

Tomasz "Zyx" Jędrzejewski

System uprawnień w PHP

Wersja 1.0 (23.05.2007)



Szczegółowe informacje o licencji znajdują się pod artykułem.

www.zyxist.com

System uprawnień w PHP

W obecnych czasach każda rozbudowana witryna internetowa potrzebuje odpowiednich mechanizmów kontroli dostępu i zarządzania uprawnieniami. Różne modele mają swoje wady i zalety, a programista musi zdecydować, który z nich wykorzystać do własnych potrzeb. Niniejszy artykuł przedstawia sposób stworzenia mechanizmu pobierania i kontroli uprawnień w języku PHP z wykorzystaniem programowania obiektowego.

Zasada działania

Na początek podkreślmy, że system uprawnień to co innego, niż sesja użytkownika. Zadaniem sesji jest powiązanie wizyt tego samego internauty tak, aby serwer mógł rozpoznać, w jaki sposób wędruje on po stronie i przesłać pewne informacje między żądaniami. Jedyna rzecz, jaka może mieć tam związek z uprawnieniami, to pobieranie podstawowych danych zalogowanego użytkownika, jednak to ostatecznie osobny mechanizm powinien decydować, czy dane te wystarczają do wpuszczenia go do wskazanego elementu serwisu.

Podczas inicjacji, system uprawnień powinien pobrać z sesji informacje o tym, kto jest zalogowany, a następnie załadować z bazy danych informacje o zasobach, do których ma dostęp. Ich pobieranie może być pasywne lub aktywne. Pasywne cechuje się tym, że ładujemy wszystkie uprawnienia danego użytkownika do pamięci przed rozpoczęciem wykonywania właściwego kodu danej akcji lub podstrony, dzięki czemu ewentualne życzenie sprawdzenia, czy użytkownik ma dostęp do elementu na drugim końcu serwisu nie stanowi problemu. Jednak przy większych stronach ilość możliwych do skonfigurowania uprawnień może iść w setki lub nawet tysiące i w takim wypadku ciężko oczekiwać, że baza z radością będzie nam te dane błyskawicznie udostępniała. Ponadto przeważnie nie ma sensu ładowanie tak wielkich ilości danych, z których i tak nie będzie żadnego pożytku. System aktywny wymaga nieco większego wysiłku, jeśli chodzi o programowanie. Kluczem jest tu stworzenie pewnego systemu cache z informacjami o tym, jakie uprawnienia są sprawdzane na każdej podstronie i korzystanie z niego do zawężenia liczby pobieranych informacji. Taki system może działać w ten sposób, że jeśli w trakcie inicjowania system autoryzacji nie znajdzie potrzebnego pliku cache dla danej podstrony, ładuje wszystkie możliwe uprawnienia danego użytkownika, jak w systemie pasywnym, jednak dodatkowo w trakcie działania serwisu zbiera informacje o tym, do których uprawnień odwoływał się kod danej akcji. Pod koniec wykonywania w pamięci skryptu przechowujemy listę takich uprawnień, którą wystarczy tylko zrzucić do pliku, aby kolejne żądania wyświetlenia danej strony łądowały jedynie tę część spisu, która jest potrzebna.

Do większych projektów przydaje się również możliwość rozbudowywania systemu o dodatkowe źródła uprawnień. Przykładowo, nasz serwis domyślnie przechowuje w bazie danych listę uprawnień „Zezwól/Odmów” do poszczególnych części panelu administracyjnego, lecz część odpowiedzialna za CMS posiada własne listy uprawnień dla każdego widocznego obiektu w formacie „Dodaj/Edytuj/Usuń/Wyślij konkurencji” itd. Rzecz jasna, kiedy pracujemy w panelu do zarządzania CMS'em, system musi dodatkowo łądować uprawnienia zapisane w tym drugim formacie.

