FIO03-J. Remove temporary files before termination

Temporary files can be used to

- Share data between processes.
- Store auxiliary program data (for example, to preserve memory).
- Construct and/or load classes, JAR files, and native libraries dynamically.

Temporary files are *files* and consequently must conform to the requirements specified by other rules governing operations on files, including FIO00-J. Do not operate on files in shared directories and FIO01-J. Create files with appropriate access permissions. Temporary files have the additional requirement that they must be removed before program termination.

Removing temporary files when they are no longer required allows file names and other resources (such as secondary storage) to be recycled. Each program is responsible for ensuring that temporary files are removed during normal operation. There is no surefire method that can guarantee the removal of orphaned files in the case of abnormal termination, even in the presence of a `finally` block, because the `finally` block may fail to execute. For this reason, many systems employ temporary file cleaner utilities to sweep temporary directories and remove old files. Such utilities can be invoked manually by a system administrator or can be periodically invoked by a system process. However, these utilities are themselves frequently vulnerable to file-based exploits.

Noncompliant Code Example

This and subsequent code examples assume that files are created in a secure directory in compliance with FIO00-J. Do not operate on files in shared directories and are created with proper access permissions in compliance with FIO01-J. Create files with appropriate access permissions. Both requirements may be managed outside the Java Virtual Machine (JVM).

This noncompliant code example fails to remove the file upon completion:

```
class TempFile {
  public static void main(String[] args) throws IOException{
    File f = new File("tempnam.tmp");
    if (f.exists()) {
      System.out.println("This file already exists");
      return;
    }

    FileOutputStream fop = null;
    try {
      fop = new FileOutputStream(f);
      String str = "Data";
      fop.write(str.getBytes());
    } finally {
      if (fop != null) {
        try {
          fop.close();
        } catch (IOException x) {
          // Handle error
        }
      }
    }
  }
}
```

Noncompliant Code Example (`createTempFile()`, `deleteOnExit()`)

This noncompliant code example invokes the `File.createTempFile()` method, which generates a unique temporary file name based on two parameters: a prefix and an extension. This is the only method from Java 6 and earlier that is designed to produce unique file names, although the names produced can be easily predicted. A random number generator can be used to produce the prefix if a random file name is required.

This example also uses the `deleteOnExit()` method to ensure that the temporary file is deleted when the JVM terminates. However, according to the Java API [API 2014] Class `File`, method `deleteOnExit()` documentation,

> *Deletion will be attempted only for normal termination of the virtual machine, as defined by the* Java Language Specification. *Once deletion has been requested, it is not possible to cancel the request. This method should therefore be used with care.*
> *Note: this method should not be used for file-locking, as the resulting protocol cannot be made to work reliably.*

Consequently, the file is not deleted if the JVM terminates unexpectedly. A longstanding bug on Windows-based systems, reported as Bug ID: 4171239 [SDN 2008], causes JVMs to fail to delete a file when `deleteOnExit()` is invoked before the associated stream or `RandomAccessFile` is closed.

```
class TempFile {
  public static void main(String[] args) throws IOException{
    File f = File.createTempFile("tempnam",".tmp");
    FileOutputStream fop = null;
    try {
      fop = new FileOutputStream(f);
      String str = "Data";
      fop.write(str.getBytes());
      fop.flush();
    } finally {
      // Stream/file still open; file will
      // not be deleted on Windows systems
      f.deleteOnExit(); // Delete the file when the JVM terminates

      if (fop != null) {
        try {
          fop.close();
        } catch (IOException x) {
          // Handle error
        }
      }
    }
  }
}
```

Compliant Solution (`DELETE_ON_CLOSE`)

This compliant solution creates a temporary file using several methods from Java's NIO.2 package (introduced in Java SE 7). It uses the `createTempFile()` method, which creates an unpredictable name. (The actual method by which the name is created is implementation-defined and undocumented.) The file is opened using the *try-with-resources* construct, which automatically closes the file regardless of whether an exception occurs. Finally, the file is opened with the `DELETE_ON_CLOSE` option, which removes the file automatically when it is closed.

```
class TempFile {
  public static void main(String[] args) {
    Path tempFile = null;
    try {
      tempFile = Files.createTempFile("tempnam", ".tmp");
      try (BufferedWriter writer =
          Files.newBufferedWriter(tempFile, Charset.forName("UTF8"),
                                  StandardOpenOption.DELETE_ON_CLOSE)) {
        // Write to file
      }
      System.out.println("Temporary file write done, file erased");
    } catch (FileAlreadyExistsException x) {
      System.err.println("File exists: " + tempFile);
    } catch (IOException x) {
      // Some other sort of failure, such as permissions.
      System.err.println("Error creating temporary file: " + x);
    }
  }
}
```

Compliant Solution

When a secure directory for storing temporary files is not available, the vulnerabilities that result from using temporary files in insecure directories can be avoided by using alternative mechanisms, including

- Other IPC mechanisms such as sockets and remote procedure calls.
- The low-level Java Native Interface (JNI).
- Memory-mapped files.
- Threads to share heap data within the same JVM (applies to data sharing between Java processes only).

Risk Assessment

Failure to remove temporary files before termination can result in information leakage and resource exhaustion.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
FIO03-J	Medium	Probable	Medium	P8	L2

Automated Detection

Tool	Version	Checker	Description
Parasoft Jtest	10.3	**SECURITY.IBA.ATF**	Implemented

Related Guidelines

SEI CERT C Coding Standard	FIO21-C. Do not create temporary files in shared directories
SEI CERT C++ Coding Standard	VOID FIO19-CPP. Do not create temporary files in shared directories
MITRE CWE	CWE-377, Insecure Temporary File CWE-459, Incomplete Cleanup

Bibliography

[API 2014]	`Class File` Method `createTempFile` Method `delete` Method `deleteOnExit`
[Darwin 2004]	Section 11.5, "Creating a Transient File"
[J2SE 2011]	
[JDK Bug 2015]	Bug JDK-4405521 Bug JDK-4631820
[SDN 2008]	Bug ID: 4171239
[Secunia 2008]	Secunia Advisory 20132