

- 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



Python static code analysis

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

All rules 216 Vulnerability 29 Bug 55 Security Hotspot 31 Code Smell 101

Tags Search by name...

Functions should not have too many lines of code
Code Smell
Track uses of "NOSONAR" comments
Code Smell
Track comments matching a regular expression
Code Smell
Statements should be on separate lines
Code Smell
Functions should not contain too many return statements
Code Smell
Files should not have too many lines of code
Code Smell
Lines should not be too long
Code Smell
Methods and properties that don't access instance data should be static
Code Smell
New-style classes should be used
Code Smell
Parentheses should not be used after certain keywords
Code Smell
Track "TODO" and "FIXME" comments that do not contain a reference to a person
Code Smell
Module names should comply with a naming convention

Using shell interpreter when executing OS commands is security-sensitive

Analyze your code

Security Hotspot Major cwe owasp sans-top25

Arbitrary OS command injection vulnerabilities are more likely when a shell is spawned rather than a new process, indeed shell meta-chars can be used (when parameters are user-controlled for instance) to inject OS commands.

Ask Yourself Whether

- OS command name or parameters are user-controlled.

There is a risk if you answered yes to this question.

Recommended Secure Coding Practices

Use functions that don't spawn a shell.

Sensitive Code Example

Python 3

```
subprocess.run(cmd, shell=True) # Sensitive
subprocess.Popen(cmd, shell=True) # Sensitive
subprocess.call(cmd, shell=True) # Sensitive
subprocess.check_call(cmd, shell=True) # Sensitive
subprocess.check_output(cmd, shell=True) # Sensitive
os.system(cmd) # Sensitive: a shell is always spawn
```

Python 2






```
cmd = "when a string is passed through these function,
(_, child_stdout, _) = os.popen2(cmd) # Sensitive
(_, child_stdout, _) = os.popen3(cmd) # Sensitive
(_, child_stdout) = os.popen4(cmd) # Sensitive

(child_stdout, _) = popen2.popen2(cmd) # Sensitive
(child_stdout, _, _) = popen2.popen3(cmd) # Sensitive
(child_stdout, _) = popen2.popen4(cmd) # Sensitive
```

Compliant Solution

Python 3

```
# by default shell=False, a shell is not spawn
subprocess.run(cmd) # Compliant
subprocess.Popen(cmd) # Compliant
subprocess.call(cmd) # Compliant
subprocess.check_call(cmd) # Compliant
```

 Code Smell
Comments should not be located at the end of lines of code
 Code Smell
Lines should not end with trailing whitespaces
 Code Smell
Files should contain an empty newline at the end
 Code Smell
Long suffix "L" should be upper case
 Code Smell

```
subprocess.check_output(cmd) # Compliant
```

```
# always in a subprocess:
```

```
os.spawnl(mode, path, *cmd) # Compliant
os.spawnle(mode, path, *cmd, env) # Compliant
os.spawnlp(mode, file, *cmd) # Compliant
os.spawnlpe(mode, file, *cmd, env) # Compliant
os.spawnv(mode, path, cmd) # Compliant
os.spawnve(mode, path, cmd, env) # Compliant
os.spawnvp(mode, file, cmd) # Compliant
os.spawnvpe(mode, file, cmd, env) # Compliant
```

```
(child_stdout) = os.popen(cmd, mode, 1) # Compliant
(_, output) = subprocess.getstatusoutput(cmd) # Compliant
out = subprocess.getoutput(cmd) # Compliant
os.startfile(path) # Compliant
os.execl(path, *cmd) # Compliant
os.execle(path, *cmd, env) # Compliant
os.execlp(file, *cmd) # Compliant
os.execlpe(file, *cmd, env) # Compliant
os.execv(path, cmd) # Compliant
os.execve(path, cmd, env) # Compliant
os.execvp(file, cmd) # Compliant
os.execvpe(file, cmd, env) # Compliant
```

Python 2

```
cmdsargs = ("use", "a", "sequence", "to", "directly", "

(_, child_stdout) = os.popen2(cmdsargs) # Compliant
(_, child_stdout, _) = os.popen3(cmdsargs) # Compliant
(_, child_stdout) = os.popen4(cmdsargs) # Compliant

(child_stdout, _) = popen2.popen2(cmdsargs) # Compliant
(child_stdout, _, _) = popen2.popen3(cmdsargs) # Compliant
(child_stdout, _) = popen2.popen4(cmdsargs) # Compliant
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-78](#) - Improper Neutralization of Special Elements used in an OS Command
- [SANS Top 25](#) - Insecure Interaction Between Components

Deprecated

This rule is deprecated, and will eventually be removed.

Available In:

sonardcloud  | **sonarqube** 