Rapport Challenge

Extension PHP
- Can You See Me ? -

•••

développement PHP bas-niveau
implémentation dans la Zend Engine

Réalisé par

Hamza Lazaar

Encadré par

Abdelmajid Bendrif
Hamza Bahlaouane

PFE 2026 – 22 février

Table des matières

Introduction 3

1 Investigation et Reverse Engineering 4
1.1 Localisation des Binaires et de la Configuration 4
1.2 Preuve de Concept via la Reflection API 5

2 Développement d'une Extension C Native 6
2.1 Préparation de l'Environnement et Génération 6
2.2 Implémentation C et Compatibilité PHP 8 7
2.3 Compilation (Toolchain C) et Tests 8

Chapitre 0 : Introduction

Contrairement aux bibliothèques PHP standards (packages Composer) qui sont interprétées à l'exécution, les extensions PHP sont des modules binaires compilés, développés en langage C ou C++. Elles s'interfacent directement avec le moteur interne de PHP (**Zend Engine**), offrant ainsi des performances d'exécution natives et l'accès à des fonctionnalités système de bas niveau.

Ce document consigne les résultats d'un environnement de laboratoire (Docker : `hbahalouane/void-labs-php-ext`). L'exercice se divise en deux phases : le reverse engineering d'une extension C existante et le développement complet (de la conception à la compilation) d'une extension native personnalisée.

Chapitre 1 : Investigation et Reverse Engineering

L'environnement de départ présente un script PHP (`app.php`) exécutant une méthode `\VoidLabs\canYouSeeMe()` sans aucune inclusion de fichier source (`require` ou autoloader). L'objectif était d'identifier l'origine de cette méthode injectée dynamiquement.

1.1. Localisation des Binaires et de la Configuration

L'investigation système a été menée via les outils d'interface en ligne de commande (CLI) de PHP :

```
# 1. Identification des fichiers d'initialisation (.ini)
php --ini
# Résultat : Chargement de /usr/local/etc/php/conf.d/docker-php-ext-
    sample.ini

# 2. Vérification des modules compilés
php -m | grep sample
# Résultat : Le module 'sample' est actif dans le Zend Engine

# 3. Localisation physique du répertoire des extensions
php -i | grep extension_dir
# Résultat : /usr/local/lib/php/extensions/no-debug-non-zts-20220829
```

Listing 1.1 – Commandes d'investigation système

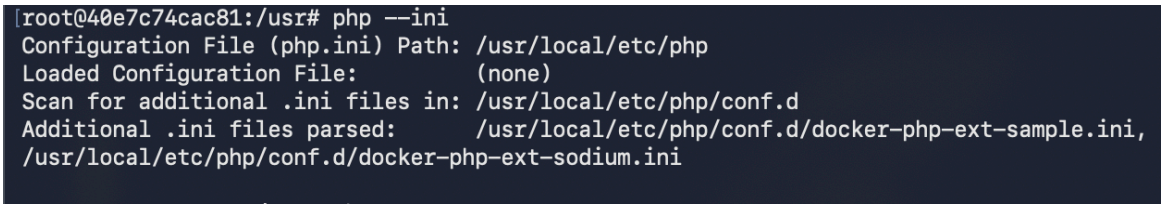

FIGURE 1.1 – commande 1 : > php –ini

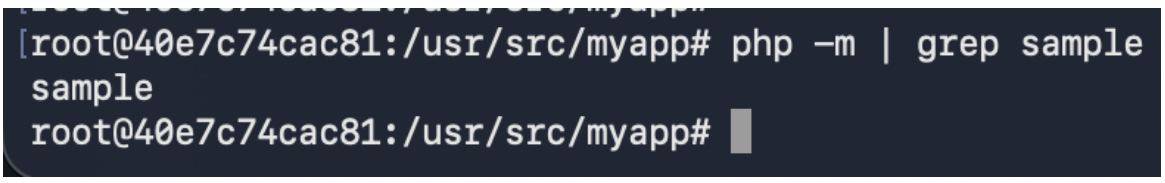

FIGURE 1.2 – commande 2 : > php -m | grep sample

FIGURE 1.3 – commande 3 : > php -i | grep extension_dir

La navigation dans ce répertoire a permis d'isoler le fichier `sample.so` (*Shared Object*), un binaire C précompilé.

