Bezpieczny framework PHP

Łukasz Pilorz

Allegro.pl

OWASP

Plan prezentacji

Część teoretyczna

- **Wstęp – frameworki PHP dla aplikacji webowych**
- **Cechy bezpiecznego frameworka**

Część praktyczna

- **Błędy, których chcemy uniknąć**
- **Przykłady podatnego kodu**
- **Demonstracja ataków**

Framework

pl.wikipedia.org:

- *Struktura wspomagająca tworzenie, rozwój i testowanie powstającej aplikacji*
- *Szkielet działania aplikacji, który zapewnia podstawowe mechanizmy i może być wypełniany właściwą treścią programu*

Szybciej, wygodniej i bezpieczniej?

PHP

- Zend Framework
- Symfony
- CakePHP
- Kohana
- CodeIgniter

i wiele innych

MVC: Model-View-Controller

Framework i bezpieczeństwo

Wykorzystanie frameworka wpływa najczęściej pozytywnie na bezpieczeństwo aplikacji:

- gotowe, sprawdzone i poprawne rozwiązania
- jednolity standard programowania

ale również

- podatności we frameworkach i pluginach
- bezpieczeństwo jako opcja, nie stan domyślny

Cechy bezpiecznego frameworka

1. **Dokumentacja**

- szczegółowa, przejrzysta, prawdziwa
- zalecenia nt. bezpiecznego programowania
- słowa kluczowe:

  injection, traversal, XSS, CSRF, authentication, authorization, access, validation, escaping

W praktyce – czasem lepiej przygotowana, niż dokumentacja PHP...

... która zaleca niebezpieczne rozwiązania:

```
<?php
$e = escapeshellcmd($userinput);
// here we don't care if $e has spaces
system("echo $e");

$f = escapeshellcmd($filename);
// and here we do, so we use quotes
system("touch \"/tmp/$f\"; ls -l \"/tmp/$f\"");
?>
```

Źródło: http://pl.php.net/escapeshellcmd

Skutek:

http://pear.php.net/bugs/bug.php?id=16200

„Mail package uses escapeshellcmd function incorrectly, so it may allow to read/write arbitrary file”

Błąd można wykorzystać w przypadku, gdy adres e-mail pochodzący od użytkownika trafia do nagłówka „From”.

Poprawka w wersji 1.2.0 (2010-03-01)

- zastosowanie funkcji escapeshell**arg**

Cechy bezpiecznego frameworka

2. Mechanizmy **uwierzytelnienia i autoryzacji**

- poprawność :-)
- elastyczność i uniwersalność
  - opcja „default deny” w konfiguracji
  - logowanie, wylogowanie, zmiana/reset hasła, rejestracja, role i uprawnienia, zmiana uprawnień w czasie trwania sesji
- bezpieczeństwo sesji
  - zapewnione niezależnie od ustawień php.ini
  - regeneracja id sesji przy podniesieniu uprawnień
  - zabezpieczenia przed przechwyceniem (np. httpOnly, secure)
  - ochrona formularzy przed CSRF
  - wygasanie ważności sesji (po stronie serwera)

Test na spostrzegawczość

```
<?php
session_start();
if(
   !isset($_SESSION['admin'])
   || $_SESSION['admin']!==true
) {
   $goto = '/login.php';
   if(isset($_SERVER['HTTP_HOST'])) {
      $goto = 'http://'.$_SERVER['HTTP_HOST'].'/login.php';
   }
   header('Location: '.$goto);
}
echo 'Tutaj panel administracyjny';
?>
```

Cechy bezpiecznego frameworka

3. **Walidacja** danych wejściowych

- mechanizmy umożliwiające (lub wymuszające) weryfikację typu, długości, kodowania znaków, dozwolonego zakresu i formatu danych

- walidacja na podstawie założeń logiki biznesowej

- „automagiczne” rozwiązania na tym etapie nie mogą zastąpić zabezpieczeń w kolejnych warstwach aplikacji

- dispatcher: whitelisting

Typowe problemy:

- bajt zerowy (np. ereg, wzorce preg_*)
- dane w zakodowanej postaci (serializacja, base64, XML itp.)
- zaufanie do tablicy $_SERVER (nagłówków HTTP)

