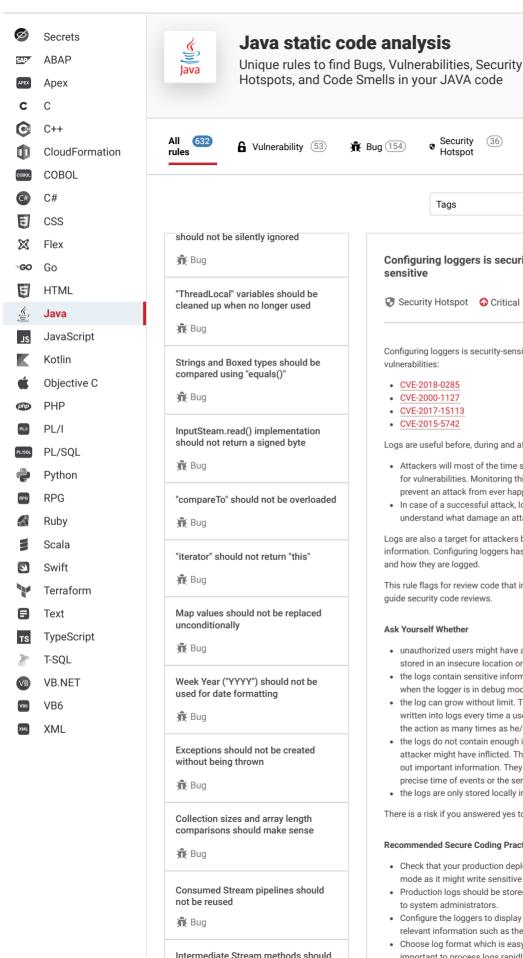
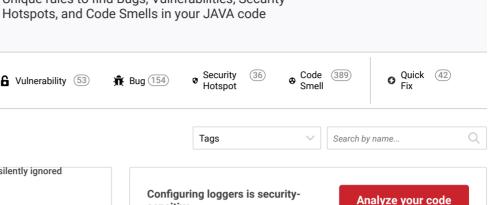


Products ✓





cwe owasp sans-top25

Configuring loggers is security-sensitive. It has led in the past to the following

• CVE-2018-0285

vulnerabilities

sensitive

- CVE-2000-1127
- CVE-2017-15113

Security Hotspot Oritical

CVE-2015-5742

Logs are useful before, during and after a security incident.

- Attackers will most of the time start their nefarious work by probing the system for vulnerabilities. Monitoring this activity and stopping it is the first step to prevent an attack from ever happening
- In case of a successful attack, logs should contain enough information to understand what damage an attacker may have inflicted.

Logs are also a target for attackers because they might contain sensitive information. Configuring loggers has an impact on the type of information logged and how they are logged.

This rule flags for review code that initiates loggers configuration. The goal is to guide security code reviews

## Ask Yourself Whether

- · unauthorized users might have access to the logs, either because they are stored in an insecure location or because the application gives access to them.
- the logs contain sensitive information on a production server. This can happen when the logger is in debug mode.
- the log can grow without limit. This can happen when additional information is written into logs every time a user performs an action and the user can perform the action as many times as he/she wants.
- the logs do not contain enough information to understand the damage an attacker might have inflicted. The loggers mode (info, warn, error) might filter out important information. They might not print contextual information like the precise time of events or the server hostname
- the logs are only stored locally instead of being backuped or replicated.

There is a risk if you answered yes to any of those questions.

## **Recommended Secure Coding Practices**

- Check that your production deployment doesn't have its loggers in "debug" mode as it might write sensitive information in logs.
- Production logs should be stored in a secure location which is only accessible to system administrators.
- Configure the loggers to display all warnings, info and error messages. Write relevant information such as the precise time of events and the hostname.
- Choose log format which is easy to parse and process automatically. It is important to process logs rapidly in case of an attack so that the impact is known and limited.
- Check that the permissions of the log files are correct. If you index the logs in some other service, make sure that the transfer and the service are secure too.

not be left unused

📆 Bug

All branches in a conditional structure should not have exactly the same implementation



Optional value should only be accessed after calling isPresent()



Overrides should match their parent class methods in synchronization



Value-based classes should not be used for locking



Add limits to the size of the logs and make sure that no user can fill the disk
with logs. This can happen even when the user does not control the logged
information. An attacker could just repeat a logged action many times.

