The PHP Company

Building secure web applications using ZF2

by Enrico Zimuel (enrico@zend.com)
Senior Software Engineer
Zend Framework Core Team
Zend Technologies Ltd

About me

- Enrico Zimuel (@ezimuel)
- Software Engineer since 1996
  - Assembly x86, C/C++, Java, Perl, PHP
- PHP Engineer at Zend Technologies in the Zend Framework Team
- International speaker, author of articles and books on PHP and secure programming
- Researcher programmer at Informatics Institute of University of Amsterdam
- Co-founder of PUG Torino (Italy)

OWASP Top Ten Attacks

zend framework 2

1) Cross-Site Scripting (XSS)
2) Injection Flaws
3) Malicious File Execution
4) Insecure Direct Object Reference
5) Cross Site Request Forgery (CSRF)
6) Information Leakage and Improper Error Handling
7) Broken Authentication and Session Management
8) Insecure Cryptographic Storage
9) Insecure Communications
10) Failure to Restrict URL Access

Security practices

zend framework 2

“Filter Input, Escape Output”

Yes, but it's not enough!

zend

Security tools in ZF2

- Zend\Authentication
- Zend\Captcha
- Zend\Crypt
- Zend\Escaper
- Zend\Filter
- Zend\InputFilter
- Zend\Permissions
- Zend\Math
- Zend\Validator

Zend\Authentication

Authentication

- **Zend\Authentication** provides API for authentication and includes concrete authentication adapters for common use case scenarios.
- Adapters:
  - Database Table
  - Digest
  - HTTP
  - LDAP
  - Your adapter

Example

```php
use Zend\Authentication\AuthenticationService;

// instantiate the authentication service
$auth = new AuthenticationService();

// Set up the authentication adapter
$authAdapter = new My\Auth\Adapter($username, $password);

// Attempt authentication, saving the result
$result = $auth->authenticate($authAdapter);

if (!$result->isValid()) {
    // Authentication failed; print the reasons why
    foreach ($result->getMessages() as $message) {
        echo "$message\n";
    }
} else {
    // Authentication succeeded; the identity ($username) is stored
    // in the session
    // $result->getIdentity() === $auth->getIdentity()
    // $result->getIdentity() === $username
}
```

Zend\Permissions

Zend\Permissions\Acl

- The component provides a lightweight and flexible access control list (ACL) implementation for privileges management
- Terminology:
  - a **resource** is an object to which access is controlled
  - a **role** is an object that may request access to a resource

Example

```php
use Zend\Permissions\Acl\Acl;
use Zend\Permissions\Acl\Role\GenericRole as Role;
use Zend\Permissions\Acl\Resource\GenericResource as Resource;

$acl = new Acl();

$acl->addRole(new Role('guest'))
    ->addRole(new Role('member'))
    ->addRole(new Role('admin'));

$parents = array('guest', 'member', 'admin');
$acl->addRole(new Role('someUser'), $parents);

$acl->addResource(new Resource('someResource'));

$acl->deny('guest', 'someResource');
$acl->allow('member', 'someResource');

echo $acl->isAllowed('someUser', 'someResource') ? 'allowed' : 'denied';
```

Zend\Permissions\Rbac (≥ ZF2.1)

- Provides a lightweight Role-Based Access Control implementation based around PHP 5.3's SPL *RecursiveIterator* and *RecursiveIteratorIterator*
- RBAC differs from access control lists (ACL) by putting the emphasis on roles and their permissions rather than objects (resources)
- Terminology:
  - an **identity** has one or more roles
  - a **role** requests access to a permission
  - a **permission** is given to a role

Zend\Filter

Zend\Filter

- The Zend\Filter component provides a set of commonly needed data filters. It also provides a simple filter chaining mechanism by which multiple filters may be applied to a single datum in a user-defined order.
- Remember: “Filter the input, always”

Standard Filter Classes

- Alnum
- Alpha
- BaseName
- Boolean
- Callback
- Compress/Decompress
- Digits
- Dir
- Encrypt/Decrypt
- HtmlEntities
- Int
- Null
- NumberFormat
- PregReplace
- RealPath
- StringToLower/ToUpper
- StringTrim
- StripNewLines/Tags

Zend\Validator

Zend\Validator

- The Zend\Validator component provides a set of commonly needed validators. It also provides a simple validator chaining mechanism by which multiple validators may be applied to a single datum in a user-defined order.
- A validator examines its input with respect to some requirements and produces a boolean result - whether the input successfully validates against the requirements.