Przykłady

```php
<?php
$input = 'Hello world!';
if(isset($_GET['s']) && is_string($_GET['s'])) {
    $input = htmlspecialchars($_GET['s'], ENT_QUOTES, 'UTF-8');
}
//...
$output = iconv('UTF-8', 'ISO-8859-2//TRANSLIT', $input);
header('Content-Type: text/html; charset=iso-8859-2');
echo $output;
?>
```

```php
<?php
$array_allowed_ids = array(1, 2, 3);
$int_id = 1;
if(isset($_GET['id']) && in_array($_GET['id'], $array_allowed_ids)) {
       $int_id = $_GET['id'];
}
readfile('static/static_'.$int_id.'.html');
?>
```

Przykłady

```php
<?php
setlocale(LC_MONETARY, 'pl_PL');
$c = 0; //ilosc sztuk
if(isset($_POST['c'])) {
    $c = intval($_POST['c']);
}
$price = 137057; //cena w groszach
$total = intval(137057 * $c); //kwota do zaplaty
$total = $total/100; //zamiana na zlote
echo money_format('%i', $total);
?>
```

Dispatcher

```php
<?php
@include('load_controllers.php');
$path = explode('/', $_SERVER['PHP_SELF']);
if(sizeof($path>3)) {
    $class = 'Default';
    if(preg_match('#\A\w+\z#', $class)) {
        $class = $path[2];
    }
    $action = $path[3];
    $param1 = isset($path[4])?$path[4]:null;
    $param2 = isset($path[5])?$path[5]:null;
    $controller = new $class($param1, $param2);
    $controller->$action();
}
?>
```

Cechy bezpiecznego frameworka

4. Formatowanie danych wyjściowych (**escape'owanie**)

- zapytania SQL
- HTML, XML, JavaScript, CSS itp.
- wszystkie formaty i protokoły stosowane przez aplikację do wymiany danych
- polecenia i ścieżki systemowe

Narzucenie programiście metod opakowujących dostęp do poszczególnych formatów i protokołów, automatycznie wymuszających bezpieczeństwo.

Escape'owanie

Jeśli jakiekolwiek dane wejściowe mogą doprowadzić do wyniku wyjściowego nie będącego poprawnym wyrażeniem docelowego formatu lub protokołu, należy założyć, że metody formatujące/escape'ujące są błędne.

Błąd !== podatność

Eliminujemy błędy.

Cechy bezpiecznego frameworka

5. **Kryptografia**

Funkcje realizujące typowe scenariusze zastosowania kryptografii w aplikacjach webowych:

- generowanie unikalnych tokenów/kodów (losowych, powiązanych z użytkownikiem, powiązanych z sesją, jedno- lub wielokrotnego użytku)
- przezroczyste szyfrowanie zawartości bazy danych
- solone haszowanie danych uwierzytelniających
- szyfrowanie i dodawanie sum kontrolnych do parametrów URL, ciasteczek, ukrytych pól formularzy i innych danych przesyłanych za pośrednictwem niezaufanych kanałów (pliki, aplikacje zewnętrzne, przeglądarki użytkowników)

Często w charakterze zabezpieczenia kryptograficznego (szyfrowania) błędnie stosowane są funkcje kodujące lub ich kombinacje:

- base64_encode
- serialize
- urlencode
- bzcompress
- bin2hex
- XOR ze stałą wartością

Rola **bezpiecznego frameworka** polega na podsuwaniu gotowych funkcji i przykładów w taki sposób, **aby skorzystanie z poprawnego rozwiązania było łatwiejsze**, niż stworzenie własnego.

Niebezpieczna serializacja

Najciekawsza podatność 2009 roku:

Zastosowanie PHP-IDS <= 0.6.2 w aplikacjach opartych o Zend Framework umożliwia zdalne wykonanie kodu PHP.

Przyczyną jest zastosowanie funkcji unserialize() do niezaufanych danych pochodzących od użytkownika – problem wykryty i opisany przez Stefana Essera http://www.suspekt.org

DEMO

Więcej informacji o bezpieczeństwie frameworków:

- OWASP ISWG (Intrinsic Security Working Group)
- Secure Web Application Framework Manifesto
- Changelogi
  - Symfony – SQL Injection – luty 2010
  - Zend Framework – XSS – styczeń 2010
  - CakePHP – błąd zabezpieczeń przed CSRF – styczeń 2010

Ty też możesz uratować Web:

- Testy i analizy bezpieczeństwa frameworków
- Porównanie zastosowanych rozwiązań i zabezpieczeń
- Zgłaszanie błędów w kodzie i dokumentacji
- Rozwój OWASP ESAPI dla PHP

Dziękuję