Can You See Me
Exploration of an Unknown PHP Extension

Hatim Salhi

February 22, 2026

Contents

1 Introduction

The objective of this laboratory exercise is to explore the PHP ecosystem and identify an unknown PHP extension that adds a new function to the runtime environment.

A Docker image was provided containing an unknown extension exposing the following function:

```
1 \VoidLabs\canYouSeeMe();
```

The mission is to determine which extension provides this function and to explain the technical steps used to discover it.

2 Environment Setup

The container was started using the following command:

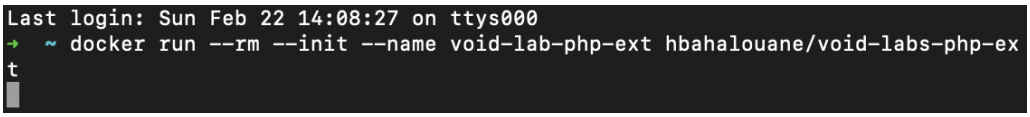

Figure 1: Starting the Docker container

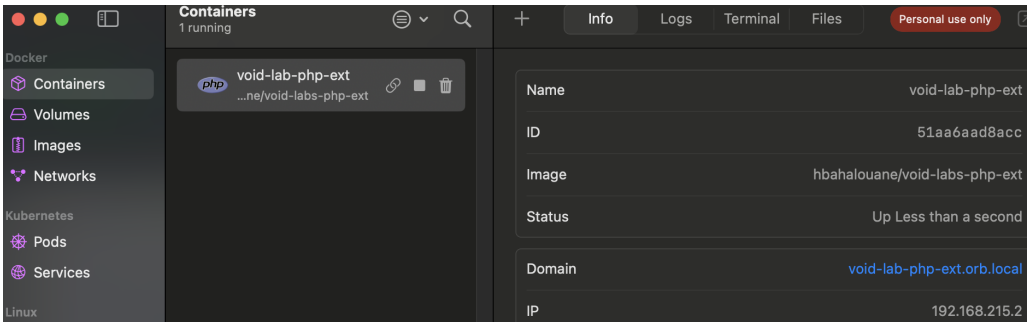

Figure 2: Container running inside OrbStack

Then, an interactive shell was opened and the script was executed:

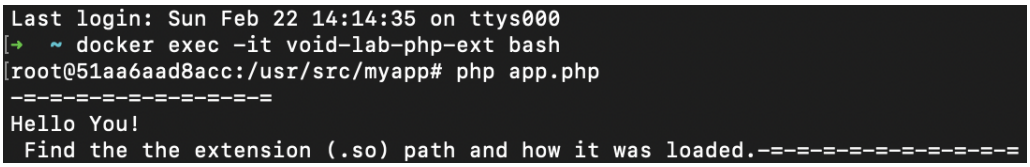

Figure 3: Execution of app.php inside the container

3 Checking if the Function is Internal

To verify whether the function comes from an extension, PHP Reflection was used.

```
root@51aa6aad8acc:/usr/src/myapp# php -r '
$f="VoidLabs\\canYouSeeMe";
var_dump(function_exists($f));
$r=new ReflectionFunction($f);
echo "internal=" . ($r->isInternal() ? "yes" : "no") . PHP_EOL;
echo "extension=" . ($r->getExtensionName() ?: "none") . PHP_EOL;
echo "file=" . var_export($r->getFileName(), true) . PHP_EOL;
'
```

Figure 4: Command used to check if the function is internal

```
bool(true)
internal=yes
extension=sample
file=false
```

Figure 5: Result of the reflection command

The output indicates that the function is internal and belongs to an extension named **sample**.

4 Listing Loaded PHP Modules

All loaded PHP modules were listed using:

```
1 php -m
```

```
PDO
pdo_sqlite
Phar
posix
random
readline
Reflection
sample
session
SimpleXML
sodium
```

Figure 6: List of loaded PHP modules

The module **sample** appears in the list.

