

- 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

Deserializing objects from an untrusted source is security-sensitive

Analyze your code

Security Hotspot Critical

Deserializing objects is security-sensitive. For example, it has led in the past to the following vulnerabilities:

- CVE-2017-17672: vBulletin: Unserialize PHP Code Execution
- CVE-2018-1000167: Jenkins Pipeline: arbitrary code execution vulnerability

Object deserialization from an untrusted source can lead to unexpected code execution. Deserialization takes a stream of bits and turns it into an object. If the stream contains the type of object you expect, all is well. But if you're deserializing data coming from untrusted input, and an attacker has inserted some other type of object, you're in trouble. Why? A known attack scenario involves the creation of a serialized PHP object with crafted attributes which will modify your application's behavior. This attack relies on PHP magic methods like \_\_destruct, \_\_wakeup or \_\_string. The attacker doesn't necessarily need the source code of the targeted application to exploit the vulnerability, he can also rely on the presence of open-source component and use tools to craft malicious payloads.

Ask Yourself Whether

- an attacker could have tampered with the source provided to the deserialization function
- you are using an unsafe deserialization function

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

Recommended Secure Coding Practices

To prevent insecure deserialization, it is recommended to:

- Use safe libraries that do not allow code execution at deserialization.
- Not communicate with the outside world using serialized objects
- Limit access to the serialized source
  - if it is a file, restrict the access to it.
  - if it comes from the network, restrict who has access to the process, such as with a Firewall or by authenticating the sender first.

See


- OWASP - Deserialization of untrusted data
- OWASP Top 10 2017 Category A8 - Insecure Deserialization
- MITRE, CWE-502 - Deserialization of Untrusted Data
- Derived from FindSecBugs rule OBJECT\_DESERIALIZATION

Deprecated


This rule is deprecated, and will eventually be removed.

Available In:




 Code Smell


**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