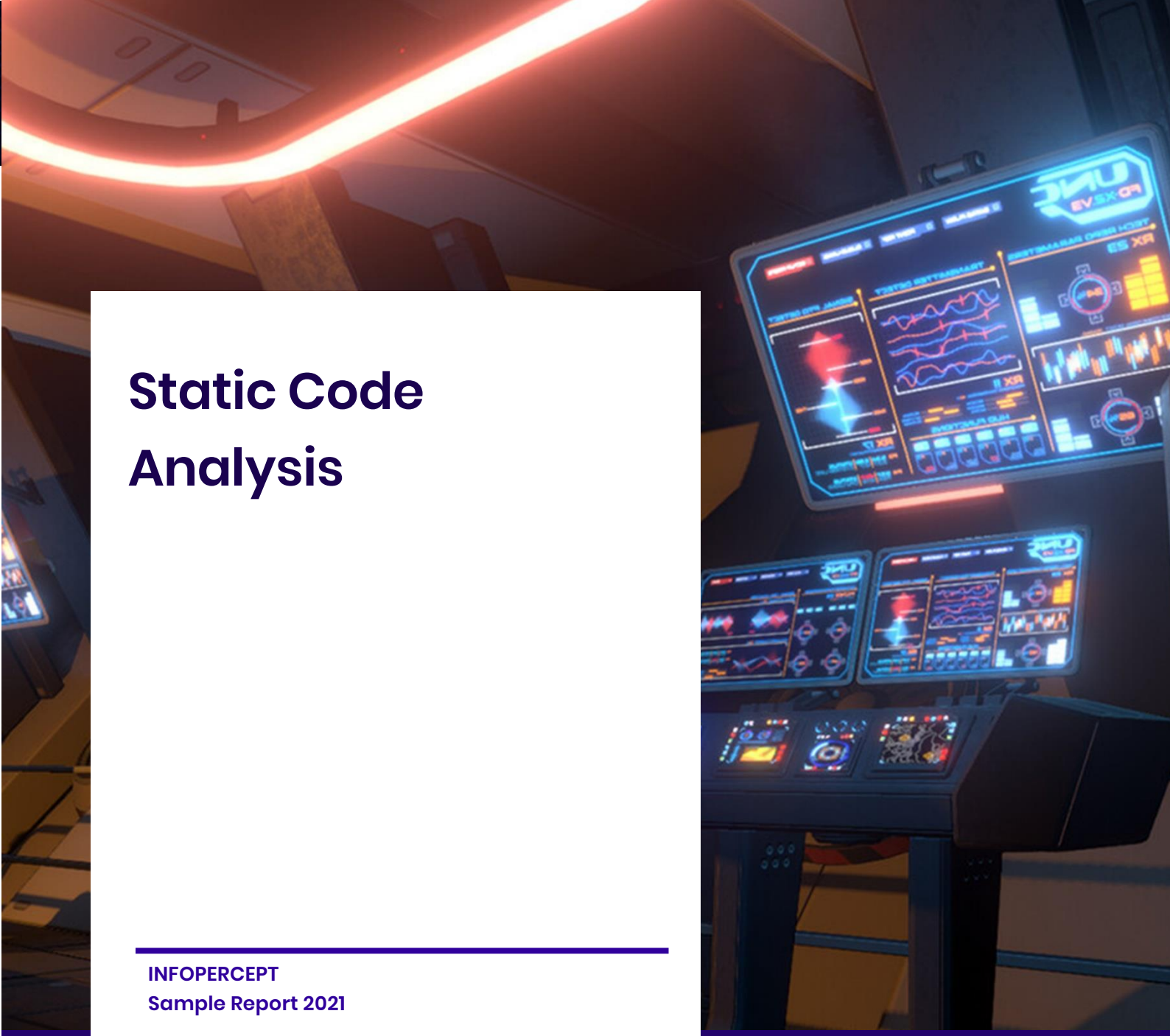


Static Code Analysis

INFOPERCEPT
Sample Report 2021

Infopercept

YOUR DATE HERE

COMPANY NAME
Authored by: Your Name

Contents

Copyright

Disclaimer

By accessing and using this report you agree to the following terms and conditions and all applicable laws, without limitation or qualification, unless otherwise stated, the contents of this document including, but not limited to, the text and images contained herein and their arrangement are the property of Infopercept Consulting Pvt Ltd (Infopercept). Nothing contained in this document shall be construed as conferring by implication, estoppel, or otherwise, any license or right to any copyright, patent, trademark or other proprietary interest of Infopercept or any third party. This document and its contents including, but not limited to, graphic images and documentation may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, without the prior written consent of Infopercept. Any use you make of the information provided, is at your own risk and liability. Infopercept makes no representation about the suitability, reliability, availability, timeliness, and accuracy of the information, products, services, and related graphics contained in this document. All such information products, services, related graphics and other contents are provided 'as is' without warranty of any kind. The relationship between you and Infopercept shall be governed by the laws of the Republic of India without regard to its conflict of law provisions. You and Infopercept agree to submit to the personal and exclusive jurisdiction of the courts located at Mumbai, India. You are responsible for complying with the laws of the jurisdiction and agree that you will not access or use the information in this report, in violation of such laws. You represent that you have the lawful right to submit such information and agree that you will not submit any information unless you are legally entitled to do so.

Introduction

This document contains results of the code analysis of Isep.

Configuration

Quality Profiles

- Names: Sonar way [CSS]; Sonar way [JavaScript]; Sonar way [PHP]; Sonar way [HTML]; Sonar way [XML];
- Files: AW_6552PAkiL_Dtivn7U.json; AW_656C0AkiL_DtivoEo.json; AW_656bAAkiL_Dtivofu.json; AW_656UkAkiL_DtivocD.json; AW_656XEAkiL_Dtivocb.json;

Quality Gate

- Name: Sonar way
- File: Sonar way.xml

Synthesis

Quality Gate	Reliability	Security	Maintainability	Coverage	Duplication
OK	E	E	A	0.0%	15.0%

Metrics

	Cyclomatic Complexity	Cognitive Complexity	Lines of code per file	Comment density (%)	Coverage	Duplication (%)
Min	0.0	0.0	0.0	0.0	0.0	0.0
Max	47286.0	73268.0	278901.0	92.3	XX-MAXCOVERAGE-XX	100.0