Budowa

Przykładowy system, który pragnę zaprezentować w tym artykule, napisałem, wzorując się w niektórych aspektach na systemie ACL dostępnym w Zend Framework. W systemie tym rozróżniamy użytkownika, który ma przypisaną konkretną rolę. Rola definiuje zestaw dostępnych uprawnień. Przykładowo, użytkownik-administrator będzie mieć dostęp do wszystkiego, natomiast użytkownik-redaktor tylko do stron umożliwiających edycję danych widocznych w serwisie. W ACL-u zdefiniowane są też tzw. zasoby, czyli poszczególne uprawnienia, jakie można ustawiać rolom.

Co ważniejsze, lista zasobów nie jest liniowa, lecz tworzy hierarchiczne drzewko, dzięki czemu zwiększa się bezpieczeństwo. W bazie danych nie jest konieczne przechowywanie drzewka, ponieważ jego konstrukcja w tamtym miejscu jest bardzo wymagająca. Zamiast tego, informacje o wzajemnych zależnościach będziemy zapisywać w formie tekstowej przypominającej trochę nazwy katalogów: zapis `www/admin/news/edit` oznacza uprawnienie „edycji” przypisane do uprawnienia „news” znajdującego się w „adminie”, który z kolei leży w obrębie całej strony „www”. Aby użytkownik mógł edytować newsy, musi mieć dostęp do wszystkich czterech zasobów zdefiniowanych w tej ścieżce. Dzięki temu, jeśli administrator

przypadkowo zdefiniowałby dostęp do zasobu „news” zwykłemu użytkownikowi, i tak nasz ACL nie wpuści go do niego, ponieważ nie będzie mieć uprawnień do nadrzędnej grupy „admin”.

Baza danych

Podstawowa struktura bazy danych dla naszego systemu ACL składa się z trzech tabel:

```
CREATE TABLE `acl_resource` (
  `id` int(11) NOT NULL auto_increment,
  `title` varchar(40) collate utf8_polish_ci NOT NULL,
  `hash` varchar(40) collate utf8_polish_ci NOT NULL,
  `default_value` tinyint(4) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;

CREATE TABLE `acl_role` (
  `id` int(11) NOT NULL auto_increment,
  `title` varchar(40) collate utf8_polish_ci NOT NULL,
  `hash` varchar(40) collate utf8_polish_ci NOT NULL,
  `root` tinyint(4) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;

CREATE TABLE `acl_access` (
  `acl_role_id` int(11) NOT NULL,
  `acl_resource_id` int(11) NOT NULL,
  `access` tinyint(4) NOT NULL,
  KEY `acl_role_id` (`acl_role_id`,`acl_resource_id`),
  KEY `acl_resource_id` (`acl_resource_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;

ALTER TABLE `acl_access` ADD CONSTRAINT `acl_access_ibfk_1` FOREIGN KEY
(`acl_role_id`) REFERENCES `acl_role` (`id`) ON DELETE CASCADE,
ALTER TABLE `acl_access` ADD CONSTRAINT `acl_access_ibfk_1` FOREIGN KEY
(`acl_role_id`) REFERENCES `acl_role` (`id`) ON DELETE CASCADE, ADD
CONSTRAINT `acl_access_ibfk_2` FOREIGN KEY (`acl_resource_id`) REFERENCES
`acl_resource` (`id`) ON DELETE CASCADE;
```

Przyjrzyjmy się tak zdefiniowanej strukturze. W celu ułatwienia zarządzania bazą, wybrałem typ tabel InnoDB umożliwiający zdefiniowanie kluczy obcych automatyzujących szereg operacji takich, jak sprzątnięcie po usuniętych rekordach z ustawionymi relacjami.

