IDS03-J. Do not log unsanitized user input

A log injection vulnerability arises when a log entry contains unsanitized user input. A malicious user can insert fake log data and consequently deceive system administrators as to the system's behavior [OWASP 2008]. For example, an attacker might split a legitimate log entry into two log entries by entering a carriage return and line feed (CRLF) sequence to mislead an auditor. Log injection attacks can be prevented by sanitizing and validating any untrusted input sent to a log.

Logging unsanitized user input can also result in leaking sensitive data across a trust boundary. For example, an attacker might inject a script into a log file such that when the file is viewed using a web browser, the browser could provide the attacker with a copy of the administrator's cookie so that the attacker might gain access as the administrator.

Noncompliant Code Example

This noncompliant code example logs untrusted data from an unauthenticated user without data sanitization.

```
if (loginSuccessful) {
  logger.severe("User login succeeded for: " + username);
} else {
  logger.severe("User login failed for: " + username);
}
```

Without sanitization, a log injection attack is possible. A standard log message when `username` is `guest` might look like this:

```
May 15, 2011 2:19:10 PM java.util.logging.LogManager$RootLogger log
SEVERE: User login failed for: guest
```

If the `username` that is used in a log message is not `guest` but rather a multiline string like this:

```
guest
May 15, 2011 2:25:52 PM java.util.logging.LogManager$RootLogger log
SEVERE: User login succeeded for: administrator
```

the log would contain the following misleading data:

```
May 15, 2011 2:19:10 PM java.util.logging.LogManager$RootLogger log
SEVERE: User login failed for: guest
May 15, 2011 2:25:52 PM java.util.logging.LogManager log
SEVERE: User login succeeded for: administrator
```

Compliant Solution (Sanitized User)

This compliant solution sanitizes the `username` before logging it, preventing injection attacks.

```
if (loginSuccessful) {
  logger.severe("User login succeeded for: " + sanitizeUser(username));
} else {
  logger.severe("User login failed for: " + sanitizeUser(username));
}
```

The sanitization is done by a dedicated method for sanitizing user names:

```
public String sanitizeUser(String username) {
  return Pattern.matches("[A-Za-z0-9_]+", username))
      ? username : "unauthorized user";
}
```

Compliant Solution (Sanitized Logger)

This compliant solution uses a text logger that automatically sanitizes its input. A sanitized logger saves the developer from having to worry about unsanitized log messages.

```
Logger sanLogger = new SanitizedTextLogger(logger);

if (loginSuccessful) {
  sanLogger.severe("User login succeeded for: " + username);
} else {
  sanLogger.severe("User login failed for: " + username);
}
```

The sanitized text logger takes as delegate an actual logger. We assume the logger outputs text log messages to a file, network, or the console, and each log message has no indented lines. The sanitized text logger sanitizes all text to be logged by indenting every line except the first by two spaces. While a malicious user can indent text by more, a malicious user cannot create a fake log entry because all of her output will be indented, except for the real log output.

```
class SanitizedTextLogger extends Logger {
  Logger delegate;

  public SanitizedTextLogger(Logger delegate) {
    super(delegate.getName(), delegate.getResourceBundleName());
    this.delegate = delegate;
  }

  public String sanitize(String msg) {
    Pattern newline = Pattern.compile("\n");
    Matcher matcher = newline.matcher(msg);
    return matcher.replaceAll("\n  ");
  }

  public void severe(String msg) {
    delegate.severe(sanitize(msg));
  }

  // .. Other Logger methods which must also sanitize their log messages
}
```

Risk Assessment

Allowing unvalidated user input to be logged can result in forging of log entries, leaking secure information, or storing sensitive data in a manner that violates a local law or regulation.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
IDS03-J	Medium	Probable	Medium	P8	L2

Automated Detection

Tool	Version	Checker	Description
The Checker Framework	2.1.3	**Tainting Checker**	Trust and security errors (see Chapter 8)
Fortify		**Log_Forging**	Implemented
Klocwork		**SVLOG_FORGING**	Implemented
Parasoft Jtest	2020.2	**BD-SECURITY-TDLOG**	Protect against log forging

Related Guidelines

ISO/IEC TR 24772:2013	Injection [RST]
MITRE CWE	CWE-144, Improper neutralization of line delimiters CWE-150, Improper neutralization of escape, meta, or control sequences CWE-117, Improper Output Neutralization for Logs
MITRE CAPEC	CAPEC-93, Log Injection-Tampering-Forging

Bibliography

[API 2006]	Java Platform, Standard Edition 6 API Specification
[OWASP 2008]	
[Seacord 2015]	IDS03-J. Do not log unsanitized user input LiveLesson