Example

zend framework 2

```php
$validator = new Zend\Validator\EmailAddress();

if ($validator->isValid($email)) {
    // email appears to be valid
} else {
    // email is invalid; print the reasons
    foreach ($validator->getMessages() as $messageId => $message) {
        echo "Validation failure '$messageId': $message\n";
    }
}
```


zend

Standard Validator Classes

- Alnum
- Alpha
- Barcode
- Between
- Callback
- CreditCard
- Date
- Db\RecordExists and NoRecordExists
- Digits
- EmailAddress
- GreaterThan/LessThan
- Hex
- Hostname
- Iban
- Identical
- InArray
- Ip
- Isbn
- NotEmpty
- PostCode
- Regex
- Sitemap
- Step
- StringLength

Zend\InputFilter

Zend\InputFilter

- The Zend\InputFilter component can be used to filter and validate generic sets of input data. For instance, you could use it to filter $_GET or $_POST values, CLI arguments, etc.
- Remember: “Filter the input, always”

Example

```php
use Zend\InputFilter\InputFilter;
use Zend\InputFilter\Input;
use Zend\Validator;

$email = new Input('email');
$email->getValidatorChain()
      ->addValidator(new Validator\EmailAddress());

$password = new Input('password');
$password->getValidatorChain()
         ->addValidator(new Validator\StringLength(8));

$inputFilter = new InputFilter();
$inputFilter->add($email)
            ->add($password)
            ->setData($_POST);

if ($inputFilter->isValid()) {
    echo "The form is valid\n";
} else {
    echo "The form is not valid\n";
    foreach ($inputFilter->getInvalidInput() as $error) {
        print_r ($error->getMessages());
    }
}
```

Zend\Escaper

Escaper

- Escape the output, multiple formats:
  - escapeHtml()
  - escapeHtmlAttr()
  - escapeJs()
  - escapeUrl()
  - EscapeCss()
- Remember: “Escape the output, always”

Zend\Captcha

Zend\Captcha

- CAPTCHA stands for “Completely Automated Public Turing test to tell Computers and Humans Apart”; it is used as a challenge-response to ensure that the individual submitting information is a human and not an automated process
- A captcha is used to prevent spam submissions

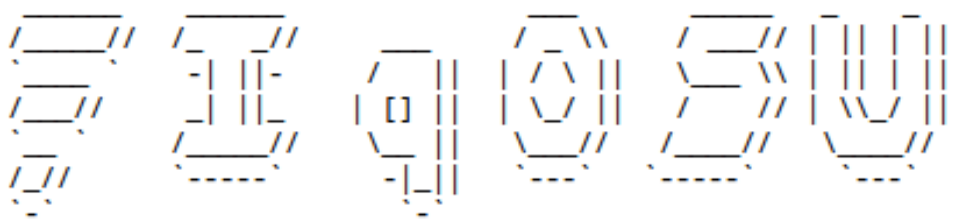
Example

```php
// Originating request:
$captcha = new Zend\Captcha\Figlet(array(
    'name' => 'foo',
    'wordLen' => 6,
    'timeout' => 300,
));

$id = $captcha->generate();

//this will output a Figlet string
echo $captcha->getFiglet()->render($captcha->getWord());


// On a subsequent request:
// Assume a captcha setup as before, with corresponding form fields, the value of $_POST['foo']
// would be key/value array: id => captcha ID, input => captcha value
if ($captcha->isValid($_POST['foo'], $_POST)) {
    // Validated!
}
```



Captcha adapters

- Zend\Captcha\AbstractWord
- Zend\Captcha\Dumb
- Zend\Captcha\Figlet
- Zend\Captcha\Image
- Zend\Captcha\ReCaptcha

Zend\Crypt

Cryptography is hard

zend framework 2

- Cryptography is hard, and the implementation is even more hard!
- PHP offers some crypto primitives but you need some cryptography background to use it (this is not straightforward)
- This can respresent a barrier that discouraged most of the PHP developers

Cryptography using ZF2

- **Zend\Crypt** wants to help PHP developers to use *strong cryptography* in their projects
- In PHP we have built-in functions and extensions for cryptography scopes:
  - crypt()
  - Mcrypt
  - OpenSSL
  - Hash (by default in PHP 5.1.2)
  - Mhash (emulated by Hash from PHP 5.3)

Zend\Crypt

- **Zend\Crypt** components:
  - Zend\Crypt**Password**
  - Zend\Crypt**Key\Derivation**
  - Zend\Crypt**Symmetic**
  - Zend\Crypt**PublicKey**
  - Zend\Crypt**Hash**
  - Zend\Crypt**Hmac**
  - Zend\Crypt**BlockCipher**