Tabela pierwsza, *acl_resource* definiuje dostępne zasoby. Każdy z nich opisany jest czytelnym dla człowieka tytułem, haszem będącym zapisem czytelnym dla PHP (*www/admin/news/edit*) oraz domyślną wartością. Druga z tabel, *acl_role* określająca wszystkie role, ma podobną strukturę: *title* to nazwa roli czytelna dla człowieka, *hash* to nazwa czytelna dla komputera, natomiast pole *root* pozwala w prosty sposób zdefiniować superużytkownika mającego dostęp do wszystkiego. Ostatnia z tabel to *acl_access* wiążąca rolę z zasobem i umożliwiająca ustawienie, czy podany zasób ma być dla danej roli dostępny, czy nie. Jeżeli nie istnieje rekord łączący daną rolę z zasobem, oznacza to automatycznie brak dostępu.

Klasa główna

Pora przystąpić do trzonu systemu ACL, który zarządza uprawnieniami. Będzie ona składać się z kilku metod umożliwiających definiowanie nowych reguł czy ustawianie informacji o roli oraz użytkownika, a także metod służących do kontroli dostępu przez programistę już w trakcie wykonywania właściwego kodu akcji.

```
<?php
define('ACL_CHECK', 0);
define('ACL_ALLOW_ALL', 1);
```

```

define('ACL_DENY_ALL', 2);

interface aclInterface
{
    public function loadRules($acl);
}

class aclClass
{
    public $account = array('anonymous' => true);
    public $role;

    private $state = ACL_CHECK;
    private $rules = array();
    private $size = 0;
}

```

Pierwsze trzy stałe definiują sposób traktowania uprawnień. Domyślnie stan systemu jest ustawiony na *ACL_CHECK*, co oznacza, że system sprawdza, czy użytkownik ma gdzieś dostęp, czy nie ma. *ACL_DENY_ALL* z definicji blokuje dostęp wszędzie, zaś *ACL_ALLOW_ALL* daje pełen dostęp. Interfejs *aclInterface* pozwala na tworzenie klas – źródeł uprawnień, które będą ładować reguły, np. z bazy danych. Służy ku temu metoda *loadRules()*, wywoływana automatycznie. Przekazywany jest do niej obiekt *aclClass*, czyli głównej klasy kontroli dostępu, która zaczyna się w dolnej części listingu. Dwa pola publiczne przechowują dane użytkownika oraz jego roli. W *\$rules* przechowywane będzie drzewo uprawnień zbudowane po stronie PHP. Do zarządzania drzewem służą dwie metody: *insertNode()* wstawiająca nowy węzeł oraz *findNode()* wyszukująca na aktualnym poziomie węzeł o podanej nazwie. Napišemy je jednak później.

```

public function __construct()
{
    // Utworz korzen drzewa na dzien dobry
    $this -> insertNode(-1, '', true);
} // end __construct();

public function setAccount($id)
{
    $model = new userModel;

    if($id > 0)
    {
        // Czytamy wszystko z profilu uzytkownika
        $data = $model -> chooseAcl($id);

        $this -> account = $data['Account'];
        $this -> account['anonymous'] = false;
        $this -> role = $data['Role'];
    }
} // end setAccount();

public function setRole($id)
{
    $model = new roleModel;

    if($id > 0)
    {
        // Czytamy wszystko z samej roli
        $this -> role = $model -> choose($id);
    }
} // end setRole();

```

Zadaniem konstruktora jest utworzenie korzenia drzewa, ponieważ uprawnienia wypadają gdzieś podpiąć. Dwie kolejne metody: *setAccount()* oraz *setRole()* pozwalają na załadowanie danych użytkownika oraz roli

na podstawie podanego ID. Chciałbym tu zwrócić uwagę na dwie rzeczy: metoda *setAccount()* automatycznie powinna łączyć także i rolę użytkownika, by oszczędzić zapytania, po czym jakąś pętlą *foreach* należy rozbić otrzymany rekord na dwie tablice. Druga rzecz jest taka, że treść tych dwóch zapytań należy sobie dopasować do własnej struktury serwisu, ponieważ w tym konkretnym wypadku użyłem tutaj mojego DAO, a wiadomo, że co programista, to ma inne upodobania, jeśli chodzi o nazewnictwo, umiejscowienie klas itd.

```
public function addRules(aclInterface $if)
{
    if($this -> state == ACL_CHECK)
    {
        $if -> loadRules($this);
    }
} // end addRules();
```

