAc/+/-
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-05-15 16:38:47
```

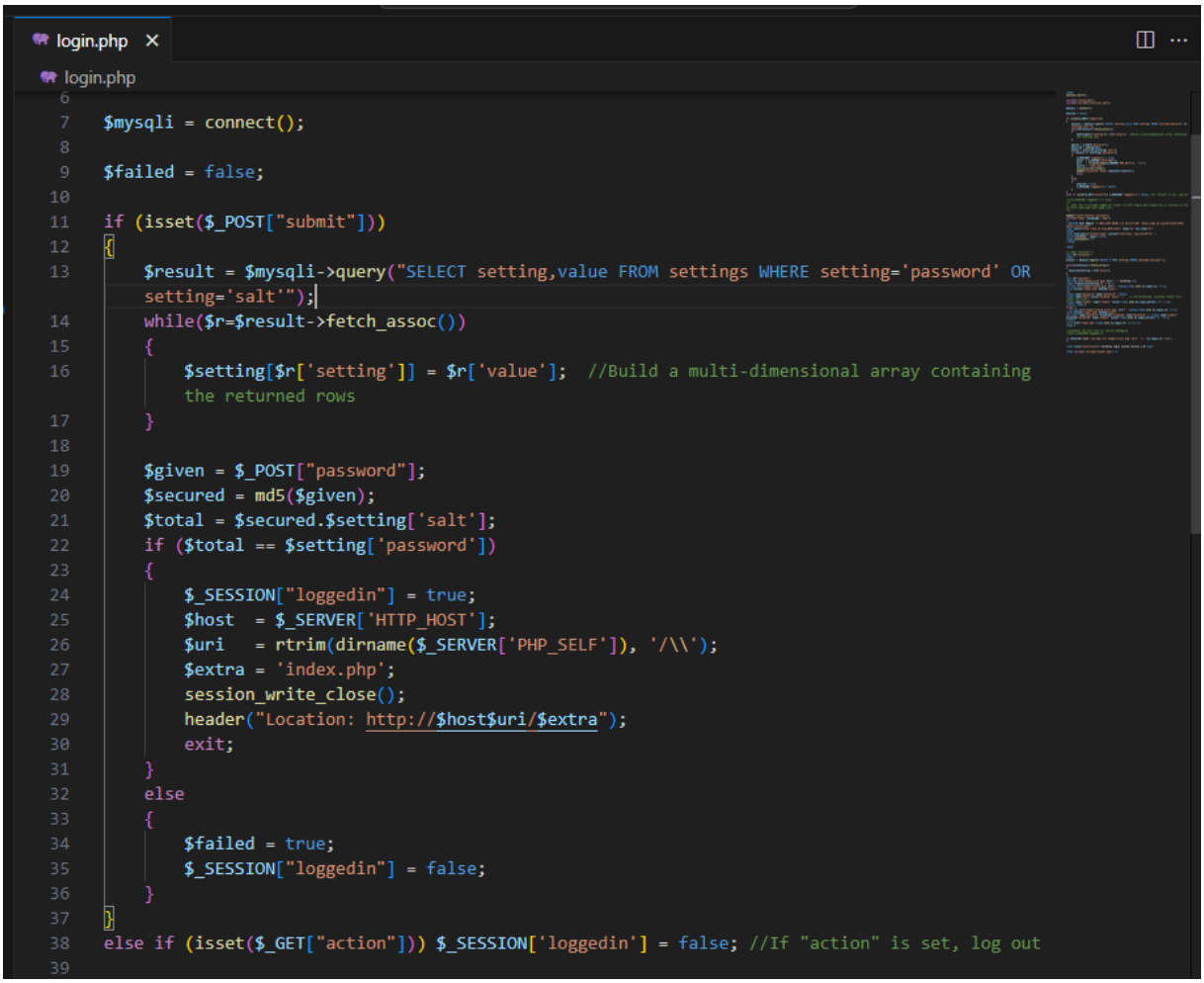
Même si les conditions ont été différentes entre les deux tests, du fait que le logiciel utilisé Hydra, a utilisé 16 instances en simultanés pour pouvoir contourner au mieux cette restriction ce qui donnait 16 tests toutes les deux secondes et non 1 seuls.