How to encrypt sensitive data

Encrypt and Authenticate

- **Zend\Crypt\BlockCipher** can be used to encrypt/decrypt sensitive data (symmetric encryption)
- Provides **encryption + authentication (HMAC)**
- Simplified API:
  - **setKey($key)**
  - **encrypt($data)**
  - **decrypt($data)**
- It uses the **Mcrypt** adapter (Zend\Crypt\Symmetric\Mcrypt)

Default encryption values

- Default values used by **BlockCipher**:
  - AES algorithm (key of 256 bits)
  - CBC mode + HMAC (SHA-256)
  - PKCS7 padding mode (RFC 5652)
  - PBKDF2 to generate encryption key + authentication key for HMAC
  - Random IV for each encryption

Example: AES encryption

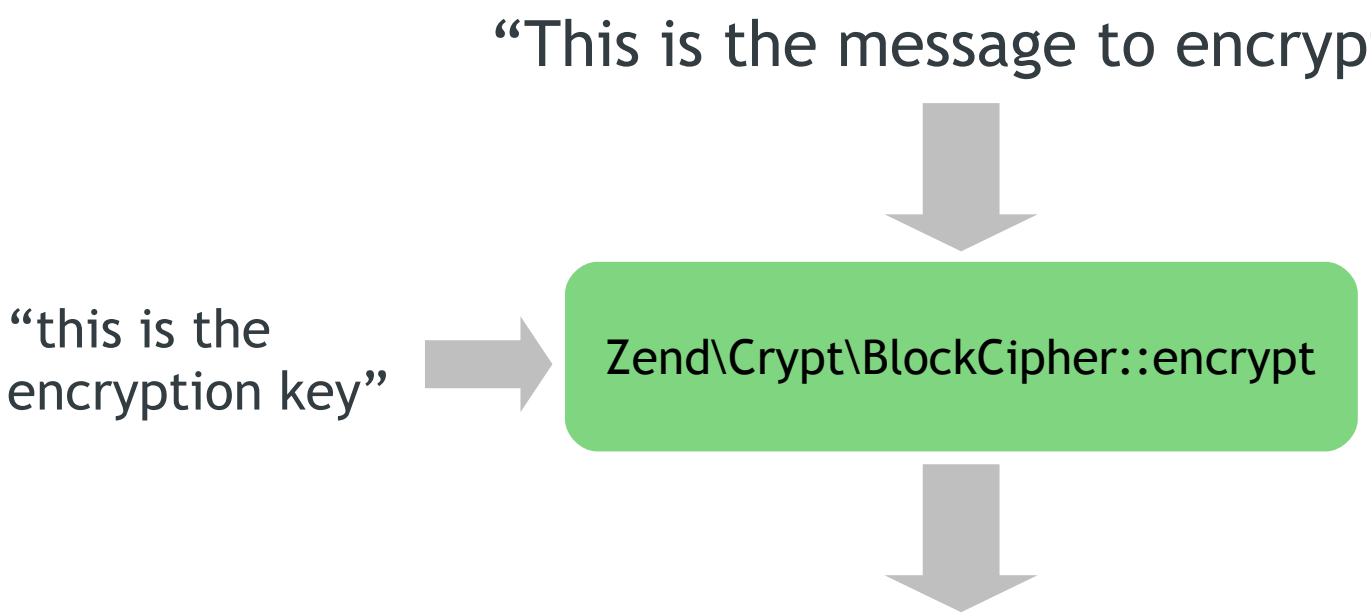
```php
// encrypt a text and store it in a file
use Zend\Crypt\BlockCipher;

$cipher = BlockCipher::factory(
    'mcrypt',
    array('algorithm' => 'aes')
);
$cipher->setKey('this is the encryption key');
$plaintext = 'This is the message to encrypt';
$encrypted = $cipher->encrypt($plaintext);

printf("Encrypted text: %s\n", $encrypted);
file_put_contents('test.crypt', $encrypted);
```

The encrypted text is encoded in Base64, you can switch to binary output using **setBinaryOutput(true)**

Example: encryption output



064b05b885342dc91e7915e492715acf0f896620d
bf9d1e00dd0798b15e72e8cZg+hO34C3f3eb8TeJ
M9xWQRVex1y5zeLrBsNv+dYeVy3SBJa+pXZbUQY
NZw0xS9s

HMAC, **IV**, **ciphertext**

Example: decrypt

zend framework 2

```php
// decrypt a text stored in a file
use Zend\Crypt\BlockCipher;

$cipher = BlockCipher::factory(
    'mcrypt',
    array('algorithm' => 'aes')
);
$cipher->setKey('this is the encryption key');
$encrypted = file_get_contents('test.crypt');
$plaintext = $cipher->decrypt($encrypted);

printf("Decrypted text: %s\n", $plaintext);
```


zend

How to safely store a user's password

How to store a password

- How do you safely store a password?
- Old school (insecure):
  - MD5/SHA1(password)
  - MD5/SHA1(password . *salt*)
    where *salt* is a random string
- New school (secure):
  - **bcrypt**

Why MD5/SHA1 ±salt is not secure?

