

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- HTML
- Java
- JavaScript
- Kotlin
- Objective C
- PHP**
- PL/I
- PL/SQL
- Python
- RPG
- Ruby
- Scala
- Swift
- Terraform
- Text
- TypeScript
- T-SQL
- VB.NET
- VB6
- XML



PHP static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your PHP code

All rules 268

Vulnerability 40

Bug 51

Security Hotspot 33

Code Smell 144

Tags ▾

Search by name... 🔍

be used to end lines	Code Smell
More than one property should not be declared per statement	Code Smell
The "var" keyword should not be used	Code Smell
"<?php" and "<?=" tags should be used	Code Smell
File names should comply with a naming convention	Code Smell
Comments should not be located at the end of lines of code	Code Smell
Local variable and function parameter names should comply with a naming convention	Code Smell
Field names should comply with a naming convention	Code Smell
Lines should not end with trailing whitespaces	Code Smell
Files should contain an empty newline at the end	Code Smell
Modifiers should be declared in the correct order	Code Smell
An open curly brace should be located at the beginning of a line	Code Smell

Encrypting data is security-sensitive

Analyze your code

Security Hotspot Critical ⓘ

Encrypting data is security-sensitive. It has led in the past to the following vulnerabilities:

- [CVE-2017-7902](#)
- [CVE-2006-1378](#)
- [CVE-2003-1376](#)

Proper encryption requires both the encryption algorithm and the key to be strong. Obviously the private key needs to remain secret and be renewed regularly. However these are not the only means to defeat or weaken an encryption.





This rule flags function calls that initiate encryption/decryption.

Ask Yourself Whether

- the private key might not be random, strong enough or the same key is reused for a long long time.
- the private key might be compromised. It can happen when it is stored in an unsafe place or when it was transferred in an unsafe manner.
- the key exchange is made without properly authenticating the receiver.
- the encryption algorithm is not strong enough for the level of protection required. Note that encryption algorithms strength decreases as time passes.
- the chosen encryption library is deemed unsafe.
- a nonce is used, and the same value is reused multiple times, or the nonce is not random.
- the RSA algorithm is used, and it does not incorporate an Optimal Asymmetric Encryption Padding (OAEP), which might weaken the encryption.
- the CBC (Cypher Block Chaining) algorithm is used for encryption, and it's IV (Initialization Vector) is not generated using a secure random algorithm, or it is reused.
- the Advanced Encryption Standard (AES) encryption algorithm is used with an unsecure mode. See the recommended practices for more information.

You are at risk if you answered yes to any of those questions.

- Recommended Secure Coding Practices**
- Generate encryption keys using secure random algorithms.
 - When generating cryptographic keys (or key pairs), it is important to use a key length that provides enough entropy against brute-force attacks. For the Blowfish algorithm the key should be at least 128 bits long, while for the RSA algorithm it should be at least 2048 bits long.
 - Regenerate the keys regularly.
 - Always store the keys in a safe location and transfer them only over safe channels.
 - If there is an exchange of cryptographic keys, check first the identity of the receiver.
 - Only use strong encryption algorithms. Check regularly that the algorithm is still deemed secure. It is also imperative that they are implemented correctly. Use only encryption libraries which are deemed secure. Do not define your own encryption algorithms as they will most probably have flaws.
 - When a nonce is used, generate it randomly every time.
 - When using the RSA algorithm, incorporate an Optimal Asymmetric Encryption Padding (OAEP).

An open curly brace should be located at the end of a line
 Code Smell
Tabulation characters should not be used
 Code Smell
Method and function names should comply with a naming convention
 Code Smell
Creating cookies with broadly defined "domain" flags is security-sensitive
 Security Hotspot

- When CBC is used for encryption, the IV must be random and unpredictable. Otherwise it exposes the encrypted value to crypto-analysis attacks like "Chosen-Plaintext Attacks". Thus a secure random algorithm should be used. An IV value should be associated to one and only one encryption cycle, because the IV's purpose is to ensure that the same plaintext encrypted twice will yield two different ciphertexts.
- The Advanced Encryption Standard (AES) encryption algorithm can be used with various modes. Galois/Counter Mode (GCM) with no padding should be preferred to the following combinations which are not secured:
 - Electronic Codebook (ECB) mode: Under a given key, any given plaintext block always gets encrypted to the same ciphertext block. Thus, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.
 - Cipher Block Chaining (CBC) with PKCS#5 padding (or PKCS#7) is susceptible to padding oracle attacks.