1.2. Preuve de Concept via la Reflection API

Pour lier de manière irréfutable le namespace PHP `\VoidLabs`, en particulier la fonction canYouSeeMe() au binaire `sample.so`, la classe interne `ReflectionFunction` a été exploitée. Cette API permet d'introspecter le comportement du moteur PHP à l'exécution.

```
1 php -r "echo (new ReflectionFunction('\VoidLabs\canYouSeeMe'))->
    getExtensionName();"
2 // Sortie standard : sample
```

Listing 1.2 – Introspection de la fonction cible

FIGURE 1.4 – ReflectionFunction pour detecter le lien entre sample.so et la fonction canYouSeeMe()

Cette commande prouve techniquement que la fonction n'est pas issue d'un script PHP standard, mais qu'elle a été allouée en mémoire lors de la phase `MINIT` (Module Initialization) de l'extension `sample`.

Chapitre 2 : Développement d'une Extension C Native

La seconde phase du laboratoire consistait à développer une extension native nommée `my_ext`, capable d'exposer une fonction personnalisée au Zend Engine.

2.1. Préparation de l'Environnement et Génération

Le développement requiert les bibliothèques C standard (`gcc`, `make`). Dans le contexte d'une image Docker PHP officielle, le code source a d'abord dû être extrait pour accéder à l'outil de création d'extensions `ext_skel.php`.

```
# Extraction du code source de PHP
docker-php-source extract

# Génération de l'arborescence de l'extension C
cd /usr/src/php/ext
php ext_skel.php --ext my_ext
```

Listing 2.1 – Extraction et génération du squelette

```
root@40e7c74cac81:/usr/src/myapp# docker-php-source extract
root@40e7c74cac81:/usr/src/myapp# cd /usr/src/php/ext
root@40e7c74cac81:/usr/src/php/ext# php ext_skel.php --ext my_ext
Copying config scripts... done
Copying sources... done
Copying tests... done

Success. The extension is now ready to be compiled. To do so, use the
following steps:

cd /usr/src/php/ext/my_ext
[phpize
./configure
[make
[
Don't forget to run tests once the compilation is done:
make test

Thank you for using PHP!
```

FIGURE 2.1 – Output de l'extraction et génération du squelette

2.2. Implémentation C et Compatibilité PHP 8

La logique métier a été rédigée dans le fichier `my_ext.c`. L'utilisation de la macro `PHP_FUNCTION` permet d'interfacer le type de retour C avec les structures internes de PHP (`zval`).

```c
PHP_FUNCTION(my_hello_hamza)
{
    RETURN_STRING("Hello Hamza et Abdelmajid, its hamza :)");
}
```

Listing 2.2 – Logique métier en C (my_ext.c)