Remember that configuring loggers properly doesn't make them bullet-proof. Here is a list of recommendations explaining on how to use your logs:

- Don't log any sensitive information. This obviously includes passwords and credit card numbers but also any personal information such as user names, locations, etc... Usually any information which is protected by law is good candidate for removal.
- Sanitize all user inputs before writing them in the logs. This includes checking
  its size, content, encoding, syntax, etc... As for any user input, validate using
  whitelists whenever possible. Enabling users to write what they want in your
  logs can have many impacts. It could for example use all your storage space or
  compromise your log indexing service.
- Log enough information to monitor suspicious activities and evaluate the impact an attacker might have on your systems. Register events such as failed logins, successful logins, server side input validation failures, access denials and any important transaction.
- Monitor the logs for any suspicious activity.

## Sensitive Code Example

This rule supports the following libraries: Log4J, java.util.logging and Logback

```
// === Log4J 2 ===
import org.apache.logging.log4j.core.config.builder.api.Conf
import org.apache.logging.log4j.Level;
import org.apache.logging.log4j.core.*;
import org.apache.logging.log4j.core.config.*;
// Sensitive: creating a new custom configuration
abstract class CustomConfigFactory extends ConfigurationFact
class A {
    void foo(Configuration config, LoggerContext context, ja
           Appender appender, java.io.InputStream stream, j
            java.io.File file, java.net.URL url, String sour
            throws java.io.IOException {
        // Creating a new custom configuration
        ConfigurationBuilderFactory.newConfigurationBuilder(
        // Setting loggers level can result in writing sensi
        Configurator.setAllLevels("com.example", Level.DEBUG
        Configurator.setLevel("com.example", Level.DEBUG);
        Configurator.setLevel(levelMap); // Sensitive
        Configurator.setRootLevel(Level.DEBUG); // Sensitiv
        config.addAppender(appender); // Sensitive: this mod
        LoggerConfig loggerConfig = config.getRootLogger();
        loggerConfig.addAppender(appender, level, filter); /
        loggerConfig.setLevel(level); // Sensitive
        context.setConfigLocation(uri); // Sensitive
        // Load the configuration from a stream or file
        new ConfigurationSource(stream): // Sensitive
        new ConfigurationSource(stream, file); // Sensitive
        new ConfigurationSource(stream, url); // Sensitive
        ConfigurationSource.fromResource(source, loader); /
        ConfigurationSource.fromUri(uri); // Sensitive
}
```

```
// === Logback ===
 import ch.qos.logback.classic.util.ContextInitializer;
 import ch.qos.logback.core.Appender;
 import ch.qos.logback.classic.joran.JoranConfigurator;
 import ch.qos.logback.classic.spi.ILoggingEvent;
 import ch.qos.logback.classic.*;
 class M {
     void foo(Logger logger, Appender<ILoggingEvent> fileAppe
          System.setProperty(ContextInitializer.CONFIG_FILE_PR
          JoranConfigurator configurator = new JoranConfigurat
          logger.addAppender(fileAppender); // Sensitive
          logger.setLevel(Level.DEBUG); // Sensitive
Exceptions
Log4J 1.x is not covered as it has reached end of life.
```

- OWASP Top 10 2021 Category A9 Security Logging and Monitoring Failures
- OWASP Top 10 2017 Category A3 Sensitive Data Exposure
- OWASP Top 10 2017 Category A10 Insufficient Logging & Monitoring
- MITRE, CWE-117 Improper Output Neutralization for Logs
- MITRE, CWE-532 Information Exposure Through Log Files
- SANS Top 25 Porous Defenses

Available In:

sonarcloud 🙆 | sonarqube

© 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