PHP static code analysis

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

1. HTTP responses should not be vulnerable to session fixation
   Vulnerability
2. Include statements should not be vulnerable to injection attacks
   Vulnerability
3. Dynamic code execution should not be vulnerable to injection attacks
   Vulnerability
4. HTTP request redirections should not be open to forging attacks
   Vulnerability
5. Deserialization should not be vulnerable to injection attacks
   Vulnerability
6. Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks
   Vulnerability
7. Database queries should not be vulnerable to injection attacks
   Vulnerability
8. XML parsers should not be vulnerable to XXE attacks
   Vulnerability
9. A secure password should be used when connecting to a database
   Vulnerability
10. XPath expressions should not be vulnerable to injection attacks
    Vulnerability
11. I/O function calls should not be vulnerable to path injection attacks
    Vulnerability
12. LDAP queries should not be vulnerable to injection attacks
    Vulnerability
13. OS commands should not be vulnerable to command injection attacks
    Vulnerability
14. Class of caught exception should be defined
    Bug
15. Caught Exceptions must derive from Throwable
    Bug
16.

	Raised Exceptions must derive from Throwable Bug
17.	
	"$this" should not be used in a static context Bug
18.	
	Hard-coded credentials are security-sensitive Security Hotspot
19.	
	Test class names should end with "Test" Code Smell
20.	
	Tests should include assertions Code Smell
21.	
	TestCases should contain tests Code Smell
22.	
	Variable variables should not be used Code Smell
23.	
	A new session should be created during user authentication Vulnerability
24.	
	Cipher algorithms should be robust Vulnerability
25.	
	Encryption algorithms should be used with secure mode and padding scheme Vulnerability
26.	
	Server hostnames should be verified during SSL/TLS connections Vulnerability
27.	
	Server certificates should be verified during SSL/TLS connections Vulnerability
28.	
	LDAP connections should be authenticated Vulnerability
29.	
	Cryptographic keys should be robust Vulnerability
30.	
	Weak SSL/TLS protocols should not be used Vulnerability
31.	
	Regular expressions should not be vulnerable to Denial of Service attacks Vulnerability
32.	
	Hashes should include an unpredictable salt Vulnerability
33.	

	Regular expressions should have valid delimiters Bug
34.	
	Regex lookahead assertions should not be contradictory Bug
35.	
	Back references in regular expressions should only refer to capturing groups that are matched before the reference Bug
36.	
	Regex boundaries should not be used in a way that can never be matched Bug
37.	
	Regex patterns following a possessive quantifier should not always fail Bug
38.	
	Assertion failure exceptions should not be ignored Bug
39.	
	References used in "foreach" loops should be "unset" Bug
40.	
	Using clear-text protocols is security-sensitive Security Hotspot
41.	
	Expanding archive files without controlling resource consumption is security-sensitive Security Hotspot
42.	
	Signalling processes is security-sensitive Security Hotspot
43.	
	Configuring loggers is security-sensitive Security Hotspot
44.	
	Using weak hashing algorithms is security-sensitive Security Hotspot
45.	
	Disabling CSRF protections is security-sensitive Security Hotspot
46.	
	Using pseudorandom number generators (PRNGs) is security-sensitive Security Hotspot
47.	
	Dynamically executing code is security-sensitive Security Hotspot
48.	
	`str_replace` should be preferred to `preg_replace` Code Smell
49.	
	"default" clauses should be first or last Code Smell