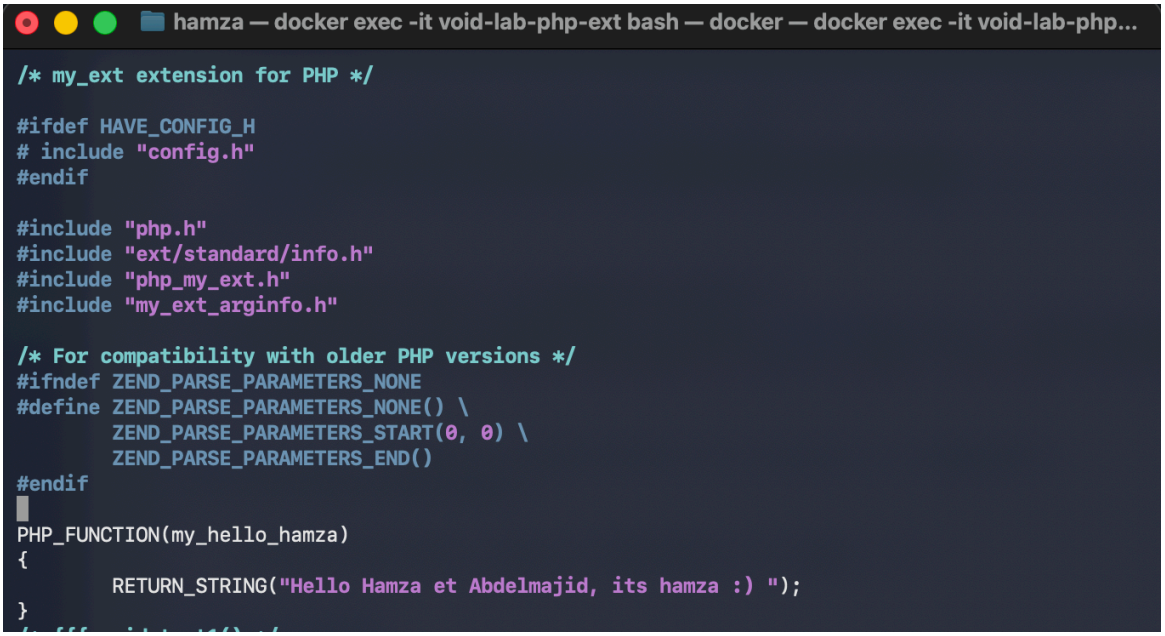

FIGURE 2.2 – Ma fonction my_hello_hamza() dans my_ext.c

Un défi architectural propre à PHP 8+ a été rencontré lors de la compilation (*redefinition of 'ext_functions'*). Dans les versions récentes du Zend Engine, la déclaration de la fonction (`arginfo`) doit être strictement dissociée du code source principal et placée dans le fichier d'en-tête `my_ext_arginfo.h`.

```c
ZEND_FUNCTION(my_hello_hamza);

static const zend_function_entry ext_functions[] = {
    ZEND_FE(my_hello_hamza, NULL)
    ZEND_FE_END
};
```

Listing 2.3 – Déclaration stricte de la fonction (my_ext_arginfo.h)

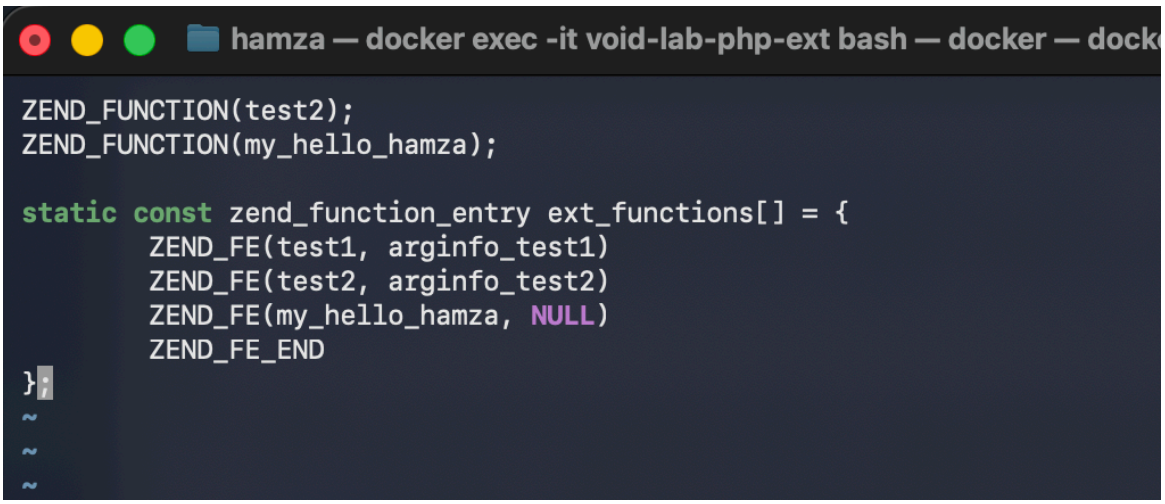

FIGURE 2.3 – Déclaration de la fonction my_hello_hamza()

2.3. Compilation (Toolchain C) et Tests

Le code source a été transformé en module dynamique binaire via le flux standard de compilation C, orchestré par `phpize`.

```
# Préparation de l'environnement de compilation
phpize
./configure

# Compilation et liaison (création du .so)
make
make install

# Activation dans la configuration PHP
docker-php-ext-enable my_ext

# Exécution finale et validation
php -r "echo my_hello_hamza() . \"\n\";"
```