Za pomocą tej metody będzie możliwe dodanie nowych źródeł uprawnień. Zauważmy, że są one ładowane tylko wtedy, kiedy ustawiony jest tryb sprawdzania. Oczywiście jest, że jeśli byśmy zabraniali lub zezwalali na wszystko, to nie warto marnować czasu na takie ładowanie. Za parametr przekazujemy obiekt klasy implementującej *aclInterface*.

```
public function setState($state)
{
    $this -> state = $state;
} // end setState();

public function isAnonymous()
{
    return $this -> account['anonymous'];
} // end isAnonymous();
```

Kolejne dwie metody administracyjne. Pierwsza pozwala zmienić stan pracy, a druga zwraca, czy mamy do czynienia z anonimowym gościem. Najważniejsza metoda jest przed nami.

```
public function isAllowed($rule)
{
    switch($this -> state)
    {
        case ACL_CHECK:
            $items = explode('/', $rule);
            $id = 0;
            $i = 0;
            $cnt = sizeof($items);
            foreach($items as $item)
            {
                if(is_null($id = $this -> findNode($id,
$item)))
                {
                    return false;
                }
                elseif($i == $cnt - 1)
                {
                    return $this -> rules[$id][1];
                }
                elseif($this -> rules[$id][1] == false)
                {
                    return false;
                }
                $i++;
            }
            return false;
        case ACL_ALLOW_ALL:
            return true;
        case ACL_DENY_ALL:
```

```

        return false;
    }
} // end isAllowed();

```

To właśnie tutaj dokonujemy sprawdzania, czy użytkownikowi można zezwolić na dostęp do danego zasobu, czy też nie. Najbardziej skomplikowane jest sprawdzanie w stanie *ACL_CHECK*, ponieważ wymaga on zejścia w głąb drzewka. Na początku rozbijamy funkcją *explode()* „ścieżkę” na poszczególne elementy składowe oraz ustawiamy parę zmiennych pomocniczych. Po wygenerowanej tablicy puszczaemy pętlę *foreach*, którą schodzimy w głąb. W zmiennej *\$id* przechowywany jest zawsze identyfikator węzła, który aktualnie przeszukujemy. Metoda *findNode()* szuka nowego węzła o podanej nazwie, którego rodzicem jest *\$id*. Jeśli takowy odnajdzie, ustawiamy go jako nowego rodzica i cały proces powtarza się od początku. Jeżeli w trakcie schodzenia okaże się, że fragmentu drzewa brakuje lub którekolwiek z uprawnień pośrednich jest niekorzystne, zwracamy *false* (brak dostępu), natomiast w przypadku dotarcia na żądaną głębokość zwracamy to, co jest ustawione.

```

public function insertRule($resource, $state)
{
    $items = explode('/', $resource);

    $id = 0;
    $x = NULL;
    $i = 0;
    $cnt = sizeof($items);
    foreach($items as $item)
    {
        if(is_null($x = $this -> findNode($id, $item)))
        {
            $x = $this -> insertNode($id, $item, $i != $cnt
- 1 ? false : $state);
        }
        elseif($i == $cnt - 1)
        {
            $this -> rules[$x][1] = $state;
        }
        $id = $x;
        $i++;
    }
} // end insertRule();

```

Tutaj na podobnej zasadzie umieszczamy nową regułę w drzewku. Rozbijamy ścieżkę na elementy składowe i schodzimy w głąb. Jeżeli jakiś węzeł nie istnieje, tworzymy go, ustawiając domyślny stan na *false*.