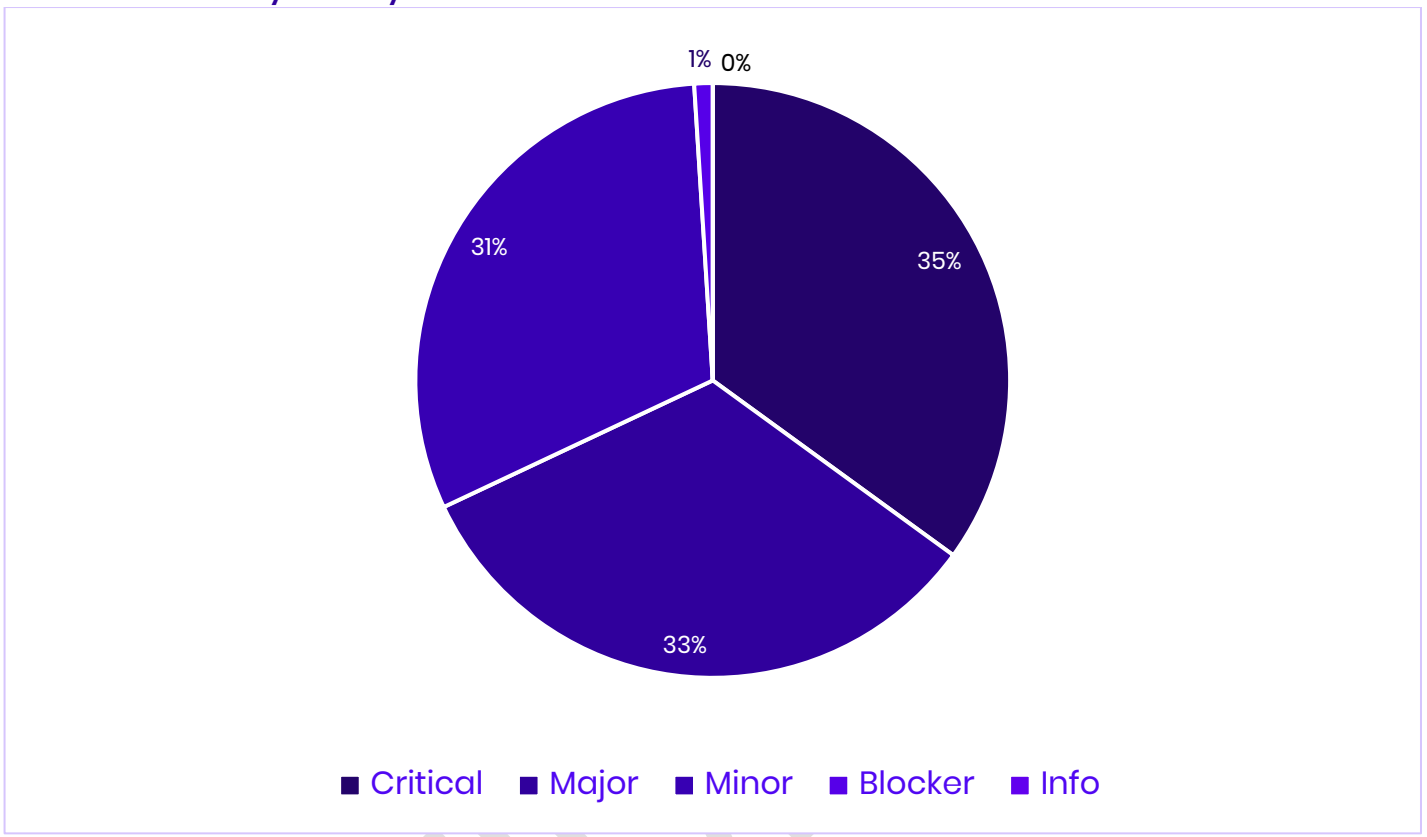
Volume

Language	Number
CSS	34458
JavaScript	75377
PHP	130554
HTML	40271
XML	398
Total	281058