```php
public function Login(string $login, string $password): void
{
    $connected = false;

    if(!isset($_SESSION["connexionAttempts"]))
        $_SESSION["connexionAttempts"] = 0;

    if (isset($_SESSION["lastConnexionAttempts"]) && ($_SESSION["lastConnexionAttempts"] + TENTATIVES_TIMEOUT <= time()))
        unset($_SESSION["lastConnexionAttempts"]);

    if(!isset($_SESSION["lastConnexionAttempts"]))
    {
        if ($this->userService->Connexion(login: $login,password: $password))
        {
            $_SESSION["username"] = $login;
            $_SESSION["userID"] = $this->userService->GetId(login: $login);
            $connected = true;
        }
        else
        {
            sleep(seconds: 2);
            $_SESSION["connexionAttempts"] += 1;
            if ($_SESSION["connexionAttempts"] >= TENTATIVES_CONNEXIONS)
            {
                $_SESSION["lastConnexionAttempts"] = time();
                $_SESSION["connexionAttempts"] = 0;
            }
        }
    }

    if ($connected) {
        (new ControllerHome())->index();
    }
    else {
        $this->page->setMsg(msg: new Message(message: "ConnectionFailed",error: true));
        echo $this->page->GeneratePage();
    }

}
```

IV. Analyse du stockage des informations

Vérification des données stockées

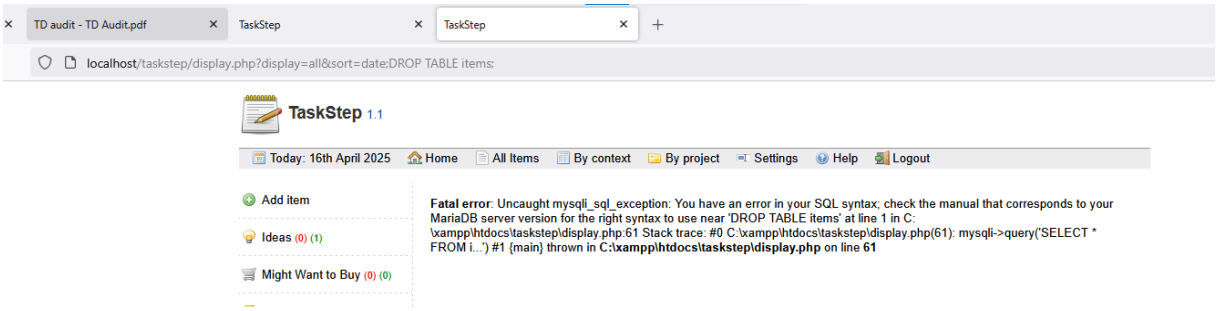
Une connexion est mise en place pour ce qui est de la connexion : étant donné qu'il n'existe qu'un seul mot de passe à l'application, il est récupéré en BDD et ensuite comparé au mot de passe entré, ce qui évite une possible injection SQL. Il est cependant haché en MD5, qui n'est plus un algorithme de hachage sécurisé.



Pour le reste des requêtes, aucune requête de sélection n'est protégée face aux injections et est donc à sujet de nombreuses failles dans la base de données.

```php
$display = (isset($_GET["display"])) ? $_GET["display"] : '';
$sortby = (isset($_GET["sort"])) ? $_GET["sort"] : 'date';
$section = (isset($_GET["section"])) ? $_GET["section"] : '';
$tid = (isset($_GET["tid"])) ? $_GET["tid"] : '';