- Dictionary/brute force attacks more efficient
- **GPU**-accelerated password hash:
  - **Whitepixel project**
    whitepixel.zorinaq.com
    4 Dual HD 5970, ~ $2800

Algorithm	Speed	8 chars	9 chars	10 chars
md5($pass)	33 billion p/s	1 ½ hour	4 ½ days	294 days

bcrypt

- **bcrypt** uses Blowfish cipher + iterations to generate secure hash values
- **bcrypt** is secure against brute force attacks because is slow, very slow (that means attacks need huge amount of time to be completed)
- The algorithm needs a *salt* value and a work factor parameter (*cost*), which allows you to determine how expensive the **bcrypt** function will be

Zend\Crypt\Password\Bcrypt

- We used the **crypt()** function of PHP to implement the **bcrypt** algorithm
- The *cost* is an integer value from 4 to 31
- The default value for Zend\Crypt\Password\Bcrypt is 14 (that is equivalent to 1 second of computation using an Intel Core i5 CPU at 3.3 Ghz).
- The cost value depends on the CPU speed, check on your system! We suggest to consume **at least 1 second**.

Example: bcrypt

```php
use Zend\Crypt\Password\Bcrypt;

$bcrypt = new Bcrypt();
$start  = microtime(true);
$hash   = $bcrypt->create('password');
$end    = microtime(true);

printf ("Hash  : %s\n", $hash);
printf ("Exec. time: %.2f\n", $end-$start);
```

- The output of **bcrypt** ($hash) is a string of 60 bytes

How to verify a password

- To check if a password is valid against an hash value we can use the method:
  - **Bcrypt::verify($password, $hash)**

  where **$password** is the value to check and **$hash** is the hash value generated by bcrypt
- This method returns true if the password is valid and false otherwise