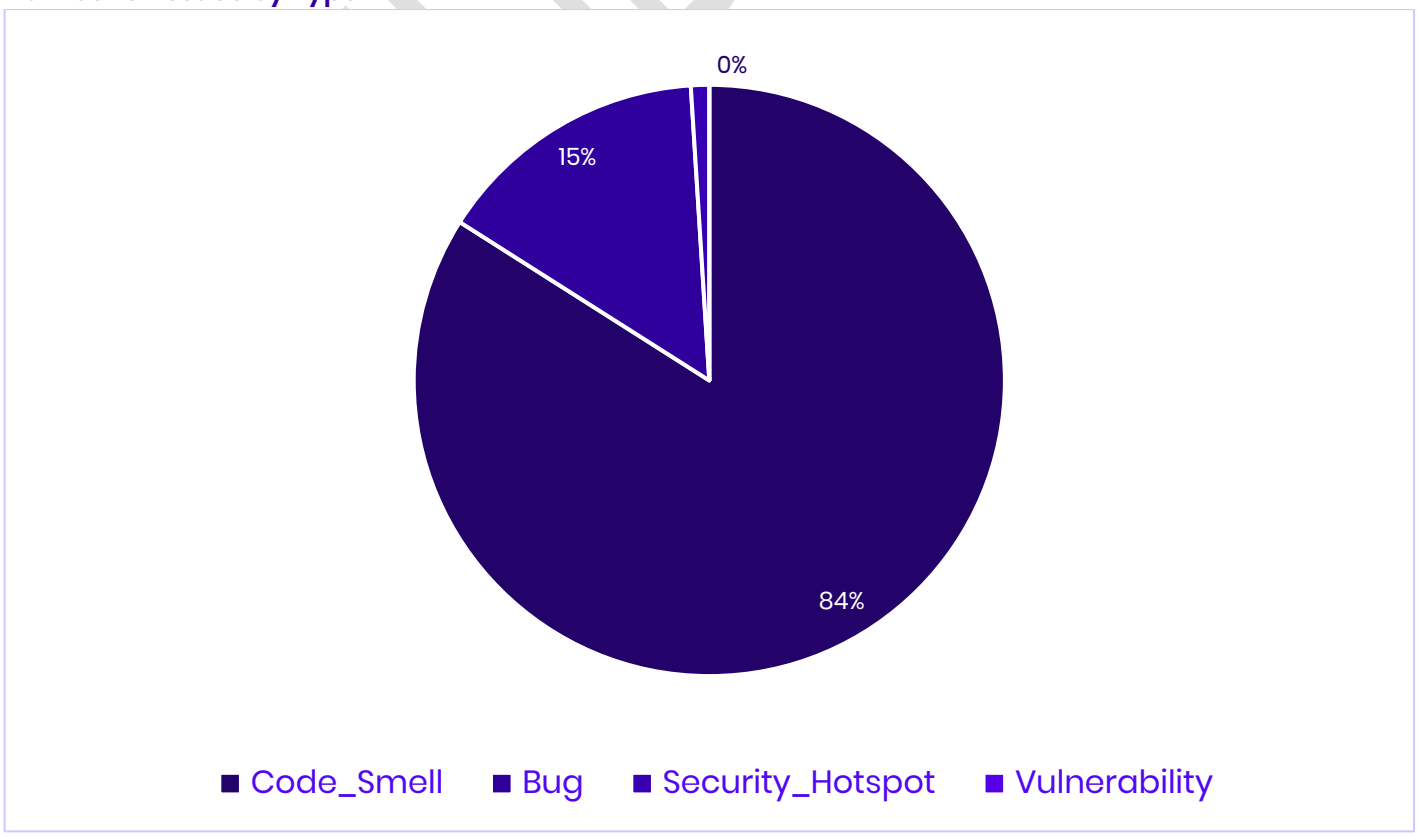
Issues Count by Severity and Type

Type	Severity	Number
Vulnerability	Blocker	4
Vulnerability	Critical	0
Vulnerability	Major	0
Vulnerability	Minor	1
Vulnerability	Info	0
Bug	Blocker	4
Bug	Critical	0
Bug	Major	151
Bug	Minor	123
Bug	Info	0
Code_Smell	Blocker	0
Code_Smell	Critical	5392
Code_Smell	Major	2224
Code_Smell	Minor	1930
Code_Smell	Info	40
Security_Hotspot	Blocker	0
Security_Hotspot	Critical	0
Security_Hotspot	Major	0
Security_Hotspot	Minor	0
Security_Hotspot	Info	0

Number of issues by severity



Number of issues by Type



Issues

Name	Description	Type	Severity	Number
"$this" should not be used in a static context	$this refers to the current class instance. But static methods can be accessed without instantiating the class, and $this is not available to them. Using $this in a static context will result in a fatal error at runtime. Noncompliant Code Example class Clazz { $name=NULL; // instance variable public static function foo(){ if ($this->name != NULL) { // ... } } } Compliant Solution class Clazz { $name=NULL; // instance variable public static function foo($nameParam){ if ($nameParam !=NULL) {// ... } } }	Bug	Blocker	4
"<!DOCTYPE>" declarations should appear before "<html>" tags	The <!DOCTYPE> declaration tells the web browser which (X)HTML version is being used on the page, and therefore how to interpret the various elements. Validators also rely on it to know which rules to enforce. It should always preceed the <html> tag. Noncompliant Code Example <html> <!-- Noncompliant --> ...</html> Compliant Solution <!DOCTYPE html> <html> <!-- Compliant --> ... </html>	Bug	Major	24
"<title>" should be present in all pages	Titles are important because they are displayed in search engine results as well as the browser's toolbar. This rule verifies that the <head> tag contains a <title> one, and the <html> tag a <head> one. Noncompliant Code Example <html> <!-- Non-Compliant --> <body> ... </body> </html> Compliant Solution <html> <!-- Compliant --> <head> <title>Some relevant title</title> </head> <body> ... </body> </html>	Bug	Major	1
Elements deprecated in HTML5 should not be used	With the advent of HTML5, many old elements were deprecated. To ensure the best user experience, deprecated elements should not be used. This rule checks for the following deprecated elements:Element Remediation Action basefont, big, blink, center, font, marquee, multicol, nobr, spacer, tt use CSS acronym use abbr applet use embed or object bgsound use audio frame, frameset, noframes restructure the page to remove frames isindex use form controls dir	Bug	Major	14

	use ul hgroup use header or div listing use pre and code nextid use GUIDS noembed use object instead of embed when fallback is necessary plaintext use the "text/plain" MIME type strike use del or s xmp use pre or code, and escape "<" and "&" characters See W3C, Obsolete Features WHATWG, Obsolete Features			
Variables should not be self-assigned	some other value or variable was intended for the assignment instead. Noncompliant Code Example public function setName($name) {$name = $name; } Compliant Solution public function setName($name) {$this->name = $name; } See CERT, MSC12-C. - Detect and remove code that has no effect or is never executed	Bug	Major	2
Jump statements should not be followed by dead code	Jump statements (return, break, continue, goto) and throw expressions move control flow out of the current code block. So any unlabelled statements that come after a jump are dead code. Noncompliant Code Example function fun($a) { $i = 10; return $i + $a; $i++; // dead code } Compliant Solution function fun($a) { $i = 10; return $i + $a; } See MITRE, CWE-561 - Dead Code CERT, MSC56-J. - Detect and remove superfluous code and values CERT, MSC12-C. - Detect and remove code that has no effect or is never executed	Bug	Major	4
Identical expressions should not be used on both sides of a binary operator	Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste error and therefore a bug, or it is simply wasted code, and should be simplified. In the case of bitwise operators and most binary mathematical operators, having the same value on both sides of an operator yields predictable results, and should be simplified. Noncompliant Code Example if ($a == $a) { // always true doZ(); } if ($a != $a) { // always false doY(); } if ($a == $b && $a == $b) { // if the first one is true, the second one is too doX(); } if ($a == $b || $a == $b) { // if the first one is true, the second one is too doW(); } $j = 5 / 5; //always 1 $k = 5 - 5; //always 0 Exceptions Left-shifting 1 onto 1 is common in the construction of bit masks, and is ignored. $i = 1 << 1; // Compliant $j = $a << $a; // Noncompliant See CERT, MSC12-C. - Detect and remove code that has no effect or is never executed S1656 - Implements a check on =.	Bug	Major	8
Related "if/else if" statements and "cases" in a "switch" should not have the same condition	A switch and a chain of if/else if statements is evaluated from top to bottom. At most, only one branch will be executed: the first one with a condition that evaluates to true. Therefore, duplicating a condition automatically leads to dead code. Usually, this is due to a copy/paste	Bug	Major	31

	error. At best, it's simply dead code and at worst, it's a bug that is likely to induce further bugs as the code is maintained, and obviously it could lead to unexpected behavior. For a switch, if the first case ends with a break, the second case will never be executed, rendering it dead code. Worse there is the risk in this situation that future maintenance will be done on the dead case, rather than on the one that's actually used. On the other hand, if the first case does not end with a break, both cases will be executed, but future maintainers may not notice that. Noncompliant Code Example if ($param == 1) open Window(); else if ($param == 2) close Window(); else if ($param == 1) // Noncompliant move Window to the Background(); switch($i) { case 1: //... break; case 3: //... break; case 1: // Noncompliant //... break; default: // ... break; } Compliant Solution if ($param == 1) openWindow(); else if ($param == 2) closeWindow(); else if ($param == 3) moveWindowToTheBackground(); switch($i) { case 1: //... break; case 3: //... break; default:// ...break; } See CERT, MSC12-C. - Detect and remove code that has no effect or is never executed			
Return values from functions without side effects should not be ignored	When the call to a function doesn't have any side effect, what is the point of making the call if the results are ignored? In such cases, either the function call is useless and should be dropped, or the source code doesn't behave as expected. Noncompliant Code Example strlen($name); // Noncompliant; "strlen" has no side effect Compliant Solution $length = strlen($name); See CERT, EXP12-C. - Do not ignore values returned by functions CERT, EXP00-J. - Do not ignore values returned by methods	Bug	Major	1
All branches in a conditional structure should not have exactly the same implementation	Having all branches in a switch or if chain with the same implementation is an error. Either a copy- paste error was made and something different should be executed, or there shouldn't be a switch/if chain at all. Noncompliant Code Example if ($b == 0) { // Noncompliant doOneMoreThing(); } else { doOneMoreThing(); } $b = $a > 12 ? 4 : 4; // Noncompliant switch ($i) { // Noncompliant case 1:doSomething(); break; case 2: doSomething(); break; case 3: doSomething(); break; default: doSomething(); } Exceptions This rule does not apply to if chains without else-s, or to switch-es without default clauses. if($b == 0) { //no issue, this could have been done on purpose to make the code more readable doSomething(); } elseif($b == 1) { doSomething(); }	Bug	Major	20