switch ($display)
{
    case "section":
        //Massively cleaned up section which obtains section titles
        from the language file
        foreach($l_sectionlist as $key=>$value){
            if($section==$key){
                $currentsection = $key;
                $sectiontitle = $value;
            }
        }
        $result = $mysqli->query("SELECT * FROM items WHERE
        section='$currentsection' ORDER BY $sortby");
        echo "<div id='sectiontitle'><h1>$sectiontitle</h1></div>";
        $noresultsurl = '?section=' . $section;
    break;
    case "project":
    case "context":
        $idresult = $mysqli->query("SELECT title FROM {$display}s
        WHERE id='$tid'");
        $disptitle = $idresult->fetch_row()[0];
        $result = $mysqli->query("SELECT * FROM items WHERE
        $display='$disptitle' ORDER BY $sortby");
        echo "<div id='sectiontitle'><h1>$disptitle</h1></div>";
        $noresultsurl = '?tid=' . $tid;
    break;
    case "all":
        $result = $mysqli->query("SELECT * FROM items ORDER BY
        $sortby");
        echo "<div id='sectiontitle'><h1>".$l_nav_allitems."</h1></
        div>";
        $noresultsurl = '';
    break;
    case "today":
        $today = date("Y-m-d");
        $todayf = date($menu_date_format);
        $result = $mysqli->query("SELECT * FROM items WHERE
        date='$today' ORDER BY $sortby");
        echo "<div id='sectiontitle'><h1>".$l_nav_today.": $todayf</
        h1></div>";
        $noresultsurl = '';
    break;
```

On peut donc effectuer n'importe quelle injection SQL si on le souhaite à partir de l'URL du site :



MySQL protège par défaut des requêtes multiples dans un même élément mais il reste possible d'exploiter cette faille d'une façon ou d'une autre afin d'attaquer la base de données du site.

Chiffrement des données stockées

Données	Sensibilité	Chiffrement utilisé	Accès à la donnée	Niveau d'adéquation
Tâches	Peu sensible	Chiffré par MySQL	Sur les pages listant les tâches	Bon car nous n'avons pas besoin de plus de chiffrement de données que celui de la BDD de base
Items	Peu sensible	Chiffré par MySQL	Sur les pages listant les items	Bon car nous n'avons pas besoin de plus de chiffrement de données que celui de la BDD de base
Login	Peu sensible	Chiffré par MySQL	Par la base de données et lors de la connexion	Bon car nous n'avons pas besoin de plus de chiffrement de données que celui de la BDD de base
Mot de passe	Très sensible	MD5 + salage	/	Non car MD5 plus sécurisé
Sel	Très sensible	Chiffré par MySQL	/	Mauvais car pourrait être accessible sous injection SQL, peut être stocké à part de la BDD

Recommandations

Toutes les requêtes du programme seront à sécuriser si on souhaite éviter les injections SQL, cela n'est actuellement pas le cas et il est possible de trouver des informations sensibles de la base de données à l'aide de la bonne requête. En mettant le code à jour, il serait donc préférable d'échapper les paramètres et de les vérifier avant d'exécuter des requêtes SQL.

Toutes fois, les failles SQL via le mot de passe sont empêchées lors de la connexion.

Le sel, servant à renforcer la sécurité derrière l'authentification, se trouve également en BDD, ce qui implique qu'une fuite de la base de données nous donnerait accès, non seulement au mot de passe haché, mais aussi à son sel et n'empêchera donc pas l'authentification. Il devrait donc se trouver dans un fichier séparé ou purement en code afin d'éviter une telle faille de sécurité.

De même, la fonction de hachage MD5 est aujourd'hui sujette à de majeures vulnérabilités et continuer de l'utiliser est un risque pour les utilisateurs de cette application. Il serait donc recommandable de modifier l'algorithme de hachage pour un plus robuste, le SHA256 étant plus actuel et optimisé dans notre cas.

Optimisation

Concernant la sécurité des mots de passe il a été choisi de les chiffrer en base de données et d'utiliser l'algorithme de chiffrement bcrypt qui est implanté de manière native et plus lent que SHA256, ce qui permet de nuire aux attaques par brute force en prolongeant le temps de calcul.

En plus d'avoir utilisé cet algorithme de chiffrement d'utiliser un salage généré de manière aléatoire pour garantir une plus grande sécurité. En effet le même mot de passe chiffré grâce à la méthode de hashage ne redonnera théoriquement jamais la même empreinte.

Pour parvenir à cela nous avons utilisé la méthode password_hash de PHP.