Sensitive Code Example

Builtin functions

```
function myEncrypt($cipher, $key, $data, $mode, $iv, $option)
{
    mcrypt_ecb ($cipher, $key, $data, $mode); // Sensitive
    mcrypt_cfb($cipher, $key, $data, $mode, $iv); // Sensiti
    mcrypt_cbc($cipher, $key, $data, $mode, $iv); // Sensiti
    mcrypt_encrypt($cipher, $key, $data, $mode); // Sensitiv

    openssl_encrypt($data, $cipher, $key, $options, $iv); //
    openssl_public_encrypt($data, $crypted, $key, $padding);
    openssl_pkcs7_encrypt($infile, $outfile, $recipcerts, $h
    openssl_seal($data, $sealed_data, $env_keys, $pub_key_id

    sodium_crypto_aead_aes256gcm_encrypt ($data, $ad, $nonce
    sodium_crypto_aead_chacha20poly1305_encrypt ($data, $ad,
    sodium_crypto_aead_chacha20poly1305_ietf_encrypt ($data,
    sodium_crypto_aead_xchacha20poly1305_ietf_encrypt ($data
    sodium_crypto_box_seal ($data, $key); // Sensitive
    sodium_crypto_box ($data, $nonce, $key); // Sensitive
    sodium_crypto_secretbox ($data, $nonce, $key); // Sensit
    sodium_crypto_stream_xor ($data, $nonce, $key); // Sensi
}
```

CakePHP

```
use Cake\Utility\Security;

function myCakeEncrypt($key, $data, $engine)
{
    Security::encrypt($data, $key); // Sensitive

    // Do not use custom made engines and remember that Mcry
    Security::engine($engine); // Sensitive. Setting the enc
}
```

CodeIgniter

```
class EncryptionController extends CI_Controller
{
    public function __construct()
    {
        parent::__construct();
        $this->load->library('encryption');
    }

    public function index()
    {
        $this->encryption->create_key(16); // Sensitive. Rev
        $this->encryption->initialize( // Sensitive.
            array(
                'cipher' => 'aes-256',
                'mode' => 'ctr',
                'key' => 'the key',
            )
        );
        $this->encryption->encrypt("mysecretdata"); // Sensi
    }
}
```

CraftCMS version 3

```
use Craft;

// This is similar to Yii as it used by CraftCMS
function craftEncrypt($data, $key, $password) {
    Craft::$app->security->encryptByKey($data, $key); // Sensitive
    Craft::$app->getSecurity()->encryptByKey($data, $key); // Sensitive
    Craft::$app->security->encryptByPassword($data, $password); // Sensitive
    Craft::$app->getSecurity()->encryptByPassword($data, $password); // Sensitive
}
```

Drupal 7 - Encrypt module

```
function drupalEncrypt() {
    $encrypted_text = encrypt('some string to encrypt'); // Sensitive
}
```

Joomla

```
use Joomla\Crypt\CipherInterface;

abstract class MyCipher implements CipherInterface // Sensitive
{

}

function joomlaEncrypt() {
    new Joomla\Crypt\Cipher_Sodium(); // Sensitive
    new Joomla\Crypt\Cipher_Simple(); // Sensitive
    new Joomla\Crypt\Cipher_Rijndael256(); // Sensitive
    new Joomla\Crypt\Cipher_Crypto(); // Sensitive
    new Joomla\Crypt\Cipher_Blowfish(); // Sensitive
    new Joomla\Crypt\Cipher_3DES(); // Sensitive
}
}
```

Laravel

```
use Illuminate\Support\Facades\Crypt;

function myLaravelEncrypt($data)
{
    Crypt::encryptString($data); // Sensitive
    Crypt::encrypt($data); // Sensitive
    // encrypt using the Laravel "encrypt" helper
    encrypt($data); // Sensitive
}
```