Variables should be initialized before use	When a variable is not initialized before its use, it's a sign that the developer made a mistake. Noncompliant Code Example function fun($condition) { $res = 1; if ($condition) {$res++; } return $result; // Noncompliant, "$result" instead of "$res" } Compliant Solution function fun($condition) { $res = 1; if ($condition) {$res++; } return $res; } See MITRE, CWE-457 - Use of Uninitialized Variable	Bug	Major	43
Non-empty statements should change control flow or have at least one side-effect	Any statement (other than a null statement, which means a statement containing only a semicolon ;) which has no side effect and does not result in a change of control flow will normally indicate a programming error, and therefore should be refactored. Noncompliant Code Example $a == 1; // Noncompliant; was assignment intended? $a <$b; // Noncompliant; have we forgotten to assign the result to a variable? {code} See MITRE, CWE- 482 - Comparing instead of Assigning	Bug	Major	3
"" and "" tags should be used	The <strong>/ and /<i> tags have exactly the same effect in most web browsers, but there is a fundamental difference between them: <strong> and have a semantic meaning whereas and <i> only convey styling information like CSS. While can have simply no effect on a some devices with limited display or when a screen reader software is used by a blind person, <strong> will: Display the text bold in normal browsers Speak with lower tone when using a screen reader such as Jaws Consequently: in order to convey semantics, the and <i> tags shall never be used, in order to convey styling information, the and <i> should be avoided and CSS should be used instead. Noncompliant Code Example <i>car</i> <!-- Noncompliant --> train <!-- Noncompliant --> Compliant Solution car	Bug	Minor	108
"<frames>" should have a "title" attribute	Frames allow different web pages to be put together on the same visual space. Users without disabilities can easily scan the contents of all frames at once. However, visually impaired users using screen readers hear the page content linearly. The title attribute is used to list all the page's frames, enabling those users to easily navigate among them. Therefore, the <frame> and <iframe> tags should always have a title attribute. Noncompliant Code Example	Bug	Minor	6

	<frame src="index.php?p=menu"> <-- Non-Compliant --> <frame src="index.php?p=home" name="contents"> <-- Non-Compliant --> Compliant Solution <frame src="index.php?p=menu" title="Navigation menu"> <-- Compliant --> <frame src="index.php?p=home" title="Main content" name="contents"> <-- Compliant -->			
Images tags and buttons should have an "alt" attribute	The alt attribute provides a textual alternative to an image. It is used whenever the actual image cannot be rendered. Common reasons for that include: The image can no longer be found Visually impaired users using a screen reader software Images loading is disabled, to reduce data consumption on mobile phones Empty alt attributes are not allowed, since purely decorative images should be specified using CSS, not using the img tag. Noncompliant Code Example <!-- Noncompliant --> <!-- Noncompliant -- > <input type="image" src="bar.png" /> <!-- Noncompliant --> Compliant Solution <input type="image" src="bar.png" alt="Textual description of bar.png" />	Bug	Minor	9
Constant names should comply with a naming convention	Shared coding conventions allow teams to collaborate efficiently. This rule checks that all constant names match a provided regular expression. Noncompliant Code Example With the default regular expression ^[A-Z][A-Z0-9]*(_[A-Z0- 9]+)*$: define("const1", true); class Foo {const const2 = "bar";} Compliant Solution define("CONST1", true); class Foo {const CONST2 = "bar"; }	CODE_SMELL	Critical	116
String literals should not be duplicated	Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences. On the other hand, constants can be referenced from many places, but only need to be updated in a single place. Noncompliant Code Example With the default threshold of 3: function run() { prepare('action1'); // Non-Compliant - 'action1' is duplicated 3 times execute('action1'); release('action1'); } Compliant Solution ACTION_1 = 'action1'; function run() { prepare(ACTION_1); execute(ACTION_1); release(ACTION_1); } Exceptions To prevent generating some false-positives, literals having less than 5 characters are excluded.	CODE_SMELL	Critical	2604
Control structures should use curly braces	While not technically incorrect, the omission of curly braces can be misleading, and may lead to the introduction of errors during maintenance. Noncompliant Code Example if (condition) // Noncompliant	CODE_SMELL	Critical	1955

	executeSomething(); Compliant Solution if (condition) {executeSomething(); } See CERT, EXP19-C. - Use braces for the body of an if, for, or while statement CERT, EXP52-J. - Use braces for the body of an if, for, or while statement			
switch" statements should have "default" clauses	The requirement for a final case default clause is defensive programming. The clause should either take appropriate action, or contain a suitable comment as to why no action is taken. Even when the switch covers all current values of an enum, a default case should still be used because there is no guarantee that the enum won't be extended. Noncompliant Code Example switch ($param) {//missing default clause case 0: do_something(); break; case 1: do_something_else(); break; } Compliant Solution switch ($param) { case 0: do_something(); break; case 1: do_something_else(); break; default: error(); break; } See MITRE, CWE-478 - Missing Default Case in Switch Statement CERT, MSC01-C. - Strive for logical completeness	CODE_SMELL	Critical	151
Parentheses should not be used for calls to "echo"	echo can be called with or without parentheses, but it is best practice to leave parentheses off the call because using parentheses with multiple arguments will result in a parse error. Noncompliant Code Example echo("Hello"); // Noncompliant, but it works echo("Hello", "World"); // Noncompliant. Parse error Compliant Solution echo "Hello"; echo "Hello","World!";	CODE_SMELL	Critical	20
Cognitive Complexity of functions should not be too high	Cognitive Complexity is a measure of how hard the control flow of a function is to understand. Functions with high Cognitive Complexity will be difficult to maintain. See Cognitive Complexity	CODE_SMELL	Critical	513
A conditionally executed single line should be denoted by indentation	In the absence of enclosing curly braces, the line immediately after a conditional is the one that is conditionally executed. By both convention and good practice, such lines are indented. In the absence of both curly braces and indentation the intent of the original programmer is entirely unclear and perhaps not actually what is executed. Additionally, such code is highly likely to be confusing to maintainers. Noncompliant Code Example if ($x > 0) // Noncompliant doTheThing(); doTheOtherThing(); foo();	CODE_SMELL	Critical	33

