ENV02-J. Do not trust the values of environment variables

Both environment variables and system properties provide user-defined mappings between keys and their corresponding values and can be used to communicate those values from the environment to a process. According to the Java API [API 2014] `java.lang.System` class documentation:

> *Environment variables have a more global effect because they are visible to all descendants of the process which defines them, not just the immediate Java subprocess. They can have subtly different semantics, such as case insensitivity, on different operating systems. For these reasons, environment variables are more likely to have unintended side effects. It is best to use system properties where possible. Environment variables should be used when a global effect is desired, or when an external system interface requires an environment variable (such as `PATH`).*

Programs that execute in a more trusted domain than their environment must assume that the values of environment variables are untrusted and must sanitize and validate any environment variable values before use.

The default values of system properties are set by the Java Virtual Machine (JVM) upon startup and can be considered trusted. However, they may be overridden by properties from untrusted sources, such as a configuration file. System properties from untrusted sources must be sanitized and validated before use.

The Java Tutorial [Campione 1996] states:

> *To maximize portability, never refer to an environment variable when the same value is available in a system property. For example, if the operating system provides a user name, it will always be available in the system property `user.name`.*

Actually, relying on environment variables is more than a portability issue. An attacker can essentially control all environment variables that enter a program using a mechanism such as the `java.lang.ProcessBuilder` class.

Consequently, when an environment variable contains information that is available by other means, including system properties, that environment variable must not be used. Finally, environment variables must not be used without appropriate validation.

Noncompliant Code Example

This noncompliant code example tries to get the user name, using an environment variable:

```
String username = System.getenv("USER");
```

First, this is a portability issue. *The Java Tutorial* Campione 1996 further suggests:

> *The way environment variables are used also varies. For example, Windows provides the user name in an environment variable called `USERNAME`, while UNIX implementations might provide the user name in `USER`, `LOGNAME`, or both.*

Second, an attacker can execute this program with the `USER` environment variable set to any value he or she chooses. The following code example does just that on a POSIX platform:

```
public static void main(String args[]) {
  if (args.length != 1) {
    System.err.println("Please supply a user name as the argument");
    return;
  }
  String user = args[0];
  ProcessBuilder pb = new ProcessBuilder();
  pb.command("/usr/bin/printenv");
  Map<String,String> environment = pb.environment();
  environment.put("USER", user);
  pb.redirectErrorStream(true);
  try {
    Process process = pb.start();
    InputStream in = process.getInputStream();
    int c;
    while ((c = in.read()) != -1) {
      System.out.print((char) c);
    }
    int exitVal = process.waitFor();
  } catch (IOException x) {
    // Forward to handler
  } catch (InterruptedException x) {
    // Forward to handler
  }
}
```

This program runs the POSIX `/usr/bin/printenv` command, which prints out all environment variables and their values. It takes a single argument string and sets the `USER` environment variable to that string. The subsequent output of the `printenv` program will indicate that the `USER` environment variable is set to the string requested.

Compliant Solution

This compliant solution obtains the user name using the `user.name` system property. The Java Virtual Machine (JVM), upon initialization, sets this system property to the correct user name, even when the `USER` environment variable has been set to an incorrect value or is missing.

```
String username = System.getProperty("user.name");
```

Risk Assessment

Untrusted environment variables can provide data for injection and other attacks if not properly sanitized.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ENV02-J	Low	Likely	Low	P9	L2

Automated Detection

Tool	Version	Checker	Description
Parasoft Jtest	10.3	**PORT.ENV**	Implemented

Android Implementation Details

On Android, the environment variable `user.name` is not used and is left blank. However, environment variables exist and are used on Android, so the rule is applicable.

Bibliography

[API 2014]	`java.lang.System`
[Campione 1996]