5 Locating the Extension File

The extension directory was identified using:

```
root@51aa6aad8acc:/usr/src/myapp# php -i | grep -i '^extension_dir'
extension_dir => /usr/local/lib/php/extensions/no-debug-non-zts-20220829 => /usr
/local/lib/php/extensions/no-debug-non-zts-20220829
```

Figure 7: Identifying the PHP extension directory

Then its content was listed:

```
root@51aa6aad8acc:/usr/src/myapp# ls -la /usr/local/lib/php/extensions/no-debug-
non-zts-20220829 | grep sample
-rwxr-xr-x 1 root root   31824 Feb 19  2025 sample.so
```

Figure 8: Presence of sample.so inside the extension directory

This confirms the existence of `sample.so`.

6 Finding How the Extension is Loaded

PHP configuration files were displayed using:

```
root@51aa6aad8acc:/usr/src/myapp# php --ini
Configuration File (php.ini) Path: /usr/local/etc/php
Loaded Configuration File:         (none)
Scan for additional .ini files in: /usr/local/etc/php/conf.d
Additional .ini files parsed:      /usr/local/etc/php/conf.d/docker-php-ext-samp
le.ini,
/usr/local/etc/php/conf.d/docker-php-ext-sodium.ini
```

Figure 9: Displaying PHP configuration files (1)

Then:

```
root@51aa6aad8acc:/usr/src/myapp# ls -la /usr/local/etc/php/conf.d/
total 8
drwxr-xr-x 1 root root 50 Feb 19  2025 .
drwxr-xr-x 1 root root 12 Feb  4  2025 ..
-rw-r--r-- 1 root root 17 Feb 19  2025 docker-php-ext-sample.ini
-rw-r--r-- 1 root root 17 Feb  4  2025 docker-php-ext-sodium.ini
```

Figure 10: Displaying PHP configuration files (2)

```
root@51aa6aad8acc:/usr/src/myapp# cat /usr/local/etc/php/conf.d/docker-php-ext-s
ample.ini
extension=sample
```

Figure 11: Configuration file loading the sample extension

The file contains:

```
1 extension=sample
```

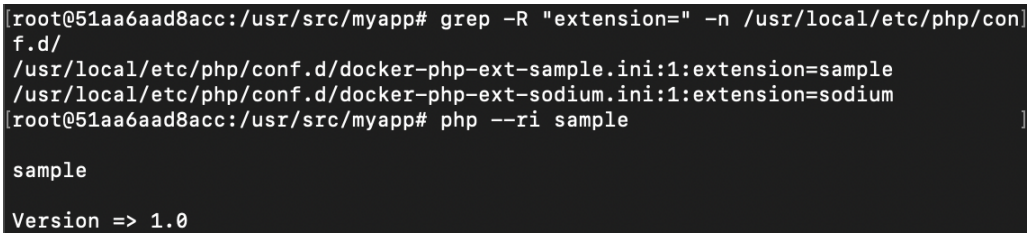

Figure 12: Extension loading verification

- The `grep` command shows which extensions PHP is instructed to load.
- The `php -ri` command displays information about a loaded extension.

7 Verifying the Function Inside the Binary

To confirm that the function is compiled inside the extension:

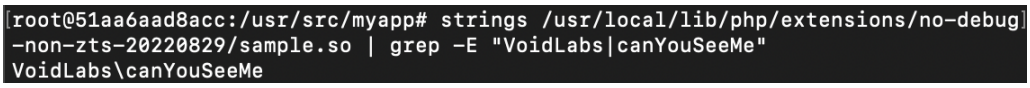

Figure 13: Strings output showing VoidLabs\canYouSeeMe

This proves that the function is embedded in the binary file.

8 Results

The function `\VoidLabs\canYouSeeMe()` is provided by the following PHP extension:

- Name: sample
- Binary file: sample.so
- Loaded via: docker-php-ext-sample.ini

9 Bonus: Creating a Similar PHP Extension

To create a similar extension:

1. Generate an extension skeleton using `phpize`
2. Implement the function in C using the Zend API
3. Compile the extension using `make`
4. Install and enable the extension

10 Bonus: Why Use PHP Extensions

PHP extensions allow:

- Faster execution using native C code
- Access to system-level libraries
- Implementation of features not possible in pure PHP

11 Conclusion

Through systematic inspection using reflection, module listing, configuration analysis, and binary inspection, the unknown function was successfully traced back to the **sample** PHP extension.