	Compliant Solution if ($x > 0) { doTheThing(); doTheOtherThing(); } foo(); or if ($x > 0) doTheThing(); doTheOtherThing(); foo();			
Track uses of "TODO" tags	TODO tags are commonly used to mark places where some more code is required, but which the developer wants to implement later. Sometimes the developer will not have the time or will simply forget to get back to that tag. This rule is meant to track those tags and to ensure that they do not go unnoticed. Noncompliant Code Example function doSomething() { // TODO } See MITRE, CWE-546 - Suspicious Comment	CODE_SMELL	Info	40
Sections of code should not be commented out	Programmers should not comment out code as it should not be bloats programs and reduces readability. Unused commented out code should be deleted and can be retrieved from source control history if required. See MISRA :2004, 2.4 - Sections of code should not be "commented out". MISRA C++:2008, 2-7-2 - Sections of code shall not be "commented out" using C-style comments. MISRA C++:2008, 2-7-3 - Sections of code should not be "commented out" using C++ comments. MISRA C:2012, Dir. 4.4 - Sections of code should not be "commented out"	CODE_SMELL	Major	42
Source files should not have any duplicated blocks	An issue is created on a file as soon as there is at least one block of duplicated code on this file	CODE_SMELL	Major	84
Collapsible "if" statements should be merged	Merging collapsible if statements increases the code's readability. Noncompliant Code Example if (condition1) { if (condition2) { ... } } Compliant Solution if (condition1 && condition2) { ... }	CODE_SMELL	Major	107
Unused "private" fields should be removed	If a private field is declared but not used in the program, it can be considered dead code and should therefore be removed. This will improve maintainability because developers will not wonder what the variable is used for. Noncompliant Code Example class MyClass { private $foo = 4; //foo is unused public function compute($a) { return $a * 4; } } Compliant Solution class MyClass { public function compute($a) { return $a * 4; } }	CODE_SMELL	Major	6
Functions should not have too	A long parameter list can indicate that a new structure should be created to wrap the numerous parameters or that the function is	CODE_SMELL	Major	20

many parameters	doing too many things. Noncompliant Code Example With a maximum number of 4 parameters: function doSomething($param1, $param2, $param3, $param4, $param5) { ... } Compliant Solution function doSomething($param1, $param2, $param3, $param4) { ... }			
Nested blocks of code should not be left empty	Most of the time a block of code is empty when a piece of code is really missing. So such empty block must be either filled or removed. Noncompliant Code Example for ($i = 0; $i < 42; $i++){} // Empty on purpose or missing piece of code ? Exceptions When a block contains a comment, this block is not considered to be empty.	CODE_SMELL	Major	23
Redundant pairs of parentheses should be removed	The use of parentheses, even those not required to enforce a desired order of operations, can clarify the intent behind a piece of code. But redundant pairs of parentheses could be misleading, and should be removed. Noncompliant Code Example $x = ($y / 2 + 1); // Compliant even if the parenthesis are ignored by the compiler if ($a && (($x + $y > 0))) { // Noncompliant //... } return (($x + 1)); // Noncompliant Compliant Solution $x = ($y / 2 + 1); if ($a && ($x + $y > 0)) { //... } return ($x + 1);	CODE_SMELL	Major	9
Local variables should not have the same name as class fields	Overriding or shadowing a variable declared in an outer scope can strongly impact the readability, and therefore the maintainability, of a piece of code. Further, it could lead maintainers to introduce bugs because they think they're using one variable but are really using another. Noncompliant Code Example class Foo { public $myField; public function doSomething() { $myField = 0; ... } } See CERT, DCL01-C. - Do not reuse variable names in subscopes CERT, DCL51-J. - Do not shadow or obscure identifiers in subscopes	CODE_SMELL	Major	29
Track uses of "FIXME" tags	FIXME tags are commonly used to mark places where a bug is suspected, but which the developer	CODE_SMELL	Major	10

	wants to deal with later. Sometimes the developer will not have the time or will simply forget to get back to that tag. This rule is meant to track those tags and to ensure that they do not go unnoticed. Noncompliant Code Example function divide($numerator, $denominator) { return $numerator / $denominator; // FIXME denominator value might be 0 } See MITRE, CWE-546 - Suspicious Comment			
Functions should not contain too many return statements	Having too many return statements in a function increases the function's essential complexity because the flow of execution is broken each time a return statement is encountered. This makes it harder to read and understand the logic of the function. Noncompliant Code Example With the default threshold of 3: function myFunction(){ // Noncompliant as there are 4 return statements if (condition1) {return true; } else {if condition2) {return false;} else {return true;} } return false; }	CODE_SMELL	Major	196
Unused "private" methods should be removed	private methods that are never executed are dead code: unnecessary, inoperative code that should be removed. Cleaning out dead code decreases the size of the maintained codebase, making it easier to understand the program and preventing bugs from being introduced. Noncompliant Code Example public class Foo { private function Foo() {} // Compliant, private empty constructor intentionally used to prevent any direct instantiation of a class. public static function doSomething() { $foo = new Foo(); ... } private function unusedPrivateFunction() { // Noncompliant } } Compliant Solution public class Foo { private function Foo(){} // Compliant, private empty constructor intentionally used to prevent any direct instantiation of a class. public static function doSomething() { $foo = new Foo(); } }	CODE_SMELL	Major	15
Unused function parameters should be removed	Unused parameters are misleading. Whatever the value passed to such parameters is, the behavior will be the same. Noncompliant Code Example function doSomething($a, $b) { // "$a" is unused return compute($b); } Compliant Solution function doSomething($b) { return compute($b); } Exceptions Functions in classes that override a class or implement interfaces are ignored. class C extends B { function doSomething($a, $b) {// no issue reported on $b compute($a); } } See CERT, MSC12-C. - Detect and remove code that has no effect or is never executed	CODE_SMELL	Major	38