PHP-Encryption library

```
use Defuse\Crypto\Crypto;
use Defuse\Crypto\File;

function myPhpEncryption($data, $key, $password, $inputFile)
{
    Crypto::encrypt($data, $key); // Sensitive
    Crypto::encryptWithPassword($data, $password); // Sensitive
    File::encryptFile($inputFilename, $outputFilename, $key); // Sensitive
    File::encryptFileWithPassword($inputFilename, $outputFilename, $password); // Sensitive
    File::encryptResource($inputHandle, $outputHandle, $key); // Sensitive
    File::encryptResourceWithPassword($inputHandle, $outputHandle, $password); // Sensitive
}
```

PhpSecLib

```
function myphpseclib($mode) {
    new phpseclib\Crypt\RSA(); // Sensitive. Note: RSA can be used for encryption and decryption
    new phpseclib\Crypt\AES(); // Sensitive
    new phpseclib\Crypt\Rijndael(); // Sensitive
    new phpseclib\Crypt\Twofish(); // Sensitive
    new phpseclib\Crypt\Blowfish(); // Sensitive
    new phpseclib\Crypt\RC4(); // Sensitive
    new phpseclib\Crypt\RC2(); // Sensitive
    new phpseclib\Crypt\TripleDES(); // Sensitive
    new phpseclib\Crypt\DES(); // Sensitive

    new phpseclib\Crypt\AES($mode); // Sensitive
}
```

```

new phpseclib\Crypt\Rijndael($mode); // Sensitive
new phpseclib\Crypt\TripleDES($mode); // Sensitive
new phpseclib\Crypt\DES($mode); // Sensitive
}

```

Sodium Compat library

```

function mySodiumCompatEncrypt($data, $ad, $nonce, $key) {
    ParagonIE_Sodium_Compat::crypto_aead_chacha20poly1305_ie
    ParagonIE_Sodium_Compat::crypto_aead_xchacha20poly1305_i
    ParagonIE_Sodium_Compat::crypto_aead_chacha20poly1305_en

    ParagonIE_Sodium_Compat::crypto_aead_aes256gcm_encrypt($

    ParagonIE_Sodium_Compat::crypto_box($data, $nonce, $key)
    ParagonIE_Sodium_Compat::crypto_secretbox($data, $nonce,
    ParagonIE_Sodium_Compat::crypto_box_seal($data, $key); /
    ParagonIE_Sodium_Compat::crypto_secretbox_xchacha20poly1
}

```

Yii version 2

```

use Yii;

// Similar to CraftCMS as it uses Yii
function YiiEncrypt($data, $key, $password) {
    Yii::$app->security->encryptByKey($data, $key); // Sensi
    Yii::$app->getSecurity()->encryptByKey($data, $key); //
    Yii::$app->security->encryptByPassword($data, $password)
    Yii::$app->getSecurity()->encryptByPassword($data, $pass
}

```

Zend

```

use Zend\Crypt\FileCipher;
use Zend\Crypt\PublicKey\DiffieHellman;
use Zend\Crypt\PublicKey\Rsa;
use Zend\Crypt\Hybrid;
use Zend\Crypt\BlockCipher;

function myZendEncrypt($key, $data, $prime, $options, $gener
{
    new FileCipher; // Sensitive. This is used to encrypt fi

    new DiffieHellman($prime, $generator, $key); // Sensitiv

    $rsa = Rsa::factory([ // Sensitive
        'public_key' => 'public_key.pub',
        'private_key' => 'private_key.pem',
        'pass_phrase' => 'mypassphrase',
        'binary_output' => false,
    ]);
    $rsa->encrypt($data); // No issue raised here. The confi

    $hybrid = new Hybrid(); // Sensitive

    BlockCipher::factory($lib, $options); // Sensitive
}

```

See

- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [MITRE, CWE-321](#) - Use of Hard-coded Cryptographic Key
- [MITRE, CWE-322](#) - Key Exchange without Entity Authentication
- [MITRE, CWE-323](#) - Reusing a Nonce, Key Pair in Encryption
- [MITRE, CWE-324](#) - Use of a Key Past its Expiration Date
- [MITRE, CWE-325](#) - Missing Required Cryptographic Step
- [MITRE, CWE-326](#) - Inadequate Encryption Strength
- [MITRE, CWE-327](#) - Use of a Broken or Risky Cryptographic Algorithm
- [SANS Top 25](#) - Porous Defenses

Deprecated

This rule is deprecated; use {rule:php:S4426}, {rule:php:S5542}, {rule:php:S5547} instead.

Available In:



© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
[Privacy Policy](#)