Ostatnia rzecz to napisanie dwóch metod pomocniczych do zarządzania drzewem. Do reprezentowania go w pamięci wybrałem zwyczajną listę. Każdy jej element zawiera dwa pola określające indeks rodzica oraz indeks następnego elementu na danej głębokości. -1 oznacza brak następnika lub rodzica (korzeń). Ponieważ jest to mechanizm wewnętrzny, który na diabła jest programiście korzystającemu z systemu, nie opłacało się do reprezentowania każdego z tych pól wykorzystywać tablic asocjacyjnych, lecz zwyczajne, z numerycznymi indeksami. Stąd też pojedynczy element tablicy *aclClass::\$rules* wygląda następująco:

- 0 – nazwa
- 1 – stan (dostępny, niedostępny)
- 2 – indeks rodzica
- 3 – indeks następnika
- 4 – indeks pierwszego dziecka
- 5 – indeks ostatniego dziecka

```

private function insertNode($parent, $name, $state)
{
    $this -> rules[$this->size] = array(

```

```

        0 => $name,          // Nazwa
        1 => $state,       // Stan
        2 => $parent,     // Rodzic
        3 => -1,          // Nastepny
        4 => -1,          // Pierwsze dziecko
        5 => -1           // Ostatnie dziecko
    );

    if($parent >= 0)
    {
        if($this -> rules[$parent][5] != -1)
        {
            $this -> rules[$this->rules[$parent][5]][3] =
$this->size;
            $this->rules[$parent][5] = $this -> size;
        }
        else
        {
            $this->rules[$parent][5] = $this->
>rules[$parent][4] = $this -> size;
        }
        $this -> size++;
        return $this -> size - 1;
    } // end insertNode();

```

Metoda *insertNode()* ma za zadanie dodać nowy węzeł do drzewa. Na początku rejestruje nowy element w pierwszym wolnym indeksie tablicy *aclClass::\$rules*, po czym, jeśli w parametrach określiliśmy indeks rodzica, podpiną nowy węzeł tam, gdzie trzeba poprzez ustawienie stosownych pól w obu elementach. Jako wynik zwracany jest indeks nowego węzła.

```

private function findNode($parent, $name)
{
    if(isset($this -> rules[$parent]))
    {
        $sid = $this -> rules[$parent][4];
        while($sid != -1)
        {
            if($this -> rules[$sid][0] == $name)
            {
                return $sid;
            }
            $sid = $this -> rules[$sid][3];
        }
        return NULL;
    } // end findNode();
} // end aclClass();
?>

```

A oto metoda skanująca wszystkie dzieci podanego rodzica w poszukiwaniu takiego, który ma zdefiniowaną nazwę *\$name*. Wykorzystywaliśmy ją, jak pamiętamy, do schodzenia w głąb drzewa.

Jak widać, napisany przez nas system to miks algorytmiki i mechanizmów bazodanowych. Trzeba mieć świadomość, że bardziej zaawansowane mechanizmy mają to do siebie, że nie można ich rozwiązać jednym prostym zapytaniem, lecz trzeba zapracować do pracy bądź jakiś algorytm, bądź odpowiednio zorganizować przepływ danych i rozłożyć ich przetwarzanie równomiernie między bazę oraz skrypt.

Interfejs bazy danych

Skonstruujemy teraz klasę, która będzie łądownać uprawnienia z naszej bazy danych. Musi on implementować interfejs *aclInterface*. Kod PHP wygląda następująco:

```

<?php

class aclGeneral implements aclInterface
{
    private $userId;
    private $primaryRoleId;

    public function setOptions($userId, $primaryRoleId)
    {
        $this -> userId = $userId;
        $this -> primaryRoleId = $primaryRoleId;
    } // end setOptions();

    public function loadRules($acl)
    {
        if($this -> userId == 0)
        {
            $acl -> setRole($this -> primaryRoleId);
        }
        else
        {
            $acl -> setAccount($this -> userId);
        }

        if($acl -> role['root'])
        {
            $acl -> setState(ACL_ALLOW_ALL);
        }
        else
        {
            global $sql;
            $stmt = $sql -> prepare('SELECT r.hash, a.access FROM
`'.DB_PREFIX.'acl_access`      a,      `'.DB_PREFIX.'acl_resource`      r      WHERE
a.acl_resource_id = r.id AND a.acl_role_id = :id');
            $stmt -> bindValue(':id', $acl -> role['id'],
PDO::PARAM_INT);

            $stmt -> execute();

            while($row = $stmt -> fetch())
            {
                $acl -> insertRule($row['hash'],
$row['access']);
            }
            $stmt -> closeCursor();
        }
    } // end loadRules();
} // end aclGeneral;

?>