50. A conditionally executed single line should be denoted by indentation
Code Smell

51. Conditionals should start on new lines
Code Smell

52. Cognitive Complexity of functions should not be too high
Code Smell

53. Parentheses should not be used for calls to "echo"
Code Smell

54. Functions should not be nested too deeply
Code Smell

55. References should not be passed to function calls
Code Smell

56. "switch" statements should have "default" clauses
Code Smell

57. Control structures should use curly braces
Code Smell

58. String literals should not be duplicated
Code Smell

59. Methods should not be empty
Code Smell

60. Constant names should comply with a naming convention
Code Smell

61. Secret keys and salt values should be robust
Vulnerability

62. Authorizations should be based on strong decisions
Vulnerability

63. Server-side requests should not be vulnerable to forging attacks
Vulnerability

64. The number of arguments passed to a function should match the number of parameters
Bug

65. Non-empty statements should change control flow or have at least one side-effect
Bug

66. Variables should be initialized before use
Bug

67.	Replacement strings should reference existing regular expression groups Bug
68.	Alternation in regular expressions should not contain empty alternatives Bug
69.	Unicode Grapheme Clusters should be avoided inside regex character classes Bug
70.	Assertions should not compare an object to itself Bug
71.	Regex alternatives should not be redundant Bug
72.	Alternatives in regular expressions should be grouped when used with anchors Bug
73.	Array values should not be replaced unconditionally Bug
74.	Exceptions should not be created without being thrown Bug
75.	Array or Countable object count comparisons should make sense Bug
76.	All branches in a conditional structure should not have exactly the same implementation Bug
77.	The output of functions that don't return anything should not be used Bug
78.	Unary prefix operators should not be repeated Bug
79.	"=+" should not be used instead of "+=" Bug
80.	A "for" loop update clause should move the counter in the right direction Bug
81.	Return values from functions without side effects should not be ignored Bug
82.	Values should not be uselessly incremented Bug
83.	Related "if/else if" statements and "cases" in a "switch" should not have the same condition

	Bug
84.	
	Objects should not be created to be dropped immediately without being used Bug
85.	
	Identical expressions should not be used on both sides of a binary operator Bug
86.	
	All code should be reachable Bug
87.	
	Loops with at most one iteration should be refactored Bug
88.	
	Short-circuit logic should be used to prevent null pointer dereferences in conditionals Bug
89.	
	Variables should not be self-assigned Bug
90.	
	Useless "if(true) {...}" and "if(false){...}" blocks should be removed Bug
91.	
	All "catch" blocks should be able to catch exceptions Bug
92.	
	Constructing arguments of system commands from user input is security-sensitive Security Hotspot
93.	
	Allowing unfiltered HTML content in WordPress is security-sensitive Security Hotspot
94.	
	Allowing unauthenticated database repair in WordPress is security-sensitive Security Hotspot
95.	
	Allowing all external requests from a WordPress server is security-sensitive Security Hotspot
96.	
	Disabling automatic updates is security-sensitive Security Hotspot
97.	
	WordPress theme and plugin editors are security-sensitive Security Hotspot
98.	
	Allowing requests with excessive content length is security-sensitive Security Hotspot
99.	
	Manual generation of session ID is security-sensitive Security Hotspot
100.	
	Setting loose POSIX file permissions is security-sensitive

Security Hotspot

101. Formatting SQL queries is security-sensitive
Security Hotspot

102. "goto" statement should not be used
Code Smell

103. Character classes in regular expressions should not contain only one character
Code Smell

104. Superfluous curly brace quantifiers should be avoided
Code Smell

105. Non-capturing groups without quantifier should not be used
Code Smell

106. WordPress option names should not be misspelled
Code Smell

107. WordPress options should not be defined at the end of "wp-config.php"
Code Smell

108. Constants should not be redefined
Code Smell

109. Regular expressions should not contain empty groups
Code Smell

110. Regular expressions should not contain multiple spaces
Code Smell

111. Single-character alternations in regular expressions should be replaced with character classes
Code Smell

112. Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string
Code Smell

113. Character classes in regular expressions should not contain the same character twice
Code Smell

114. Regular expressions should not be too complicated
Code Smell

115. PHPUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion
Code Smell

116. Methods should not have identical implementations
Code Smell

117. Functions should use "return" consistently
Code Smell

118. Assertion arguments should be passed in the correct order
Code Smell

119. Ternary operators should not be nested
Code Smell

120. Reflection should not be used to increase accessibility of classes, methods, or fields
Code Smell

121. Multiline blocks should be enclosed in curly braces
Code Smell

122. Parameters should be passed in the correct order
Code Smell

123. Classes named like "Exception" should extend "Exception" or a subclass
Code Smell

124. Two branches in a conditional structure should not have exactly the same implementation
Code Smell

125. Unused assignments should be removed
Code Smell

126. Method arguments with default values should be last
Code Smell

127. A reason should be provided when skipping a test
Code Smell

128. "__construct" functions should not make PHP 4-style calls to parent constructors
Code Smell