```
1 reference | 0 overrides
public function Register(string $login,string $password1,string $password2): void
{
    if ($password1 != $password2)
        throw new Exception(message: "The two passwords aren't the same.");
    if (!$this->userDAO->IsLoginAvaible(login: $login))
        throw new Exception(message: "This login is not avaible.");
    if (!$this->isValidPassword(password: $password1))
        throw new Exception(message: "The password should have a length equal to 12 and must have numbers with specials caracters.");
    $user = new User();
    $user->setLogin(login: $login);
    $user->setPassword(hashedPassword: password_hash(password: $password1,algo: PASSWORD_DEFAULT));
    $this->userDAO->CreateUser(user: $user);
}
```

Le problème ayant été aussi remonté durant la phase d'audit concernant les données stockées en base de données était les failles d'injections SQL. Ce problème a été pu être endigué grâce à l'utilisation de requête préparée ce qui veut dire que la requête de base est écrite de dur mais les paramètres sont rajoutés après par l'utilisation de chaine de caractères ce qui empêche au SGBD d'exécuter la deuxième partie qui peut donc nuire à l'intégrité et à la sécurité des données.

```
1 reference | 0 overrides
public function DeleteUser(int $identifiant): void
{
    $request = "DELETE FROM USERS WHERE id = :id;";
    $params = [
        ":id" => $identifiant
    ];
    $this->execRequest(sql: $request, params: $params);
}

3 references | 0 overrides
public function GetUser(string $login) : ?User
{
    $user = null;
    $request = "SELECT id,password,login FROM USERS WHERE login = :login";
    $params = [
        ":login" => $login
    ];
    $response = $this->execRequest(sql: $request,params: $params);
    if (($row = $response->fetch(mode: PDO::FETCH_ASSOC)) != null)
    {
        $user = new User();
        $user->hydrate(data: $row);
    }
    return $user;
}
```

V. Résultats finaux

Le tableau suivant liste les tâches d'optimisation qui étaient définies par l'audit de gestion de données, la dernière colonne indiquera en vert si la tâche a pu être réalisée, en orange si elle ne l'est que partiellement, en rouge si elle n'a pas pu l'être

Recommandation	Priorité	Complexité	Effort	Gravité	Avancement
Passage Bcrypt	6	1	1	3	
Passage https	6	5	4	3	
Limite tentatives mot de passe	6	4	3	3	
Limite de temps connecté	5	3	3	2	
Gérer injection SQL	6	2	2	3	

VI. Synthèse

L'objectif du projet était d'identifier les vulnérabilités et failles de sécurité du site web, chose qui avait été faite durant l'audit de sécurité durant lequel nous avons pu déterminer que non seulement des injections SQL étaient possibles, mais aussi que le site pouvait aisément être brut force, en plus de ne pas avoir de mots de passes sécurisés (aucune contrainte imposée par exemple).

Ainsi, nous avons non seulement mis à jour la méthode de hachage des mots de passes, celui de Taskstep étant originellement dépassé, tout en imposant un caractère majuscule, minuscule, un caractère, un chiffre ainsi qu'une taille minimum de 12 caractères.

De même, les injections SQL sont maintenant nativement gérées en PHP et ont pu être gérées durant la restructuration de ce projet, évitant donc des failles de sécurité du côté de la base de données.

Enfin, nous avons sécurisé le système de connexion à travers une variable de session limitant le nombre de tentatives pour un temps donné (après 5 tentatives échouées, on doit attendre 10 minutes par exemple) mais aussi en limitant le temps de connexion d'un utilisateur.

La seule optimisation que nous n'avons pas pu réaliser était celle du passage au HTTPS, étant donné que cela nécessite un certificat et que nous n'en avons pas les moyens. Ceci constitue une faille et dépend donc de la personne hébergeant le site web et non de nous.

Finalement, nous avons donc pu optimiser la quasi-totalité des failles de sécurité du site web, beaucoup peuvent nécessiter de bien meilleures fonctionnalités (limite par session pouvant être évitée par suppression du cookie de session par exemple) mais pour un site web ne servant

qu’à lister des tâches de façon locale ou interne à une entreprise, nous estimons avoir apporté des ajustements suffisants à la sécurité du site web.