```

Metoda *setOptions()* służy programiście, który w ten sposób może przekazać ID zalogowanego użytkownika. Gdyby nikt nie był zalogowany (wizyta anonimowa), musimy podać ID roli określającej uprawnienia użytkowników anonimowych. W *loadRules()* dostajemy do naszych rąk obiekt klasy *aclClass*, do której musimy powkładać uprawnienia. Najpierw wczytujemy dane roli oraz użytkownika. Jeśli mamy do czynienia z kontem „root”, ustawiamy pełen dostęp, a w przeciwnym razie przystępujemy do pasywnego wczytywania listy uprawnień z bazy. Każdy pobrany zasób wstawiamy metodą *aclClass::insertRule()* do drzewa uprawnień.

W Twoich własnych źródłach uprawnień musisz sam przyjąć konwencję przetłumaczenia ich wewnętrznego formatu na postać drzewa. Wracając do przykładu z systemem CMS – tutaj naraz musi być

załadowana tylko informacja o uprawnieniach edycyjnych aktualnie oglądanego obiektu (artykułu, newsa czy działu). Dlatego nasza klasa może budować sobie ścieżkę `/www/admin/currentObject/` i w jej obrębie stworzyć elementy np. `add`, `edit` czy `remove`, które zawsze będą w ten sposób dotyczyły aktualnie oglądanego obiektu. Dokładne dane, czy zezwolić na taką akcję, pobieramy oczywiście z bazy.

Przykład użycia

Skoro mamy gotowy kod PHP, pora na zaprezentowanie możliwości naszej klasy. Aby użyć ACL w serwisie, musimy utworzyć obiekt `aclClass` i wrzucić do niego inny obiekt dowolnej klasy implementującej `aclInterface`.

```
// tworzymy obiekt glowny
$acl = new aclClass;
// tworzymy glowne zrodlo uprawnień, konfigurujemy je i rejestrujemy
$general = new aclGeneral;
$general -> setOptions($_SESSION['user_id'], $config['primary_role_id']);
$acl -> addRules($general);

if($acl -> isAllowed('admin/blog'))
{
    echo 'Dziękujemy!';
}
else
{
    echo 'Dostęp zabroniony!';
}
```

Jak widać, ostateczne użycie jest bajecznie proste, a przy tym zyskujemy duże możliwości manipulowania uprawnieniami. Potrzeba jeszcze wypełnić oczywiście bazę danych. Oto przykładowa zawartość:

```
INSERT INTO `acl_access` (`acl_role_id`, `acl_resource_id`, `access`) VALUES
(1, 1, 1),
(1, 2, 1),
(2, 1, 1),
(2, 2, 1),
(2, 3, 1),
(1, 4, 1);

INSERT INTO `acl_resource` (`id`, `title`, `hash`, `default_value`) VALUES
(1, 'Website', 'website', 1),
(2, 'Website: Insertion', 'website/insert', 1),
(3, 'Control panel', 'admin', 0),
(4, 'Website: Options', 'website/options', 0),
(5, 'Admin: Main', 'admin/main', 0),
(6, 'Admin: Blog', 'admin/blog', 0),
(7, 'Admin: CMS', 'admin/cms', 0),
(8, 'Admin: Security', 'admin/security', 0),
(9, 'Admin: Technical', 'admin/technical', 0),
(10, 'Admin: Blog / Notes', 'admin/blog/notes', 0),
(11, 'Admin: Blog / Categories', 'admin/blog/categories', 0),
(12, 'Admin: Blog / Comments', 'admin/blog/comments', 0),
(13, 'Admin: Blog / Trackbacks', 'admin/blog/trackbacks', 0),
(14, 'Admin: Blog / Notes / Add', 'admin/blog/notes/add', 0);

INSERT INTO `acl_role` (`id`, `title`, `hash`, `root`) VALUES
(1, 'Guest', 'guest', 0),
(2, 'Editor', 'editor', 0),
(3, 'Admin', 'admin', 1);
```