Listing 2.4 – Processus de compilation et test

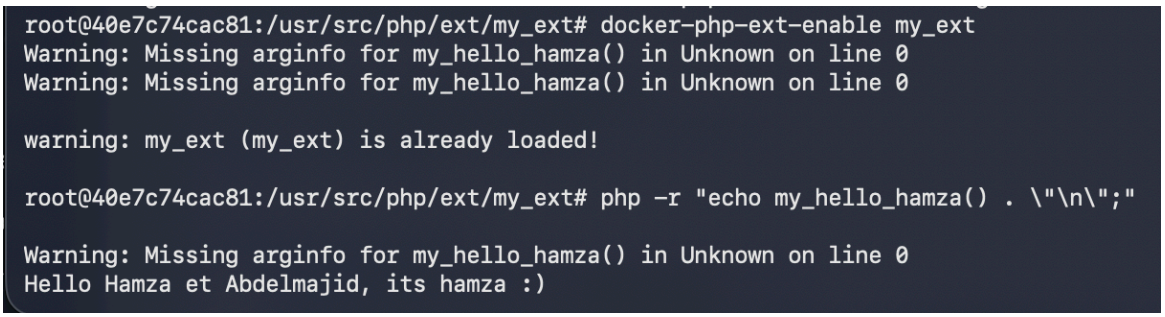

FIGURE 2.4 – Execution de ma fonction avec php -r

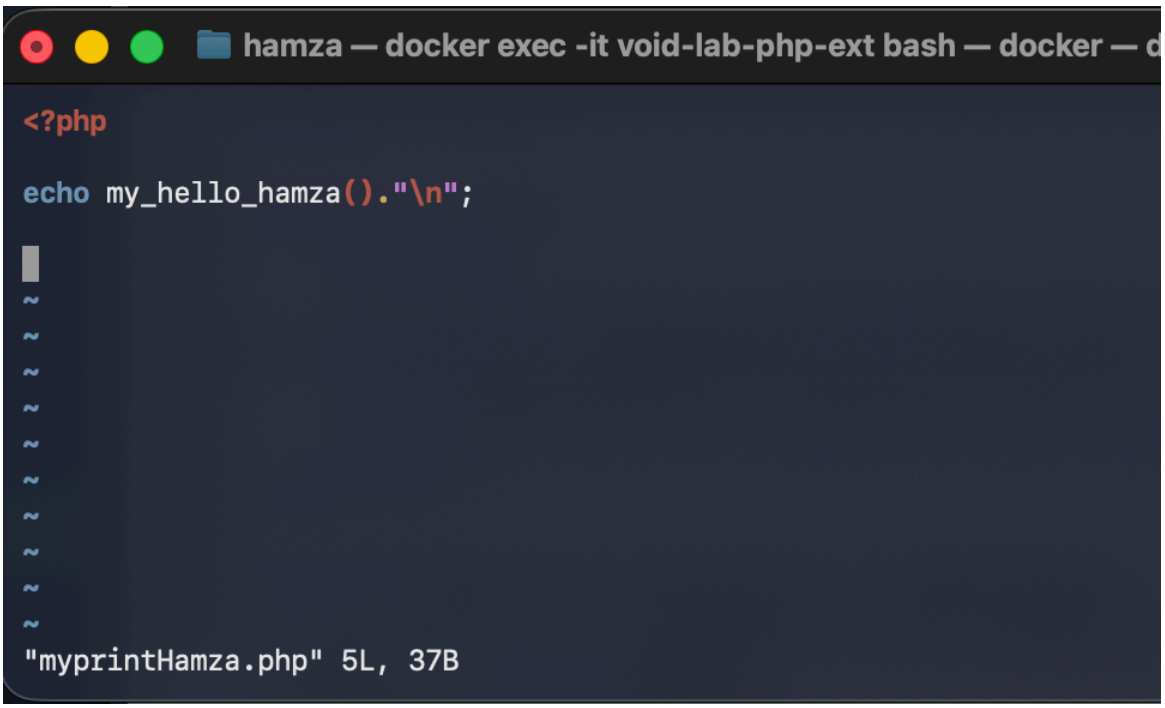

FIGURE 2.5 – Création d'un fichier php qui execute la fonction

FIGURE 2.6 – Execution du script PHP contenant ma fonction

Conclusion du Lab :
L'ingénierie d'une extension PHP requiert une compréhension bas niveau de la gestion de la mémoire et de l'architecture du Zend Engine. Ce laboratoire valide la capacité à analyser des environnements conteneurisés complexes et à étendre nativement les capacités du langage via des processus de compilation C.