Secure random numbers in PHP

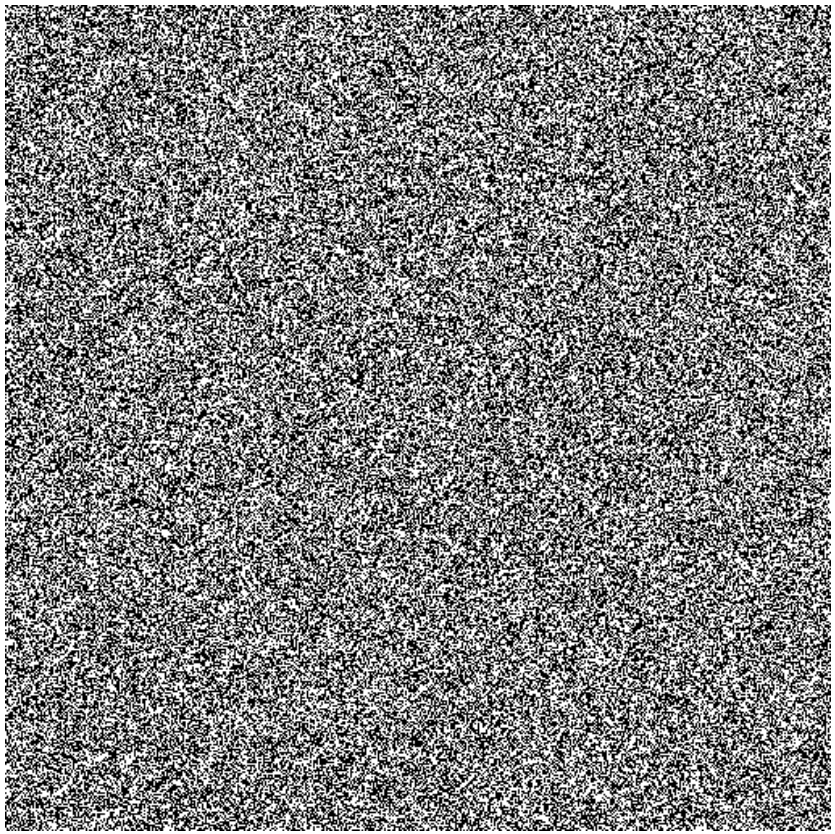
PHP vs. randomness

zend framework 2

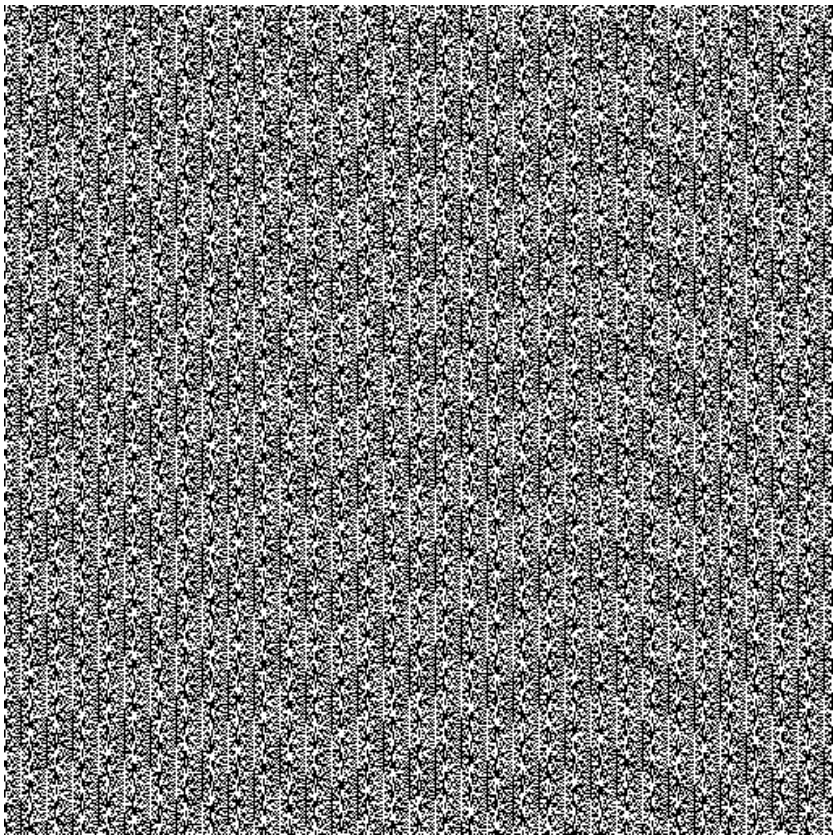
- How generate a pseudo-random value in PHP?
- Not good for cryptography purpose:
  - rand()
  - mt_rand()
- Good for cryptography (PHP 5.3+):
  - openssl_random_pseudo_bytes()

rand() is not so random :(

Pseudo-random bits



rand() of PHP on Windows



Source: random.org

Random Number Generator in ZF2

- We refactored the random number generator in ZF2 to use (in order):
  1) **openssl_random_pseudo_bytes()**
  2) **mcrypt_create_iv(), with MCRYPT_DEV_URANDOM**
  3) **mt_rand(), not used for cryptography!**
- OpenSSL provides secure random numbers
- Mcrypt with /dev/urandom provides good security
- mt_rand() is not secure for crypto purposes

Random number in Zend\Math

- We provides a couple of methods for RNG:
  - Zend\Math\Math::**randBytes**($length, $strong = false)
  - Zend\Math\Math::**rand**($min, $max, $strong = false)
- **randBytes()** generates *$length* random bytes
- **rand()** generates a random number between $min and $max
- If **$strong === true**, the functions use only OpenSSL or Mcrypt (if PHP doesn't support these extensions throw an Exception)

Some references

- Colin Percival, *Stronger Key Derivation via Sequential Memory-Hard Functions*, presented at BSDCan'09, May 2009 (link)
- T. Myer, M. Southwell, *Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses*, Apress, 2 edition, 2010
- P. Niels, T. J. Sutton, *A Future-Adaptable Password Scheme*, Proceedings of USENIX Annual Technical Conference, 1999 (link)
- Chris Shiflett, *Essential PHP Security. A Guide to Building Secure Web Applications*, O'Reilly Media, 2005
- Enrico Zimuel, *Cryptography made easy using Zend Framework 2*, Zend Webinar, 2012 (video - slides)
- Enrico Zimuel, *Cryptography in PHP. How to protect sensitive data in PHP using cryptography,* Web & PHP Magazine. Issue 2/2012 (link)

Thank you!

- More information
  - http://framework.zend.com
  - Send an email to enrico@zend.com
- IRC channels (freenode)
  - #zftalk, #zftalk.dev

zend framework 2

The **most popular framework** for modern, high-performing PHP applications

zend