Dzięki kluczom obcym, edycja takiej listy uprawnień jest uproszczona nawet z poziomu phpMyAdmina, jednak warto do tego napisać dedykowany system.

Dalsza rozbudowa

Zastanów się nad dalszymi możliwościami rozbudowy zaproponowanego tutaj systemu. Przede wszystkim spróbuj samodzielnie zaimplementować aktywne ładowanie listy uprawnień wspomniane na początku artykułu. Drugim ważnym dodatkiem jest możliwość dziedziczenia uprawnień po innej grupie w trakcie tworzenia nowej. Nadmienię, że ten problem również można rozwiązać pasywnie lub aktywnie, przy czym tutaj dla większych serwisów korzystniejsze będzie ze względów wydajnościowych pierwsze rozwiązanie, tj. dziedziczenie uprawnień podczas tworzenia nowej roli poprzez zduplikowanie rekordów w tablicy `acl_access`. W końcu uprawnienia ustawia się raz na długi czas i nie ma potrzeby, by system przy każdym żądaniu na nowo określał zależności między grupami i wykonywał karkołomne zapytania, byleby dopasować się do ustawień.

Polecam także zapoznanie się z systemem ACL dostępnym w Zend Frameworku, który posiada możliwość określania po stronie skryptu wielu ról naraz.

Zakończenie

Oczywiście precyzyjne zdefiniowanie poziomów uprawnień nie na wiele się zda, jeśli nasze skrypty będą napisane ze złamaniem wszelkich reguł dotyczących tworzenia bezpiecznych skryptów. Podatność na SQL Injection, czy stosowanie łatwych do złamania haseł sprawi, że nawet najpotężniejszy system uprawnień stanie się bezużyteczny. Dopiero w połączeniu z innymi, odpowiednio skonstruowanymi elementami serwisu odpowiadającymi za bezpieczeństwo mamy pewność, że nikt niepowołany nie będzie manipulować witryną.

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na www.zyxist.com

Licencja

Artykuł rozpowszechniany jest na licencji **Creative Commons Uznanie autorstwa-Użycie niekomercyjne-Bez utworów zależnych 2.5 Polska**.

Wolno:

- kopiować, rozpowszechniać, odtwarzać i wykonywać utwór

Na następujących warunkach:

1. *Uznanie autorstwa.* Utwór należy oznaczyć w sposób określony przez Twórcę lub Licencjodawcę*.
2. *Użycie niekomercyjne.* Nie wolno używać tego utworu do celów komercyjnych.
3. *Bez utworów zależnych.* Nie wolno zmieniać, przekształcać ani tworzyć nowych dzieł na podstawie tego utworu.

W celu ponownego użycia utworu lub rozpowszechniania utworu należy wyjaśnić innym warunki licencji, na której udostępnia się utwór.

Każdy z tych warunków może zostać uchylony, jeśli uzyska się zezwolenie właściciela praw autorskich.

Powyższe postanowienia w żaden sposób nie naruszają uprawnień wynikających z dozwolonego użytku ani żadnych innych praw.

Internetowa skrócona wersja licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/>

Pełen tekst licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/legalcode>

Podczas publikacji należy podać następujące informacje:

1. Link do internetowej skróconej wersji licencji.
2. Informację o wersji publikowanego tekstu.
3. Informacje o autorze w następujący sposób:

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na www.zyxist.com