Sections of code should not be commented out	Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required	CODE_SMELL	Major	1241
"for" loop stop conditions should be invariant	A for loop stop condition should test the loop counter against an invariant value (i.e. one that is true at both the beginning and ending of every loop iteration). Ideally, this means that the stop condition is set to a local variable just before the loop begins. Stop conditions that are not invariant are slightly less efficient, as well as being difficult to understand and maintain, and likely lead to the introduction of errors in the future. This rule tracks three types of non-invariant stop conditions: When the loop counters are updated in the body of the for loop When the stop condition depend upon a method call When the stop condition depends on an object property, since such properties could change during the execution of the loop. Noncompliant Code Example for ($i = 0; $i < 10; $i++) {echo $i; if(condition) {$i = 20; } } Compliant Solution for ($i = 0; $i < 10; $i++) { echo $i; }	CODE_SMELL	Major	13
Functions should not have too many lines of code	A function that grows too large tends to aggregate too many responsibilities. Such functions inevitably become harder to understand and therefore harder to maintain. Above a specific threshold, it is strongly advised to refactor into smaller functions which focus on well-defined tasks. Those smaller functions will not only be easier to understand, but also probably easier to test.	CODE_SMELL	Major	73
Classes should not have too many methods	A class that grows too much tends to aggregate too many responsibilities and inevitably becomes harder to understand and therefore to maintain. Above a specific threshold, it is strongly advised to refactor the class into smaller ones which focus on well-defined topics.	CODE_SMELL	Major	62
"switch" statements should not have too many "case" clauses	When switch statements have large sets of case clauses, it is usually an attempt to map two sets of data. A real map structure would be more readable and maintainable, and should be used instead.	CODE_SMELL	Major	13
PHP 4 constructor declarations should not be used	In PHP 4, any function with the same name as the nesting class was considered a class constructor. In PHP 5, this mechanism has been deprecated and the " construct" method name should be used instead. If both styles are present in the same class, PHP 5 will treat the function named " construct" as the class constructor. This rule rule raises an issue for each method with the same name as the	CODE_SMELL	Major	2

	enclosing class. Noncompliant Code Example class Foo { function Foo(){...} } Compliant Solution class Foo { function construct(){...} }			
Method arguments with default values should be last	The ability to define default values for method arguments can make a method easier to use. Default argument values allow callers to specify as many or as few arguments as they want while getting the same functionality and minimizing boilerplate, wrapper code. But all method arguments with default values should be declared after the method arguments without default values. Otherwise, it makes it impossible for callers to take advantage of defaults; they must re-specify the defaulted values in order to "get to" the non- default arguments. Noncompliant Code Example function makeyogurt($type = "acidophilus", $flavor){...} // Noncompliant makeyogurt("raspberry")}} // Runtime error: Missing argument 2 in call to makeyogurt() Compliant Solution function makeyogurt($flavor, $type = "acidophilus",){...} makeyogurt("raspberry")}} // Works as expected	CODE_SMELL	Major	24
Dead stores should be removed	A dead store happens when a local variable is assigned a value that is not read by any subsequent instruction. Calculating or retrieving a value only to then overwrite it or throw it away, could indicate a serious error in the code. Even if it's not an error, it is at best a waste of resources. Therefore all calculated values should be used. Noncompliant Code Example $i = $a + $b; // Noncompliant; calculation result not used before value is overwritten $i = compute(); Compliant Solution $i = $a + $b; $i += compute(); Exceptions This rule ignores initializations to -1, 0, 1, null, true, false, "", [] and array(). See MITRE, CWE-563 - Assignment to Variable without Use ('Unused Variable') CERT, MSC13-C. - Detect and remove unused values CERT, MSC56-J. - Detect and remove superfluous code and values	CODE_SMELL	Major	84
Two branches in a conditional structure should not have exactly the same implementation	Having two cases in a switch statement or two branches in an if chain with the same implementation is at best duplicate code, and at worst a coding error. If the same logic is truly needed for both instances, then in an if chain they should be combined, or for a switch, one should fall through to the other. Noncompliant Code Example switch ($i) { case 1: doFirst(); doSomething(); break; case 2: doSomethingDifferent(); break; case 3: // Noncompliant; duplicates case 1's mplementation doFirst(); doSomething(); break; default: doTheRest(); } if ($a >= 0&& $a < 10) { doFirst();	CODE_SMELL	Major	34

	doTheThing(); } else if ($a >= 10 && $a < 20) { doTheOtherThing(); } else if ($a >= 20 && $a < 50) { doFirst(); doTheThing(); // Noncompliant; duplicates first condition } Exceptions Blocks in an if chain that contain a single line of code are ignored, as are blocks in a switch statement that contain a single line of code with or without a following break. if ($a >= 0 && $a < 10) { doTheThing(); } else if ($a >= 10 && $a < 20) { doTheOtherThing(); } else if ($a >= 20 && $a < 50) { doTheThing(); // no issue, usually this is done on purpose to increase the readability } But this exception does not apply to if chains without else-s, or to switch-es without default clauses when all branches have the same single line of code. In case of if chains with else-s, or of switch-es with default clauses, rule S3923 raises a bug. if ($a >= 0 && $a < 10) { doTheThing(); } else if ($a >= 20 && $a < 50) { doTheThing(); //Noncompliant; this might have been done on purpose but probably not }			
Ternary operators should not be nested	Just because you can do something, doesn't mean you should, and that's the case with nested ternary operations. Nesting ternary operators results in the kind of code that may seem clear as day when you write it, but six months later will leave maintainers (or worse - future you) scratching their heads and cursing. Instead, err on the side of clarity, and use another line to express the nested operation as a separate statement. Noncompliant Code Example function get_title($gender, $is_married) { return $gender == "MALE" ? "Mr. " : ($is_married ? "Mrs. " : "Miss "); // Noncompliant } Compliant Solution function get_title($gender, $is_married) { if ($gender == "MALE") { return "Mr. "; } return $is_married ? "Mrs. " : "Miss "; }	CODE_SMELL	Major	13
Functions should use "return" consistently	Because it is dynamically typed, PHP does not enforce a return type on a function. This means that different paths through a function can return different types of values, which can be very confusing to the user and significantly harder to maintain. In particular, it is consequently	CODE_SMELL	Major	4

	also possible to mix empty return statements (implicitly returning null) with some returning an expression. This rule verifies that all the return statements from a function are consistent. Noncompliant Code Example function foo($a) { // Noncompliant, function will return "true" or null if ($a == 1) { return true; } return; } Compliant Solution function foo($a) { if ($a == 1) { return true; } return false; } or function foo($a) { if ($a == 1) { return true; } return null; }			
Methods should not have identical implementations	When two methods have the same implementation, either it was a mistake - something else was intended - or the duplication was intentional, but may be confusing to maintainers. In the latter case, one implementation should invoke the other. Noncompliant Code Example class A { private const CODE = "bounteous"; public function getCode() { doTheThing(); return A::CODE;} public function getName() { // Noncompliant doTheThing(); return A::CODE;} } Compliant Solution class A {private const CODE = "bounteous"; public function getCode() { doTheThing(); return A::CODE; } public function getName() { return $this- >getCode(); } } Exceptions Methods that are not accessors (getters and setters), with fewer than 2 statements are ignored.	CODE_SMELL	Major	11
Class names should comply with a naming convention	Shared coding conventions allow teams to collaborate effectively. This rule allows to check that all class names match a provided regular expression. Noncompliant Code Example With default provided regular expression ^[A-Z][a-zA-Z0- 9]*$: class my_class {...} Compliant Solution class MyClass {...}	CODE_SMELL	Minor	242
A close curly brace should be located at the beginning of a line	Shared coding conventions make it possible for a team to efficiently collaborate. This rule makes it mandatory to place a close curly brace at the beginning of a line. Noncompliant Code Example if(condition) { doSomething();} Compliant Solution if(condition) { doSomething(); } Exceptions When blocks are inlined (open and close curly braces on the same line), no issue is triggered. if(condition) {doSomething();}	CODE_SMELL	Minor	20
Empty statements should be removed	Empty statements, i.e. ;, are usually introduced by mistake, for example because:It was meant to be replaced by an actual statement, but this was forgotten. There was a typo which lead the semicolon to be doubled, i.e. ;;. Noncompliant Code Example function	CODE_SMELL	Minor	6