129. PHP 4 constructor declarations should not be used
Code Smell

130. Deprecated predefined variables should not be used
Code Smell

131. "switch" statements should not have too many "case" clauses
Code Smell

132. Classes should not have too many methods
Code Smell

133. Functions should not have too many lines of code

	Code Smell
134.	
	"for" loop stop conditions should be invariant Code Smell
135.	
	Sections of code should not be commented out Code Smell
136.	
	Unused function parameters should be removed Code Smell
137.	
	Unused "private" methods should be removed Code Smell
138.	
	Functions should not contain too many return statements Code Smell
139.	
	Track uses of "FIXME" tags Code Smell
140.	
	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown Code Smell
141.	
	Local variables should not have the same name as class fields Code Smell
142.	
	Redundant pairs of parentheses should be removed Code Smell
143.	
	Inheritance tree of classes should not be too deep Code Smell
144.	
	Nested blocks of code should not be left empty Code Smell
145.	
	Functions should not have too many parameters Code Smell
146.	
	Unused "private" fields should be removed Code Smell
147.	
	Collapsible "if" statements should be merged Code Smell
148.	
	OS commands should not be vulnerable to argument injection attacks Vulnerability
149.	
	Logging should not be vulnerable to injection attacks Vulnerability
150.	

Repeated patterns in regular expressions should not match the empty string
Bug

151.

Function and method parameters' initial values should not be ignored
Bug

152.

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive
Security Hotspot

153.

Delivering code in production with debug features activated is security-sensitive
Security Hotspot

154.

Creating cookies without the "HttpOnly" flag is security-sensitive
Security Hotspot

155.

Creating cookies without the "secure" flag is security-sensitive
Security Hotspot

156.

Using hardcoded IP addresses is security-sensitive
Security Hotspot

157.

Regular expression quantifiers and character classes should be used concisely
Code Smell

158.

Character classes should be preferred over reluctant quantifiers in regular expressions
Code Smell

159.

A subclass should not be in the same "catch" clause as a parent class
Code Smell

160.

Jump statements should not be redundant
Code Smell

161.

"catch" clauses should do more than rethrow
Code Smell

162.

"&&" and "||" should be used
Code Smell

163.

Boolean checks should not be inverted
Code Smell

164.

Local variables should not be declared and then immediately returned or thrown
Code Smell

165.

Unused local variables should be removed
Code Smell

166.

"switch" statements should have at least 3 "case" clauses
Code Smell

167.

	A "while" loop should be used instead of a "for" loop Code Smell
168.	
	Overriding methods should do more than simply call the same method in the super class Code Smell
169.	
	"empty()" should be used to test for emptiness Code Smell
170.	
	Interface names should comply with a naming convention Code Smell
171.	
	Return of boolean expressions should not be wrapped into an "if-then-else" statement Code Smell
172.	
	Boolean literals should not be redundant Code Smell
173.	
	Empty statements should be removed Code Smell
174.	
	A close curly brace should be located at the beginning of a line Code Smell
175.	
	URIs should not be hardcoded Code Smell
176.	
	Class names should comply with a naming convention Code Smell
177.	
	Track uses of "TODO" tags Code Smell
178.	
	"file_uploads" should be disabled Vulnerability
179.	
	"enable_dl" should be disabled Vulnerability
180.	
	"session.use_trans_sid" should not be enabled Vulnerability
181.	
	"allow_url_fopen" and "allow_url_include" should be disabled Vulnerability
182.	
	"open_basedir" should limit file access Vulnerability
183.	
	Neither DES (Data Encryption Standard) nor DESede (3DES) should be used Vulnerability
184.	

	"exit(...)" and "die(...)" statements should not be used Bug
185.	
	Functions and variables should not be defined outside of classes Code Smell
186.	
	Track lack of copyright and license headers Code Smell
187.	
	Octal values should not be used Code Smell
188.	
	Switch cases should end with an unconditional "break" statement Code Smell
189.	
	Session-management cookies should not be persistent Vulnerability
190.	
	Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding) Vulnerability
191.	
	SHA-1 and Message-Digest hash algorithms should not be used in secure contexts Vulnerability
192.	
	Assertions should not be made at the end of blocks expecting an exception Bug
193.	
	Regular expressions should be syntactically valid Bug
194.	
	Only one method invocation is expected when testing exceptions Bug
195.	
	Reading the Standard Input is security-sensitive Security Hotspot
196.	
	Using command line arguments is security-sensitive Security Hotspot
197.	
	Using Sockets is security-sensitive Security Hotspot
198.	
	Encrypting data is security-sensitive Security Hotspot
199.	
	Using regular expressions is security-sensitive Security Hotspot
200.	
	Deserializing objects from an untrusted source is security-sensitive Security Hotspot

201.

