FIO13-J. Do not log sensitive information outside a trust boundary

Logging is essential for debugging, incident response, and collecting forensic evidence. Nevertheless, logging sensitive data raises many concerns, including the privacy of the stakeholders, limitations imposed by the law on the collection of personal information, and the potential for data exposure by insiders. Sensitive information includes, but is not limited to, IP addresses, user names and passwords, email addresses, credit card numbers, and any personally identifiable information such as social security numbers. Many countries prohibit or restrict collection of personal data; others permit retention of personal data only when held in an anonymized form. For example, leaking unencrypted credit card numbers into a log file could be a violation of PCI DSS (Payment Card Industry Data Security Standard) regulations [PCI 2010]. Consequently, logs must not contain sensitive data, particularly when prohibited by law.

Unfortunately, violations of this rule are common. For example, prior to version 0.8.1, the LineControl Java client logged sensitive information, including the local user's password, as documented by CVE-2005-2990.

The `java.util.logging` class provides a basic logging framework for JDK versions 1.4 and higher. Other logging frameworks exist, but the basic principles apply regardless of the particular logging framework chosen.

Programs typically support varying levels of protection. Some information, such as access times, can be safely logged. Some information can be logged, but the log file must be restricted from everyone but particular administrators. Other information, such as credit card numbers, can be included in logs only in encrypted form. Information such as passwords should not be logged at all.

For the following code examples, the log lies outside the trust boundary of the information being recorded. Also, normal log messages should include additional parameters such as date, time, source event, and so forth. This information has been omitted from the following code examples for brevity.

Noncompliant Code Example

In this noncompliant code example, a server logs the IP address of the remote client in the event of a security exception. This data can be misused, for example, to build a profile of a user's browsing habits. Such logging may violate legal restrictions in many countries.

When the log cannot contain IP addresses, it should not contain any information about a `SecurityException`, because it might leak an IP address. When an exception contains sensitive information, the custom `MyExceptionReporter` class should extract or cleanse it before returning control to the next statement in the `catch` block (see ERR00-J. Do not suppress or ignore checked exceptions).

```
public void logRemoteIPAddress(String name) {
  Logger logger = Logger.getLogger("com.organization.Log");
  InetAddress machine = null;
  try {
    machine = InetAddress.getByName(name);
  } catch (UnknownHostException e) {
    Exception e = MyExceptionReporter.handle(e);
  } catch (SecurityException e) {
    Exception e = MyExceptionReporter.handle(e);
    logger.severe(name + "," + machine.getHostAddress() + "," +
                  e.toString());
  }
}
```

Compliant Solution

This compliant solution does not log security exceptions except for the logging implicitly performed by `MyExceptionReporter`:

```
  // ...
  catch (SecurityException e) {
    Exception e = MyExceptionReporter.handle(e);
  }
```

Noncompliant Code Example

Log messages with sensitive information should not be printed to the console display for security reasons (a possible example of sensitive information is passenger age). The `java.util.logging.Logger` class supports different logging levels that can be used for classifying such information: `FINEST`, `FINER`, `FINE`, `CONFIG`, `INFO`, `WARNING`, and `SEVERE`. By default, the `INFO`, `WARNING`, and `SEVERE` levels print the message to the console, which is accessible by end users and system administrators.

If we assume that the passenger age can appear in log files on the current system but not on the console display, this code example is noncompliant.

```
logger.info("Age: " + passengerAge);
```

Compliant Solution

This compliant solution logs the passenger age at the `FINEST` level to prevent this information from displaying on the console. As noted previously, we are assuming the age may appear in system log files but not on the console.

```
// Make sure that all handlers only print log messages rated INFO or higher
Handler handlers[] = logger.getHandlers();
for (int i = 0; i < handlers.length; i++) {
  handlers[i].setLevel(Level.INFO);
}
// ...
logger.finest("Age: " + passengerAge);
```

Risk Assessment

Logging sensitive information can violate system security policies and can violate user privacy when the logging level is incorrect or when the log files are insecure.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
FIO13-J	Medium	Probable	High	**P4**	**L3**

Automated Detection

Tool	Version	Checker	Description
Parasoft Jtest	10.3	**BD.SECURITY.SENS, HIBERNATE.LHII, SECURITY.ESD.PEO, SECURITY.ESD.CONSEN**	Implemented

Related Guidelines

MITRE CWE	CWE-359, Privacy Violation CWE-532, Information Exposure through Log Files CWE-533, Information Exposure through Server Log Files CWE-542, Information Exposure through Cleanup Log Files

Android Implementation Details

DRD04-J. Do not log sensitive information is an Android-specific instance of this rule.

Bibliography

[API 2014]	Class `java.util.logging.Logger`
[Chess 2007]	Section 11.1, "Privacy and Regulation: Handling Private Information"
[CVE 2011]	CVE-2005-2990
[PCI DSS Standard]	Payment Card Industry (PCI) Data Security Standard
[Sun 2006]	Java Logging Overview