	doSomething() { ; // oncompliant - was used as a kind of TODO marker } function doSomethingElse($p) { echo $p;;// Noncompliant - double ; } for ($i = 1; $i <= 10; doSomething($i), $i++); // Noncompliant - Rarely, they are used on purpose as the body of a loop. It is a bad practice to have side-effects outside of the loop body Compliant Solution function doSomething() {} function doSomethingElse($p) { echo $p; for ($i = 1; $i <= 10; $i++) { oSomething($i); } } See CERT, MSC12-C. - Detect and remove code that has no effect or is never executed CERT, MSC51-J. - Do not place a semicolon immediately following an if, for, or while condition CERT, EXP15-C. - Do not place a semicolon on the same line as an if, for, or while statement			
Boolean literals should not be redundant	Redundant Boolean literals should be removed from expressions to improve readability. Noncompliant Code Example if ($booleanVariable == true) { /* ... */ } if ($booleanVariable != true) { /* ... */ } if ($booleanVariable || false) { /* ... */ } doSomething(!false); $booleanVariable = condition ? true : exp; $booleanVariable = condition ? false : exp; $booleanVariable = condition ? exp : true; $booleanVariable = condition ? exp : false; Compliant Solution if ($booleanVariable) { /* ... */ } if (!$booleanVariable) { /* ... */ } if ($booleanVariable) { /* ... */ } doSomething(true); $booleanVariable = condition || exp; $booleanVariable = !condition && exp; $booleanVariable = !condition || exp; $booleanVariable = condition && exp; Exceptions The use of literal booleans in comparisons which use identity operators (=== and !==) are ignored.	CODE_SMELL	Minor	118
Return of boolean expressions should not be wrapped into an "if-then-else" statement	Return of boolean literal statements wrapped into if-then-else ones should be simplified. Noncompliant Code Example if (expression) { return true; } else { return false; } Compliant Solution return expression;	CODE_SMELL	Minor	13
Interface names should comply with a naming convention	Sharing some naming conventions is a key point to make it possible for a team to efficiently collaborate. This rule allows to check that all interface names match a provided regular expression. Noncompliant Code Example With the default regular expression ^[A-Z][a-zA-Z0-9]*$: interface myInterface {...} // Noncompliant Compliant Solution interface MyInterface {...}	CODE_SMELL	Minor	7

Overriding methods should do more than simply call the same method in the super class	Overriding a method just to call the same method from the super class without performing any other actions is useless and misleading. The only time this is justified is in final overriding methods, where the effect is to lock in the parent class behavior. This rule ignores such overrides of equals, hashCode and toString. Noncompliant Code Example class Child extends Parent { public function func($n,$m) { parent::func(nm); // Noncompliant } } class Parent { public function func($n, $m) { // do something } } Compliant Solution class Child extends Parent { public function func($n,$m) { parent::func(nm); // do additional things... } } class Parent { public function func($n, $m) { // do something } } or class Child extends Parent { // function eliminated } class Parent { public function func($n, $m) { // do something } }	CODE_SMELL	Minor	17
"switch" statements should have at least 3 "case" clauses	switch statements are useful when there are many different cases depending on the value of the same expression. For just one or two cases however, the code will be more readable with if statements. Noncompliant Code Example switch ($variable) { case 0: do_something(); break; default: do_something_else(); break; } Compliant Solution if ($variable == 0) { do_something(); } else { do_something_else(); }	CODE_SMELL	Minor	36
Unused local variables should be removed	If a local variable is declared but not used, it is dead code and should be removed. Doing so will improve maintainability because developers will not wonder what the variable is used for. Noncompliant Code Example function numberOfMinutes($hours) { $seconds = 0; // seconds is never used return hours * 60; } Compliant Solution function numberOfMinutes($hours) { return hours * 60; }	CODE_SMELL	Minor	1184
Local variables should not be declared and then immediately returned or thrown	Declaring a variable only to immediately return or throw it is a bad practice. Some developers argue that the practice improves code readability, because it enables them to explicitly name what is being returned. However, this variable is an internal implementation detail that is not exposed to the callers of the method. The method name should be sufficient for callers to know exactly what will be returned. Noncompliant Code Example function computeDurationInMilliseconds() {$duration = (((($hours * 60) + $minutes) * 60 + $seconds) * 1000 ; return $duration; } Compliant Solution function computeDurationInMilliseconds() { return	CODE_SMELL	Minor	163

	((($hours * 60) + $minutes) * 60 + $seconds) * 1000; }			
"&&" and "||" should be used	PHP has two sets of logical operators: && / ||, and and / or. The difference between the sets is precedence. Because and / or have a lower precedence than almost any other operator, using them instead of && / || may not have the result you expect. Noncompliant Code Example $have_time = true; $have_money = false; $take_vacation = $have_time and $have_money; // Noncompliant. $take_vacation == true. Compliant Solution $have_time = true; $have_money = false; $take_vacation = $have_time && $have_money; // $take_vacation == false.	CODE_SMELL	Minor	89
Jump statements should not be redundant	Jump statements, such as return, goto, and continue let you change the default flow of program execution, but jump statements that direct the control flow to the original direction are just a waste of keystrokes. Noncompliant Code Example function foo($p) { $i = $p; while ($i > 0) { $i--; continue; // Noncompliant } } Compliant Solution function foo($p) { $i = $p; while ($i > 0) { $i--; } }	CODE_SMELL	Minor	35
Using pseudorandom number generators (PRNGs) is security-sensitiv	Using pseudorandom number generators (PRNGs) is security-sensitive. For example, it has led in the past to the following vulnerabilities: CVE-2013- 6386 CVE-2006-3419 CVE-2008-4102 When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information. As the rand() and mt_rand functions rely on a pseudorandom number generator, it should not be used for security-critical applications or for protecting sensitive data. Ask Yourself Whether the code using the generated value requires it to be unpredictable. It is the case for all encryption mechanisms or when a secret value, such as a password, is hashed. the function you use generates a value which can be predicted (pseudo-random). the generated value is used multiple times. an attacker can access the generated value. You are at risk if you answered yes to the first question and any of the following ones. Recommended Secure Coding Practices Use functions which rely on a cryptographically strong random number generator such as random_int() or random_bytes() or openssl_random_pseudo_bytes() When using openssl_random_pseudo_bytes(),	Security_Hotspot	Critical	23