Literal boolean values and nulls should not be used in equality assertions
Code Smell

202.

"global" should not be used
Code Smell

203.

"switch" statements should not be nested
Code Smell

204.

Cyclomatic Complexity of functions should not be too high
Code Smell

205.

Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply
Code Smell

206.

Cyclomatic Complexity of classes should not be too high
Code Smell

207.

"if ... else if" constructs should end with "else" clauses
Code Smell

208.

Expressions should not be too complex
Code Smell

209.

"cgi.force_redirect" should be enabled
Vulnerability

210.

Files that define symbols should not cause side-effects
Bug

211.

Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression
Code Smell

212.

Test methods should be discoverable
Code Smell

213.

Use of namespaces should be preferred to "include" or "require" functions
Code Smell

214.

Duplicate values should not be passed as arguments
Code Smell

215.

Configuration should not be changed dynamically
Code Smell

216.

Class constructors should not create other objects
Code Smell

217.

	PHP parser failure Code Smell
218.	
	The names of methods with boolean return values should start with "is" or "has" Code Smell
219.	
	"php_sapi_name()" should not be used Code Smell
220.	
	Classes should not have too many lines of code Code Smell
221.	
	Deprecated features should not be used Code Smell
222.	
	Files should not contain inline HTML Code Smell
223.	
	Files should contain only one top-level class or interface each Code Smell
224.	
	Classes should not have too many fields Code Smell
225.	
	Track uses of "NOSONAR" comments Code Smell
226.	
	Statements should be on separate lines Code Smell
227.	
	Classes should not be coupled to too many other classes (Single Responsibility Principle) Code Smell
228.	
	"switch case" clauses should not have too many lines of code Code Smell
229.	
	Assignments should not be made from within sub-expressions Code Smell
230.	
	Files should not have too many lines of code Code Smell
231.	
	Lines should not be too long Code Smell
232.	
	HTTP response headers should not be vulnerable to injection attacks Vulnerability
233.	
	"sleep" should not be called Vulnerability
234.	

	Static members should be referenced with "static::" Bug
235.	
	"require_once" and "include_once" should be used instead of "require" and "include" Bug
236.	
	Errors should not be silenced Bug
237.	
	Files should not contain characters before "<?php" Bug
238.	
	Method visibility should be explicitly declared Bug
239.	
	Controlling permissions is security-sensitive Security Hotspot
240.	
	Writing cookies is security-sensitive Security Hotspot
241.	
	Framework-provided functions should be used to test exceptions Code Smell
242.	
	Unicode-aware versions of character classes should be preferred Code Smell
243.	
	Alias functions should not be used Code Smell
244.	
	Perl-style comments should not be used Code Smell
245.	
	Superglobals should not be accessed directly Code Smell
246.	
	Colors should be defined in upper case Code Smell
247.	
	String literals should not be concatenated Code Smell
248.	
	"final" should not be used redundantly Code Smell
249.	
	Source code should comply with formatting standards Code Smell
250.	
	"elseif" keyword should be used in place of "else if" keywords Code Smell
251.	

PHP keywords and constants "true", "false", "null" should be lower case
Code Smell

252.

Closing tag "?>" should be omitted on files containing only PHP
Code Smell

253.

Only LF character (Unix-like) should be used to end lines
Code Smell

254.

More than one property should not be declared per statement
Code Smell

255.

The "var" keyword should not be used
Code Smell

256.

"<?php" and "<?=" tags should be used
Code Smell

257.

File names should comply with a naming convention
Code Smell

258.

Comments should not be located at the end of lines of code
Code Smell

259.

Local variable and function parameter names should comply with a naming convention
Code Smell

260.

Field names should comply with a naming convention
Code Smell

261.

Lines should not end with trailing whitespaces
Code Smell

262.

Files should contain an empty newline at the end
Code Smell

263.

Modifiers should be declared in the correct order
Code Smell

264.

An open curly brace should be located at the beginning of a line
Code Smell

265.

An open curly brace should be located at the end of a line
Code Smell

266.

Tabulation characters should not be used
Code Smell

267.

Method and function names should comply with a naming convention
Code Smell

268.

Creating cookies with broadly defined "domain" flags is security-sensitive Security Hotspot