	provide and check the crypto_strong parameter Use the generated random values only once. You should not expose the generated random value. If you have to store it, make sure that the database or file is secure. Questionable Code Example $random = rand(); $random2 = mt_rand(0, 99); Compliant			
Hashing data is security-sensitive	Hashing data is security-sensitive. It has led in the past to the following vulnerabilities: CVE-2018- 9233 CVE-2013-5097 CVE-2007-1051 Cryptographic hash functions are used to uniquely identify information without storing their original form. When not done properly, an attacker can steal the original information by guessing it (ex: with a rainbow table), or replace the original data with another one having the same hash. This rule creates an issue when one of the following functions are called: hash, hash_init, crypt, password_hash, hash_pbkdf2, openssl_pbkdf2, md5, sha1 Ask Yourself Whether the hashed value is used in a security context. the hashing algorithm you are using is known to have vulnerabilities. salts are not automatically generated and applied by the hashing function. any generated salts are cryptographically weak or not credential-specific. You are at risk if you answered yes to the first question and any of the following ones. Recommended Secure Coding Practices If the hashed data is sensitive, just use the functions officially recommended by PHP, i.e. password_hash, password_verify and password_needs_rehash. Alternatively you can use the crypt function or hash_pbkdf2 functions. Do not use the md5 or sha1 for sensitive values, and avoid hash and hash_init whenever possible. If you use hash_pbkdf2 or crypt choose a hashing algorithms which is known to be strong. Check regularly that this is still the case as hashing algorithms often lose strength over time. It is recommended to use a hashing function that generate salts automatically, but if you generate salts separately: generate a cryptographically strong and random salt that is unique for every credential being hashed. the salt is applied correctly before the hashing. save both the salt and the hashed value in the relevant database record; during future validation operations, the salt and hash can then be retrieved from the database. The hash is recalculated with the stored salt and the value being validated, and the result compared to the stored hash. Note that password_hash generates	Security_Hotspot	Critical	33

	strong salts automatically. Remember to rehash your data regularly as the hashing algorithms become less secure over time. The password_needs_rehash function helps you with that. Exceptions HMAC computing is out of the scope of this rule. Thus no issue will be raised when the hash_init function is called with HASH_HMAC given as second parameter. See OWASP Top 10 2017 Category A3 - Sensitive Data Exposure OWASP Top 10 2017 Category A6 - Security Misconfiguration MITRE, CWE-916 - Use of Password Hash With Insufficient Computational Effort MITRE, CWE-759 - Use of a One-Way Hash without a Salt MITRE, CWE-760 - Use of a One-Way Hash with a Predictable Salt SANS Top 25 - Porous Defenses			
Creating cookies without the "secure" flag is security-sensitive	The "secure" attribute prevents cookies from being sent over plaintext connections such as HTTP, where they would be easily eavesdropped upon. Instead, cookies with the secure attribute are only sent over encrypted HTTPS connections. Recommended Secure Coding Practices set the last parameter of the setcookie function to "true" set session.cookie_secure = 1 in the php.ini file Noncompliant Code Example ; php.ini session.cookie_secure = 0; Noncompliant // in PHP code session_set_cookie_params($lifetime, $path, $domain, false); // Noncompliant, the last parameter means that the session cookie should not be secure setcookie($name, $value, $expire, $path, $domain, false); // Noncompliant, the last parameter means that the cookie should not be secure See OWASP Top 10 2017 Category A2 - Broken Authentication OWASP Top 10 2017 Category A3 - Sensitive Data Exposure MITRE, CWE-311 - Missing Encryption of Sensitive Data MITRE, CWE-315 - Cleartext Storage of Sensitive Information in a Cookie MITRE, CWE-614 - Sensitive Cookie in HTTPS Session Without 'Secure' Attribute SANS Top 25 - Porous Defenses	Security_Hotspot	Minor	1
Credentials should not be hard-coded	Because it is easy to extract strings from a compiled application, credentials should never be hard- coded. Do so, and they're almost guaranteed to end up in the hands of an attacker. This is particularly true for applications that are distributed. Credentials should be stored outside of the code in a strongly-protected encrypted configuration file or database. Noncompliant Code Example $uname = "steve"; $password = "blue"; connect($uname, $password); Compliant Solution	Vulnerability	Blocker	4

	$uname = getEncryptedUser(); $password = getEncryptedPass(); connect($uname, $password); See OWASP Top 10 2017 Category A2 - Broken Authentication MITRE, CWE-798 - Use of Hard- coded Credentials MITRE, CWE-259 - Use of Hard-coded Password CERT, MSC03-J. - Never hard code sensitive information SANS Top 25 - Porous Defenses Derived from FindSecBugs rule Hard Coded Password			
"sleep" should not be called	sleep is sometimes used in a mistaken attempt to prevent Denial of Service (DoS) attacks by throttling response rate. But because it ties up a thread, each request takes longer to serve that it otherwise would, making the application more vulnerable to DoS attacks, rather than less. Noncompliant Code Example if (is_bad_ip($requester)) { sleep(5); // Noncompliant } See OWASP Top 10 2017 Category A6 - Security Misconfiguration	Vulnerability	Minor	1

About INFOPERCEPT

Infopercept's vision and core values revolve around making organizations more secure through the core values of Honesty, Transparency and Knowledge, so as to enable them to make better informed decisions about their security practices & goals. With our synergistic vision to combine technical expertise and professional experience, we aim to further establish our place as a one stop shop for our clients and partners' cybersecurity and accreditation needs.

Our specialized core team comprises of experienced veterans, technical experts & security enthusiasts having good practical experience & thorough knowledge in the Cybersecurity domain, are abreast of the latest trends and security innovations; ensuring that you always get the best security approach & solutions for your specific business needs, exactly the way you want it to be.

Imprint

Publisher
H-1209, Titanium City Center,
Satellite Road,
Ahmedabad – 380 015,
Gujarat, India.

Contact Info
M: +91 9898857117
W: www.infopercept.com
E : sos@infopercept.com

Global Offices

UNITED STATES OF AMERICA
+1 516 713 5040

UNITED KINGDOM
+44 2035002056

SRI LANKA
+94 702 958 909

KUWAIT
+965 6099 1177

INDIA
+91 9898857117

Infopercept