# IDS01-J. Normalize strings before validating them

Many applications that accept untrusted input strings employ input filtering and validation mechanisms based on the strings' character data. For example, an application's strategy for avoiding cross-site scripting (XSS) vulnerabilities may include forbidding <script> tags in inputs. Such blacklisting mechanisms are a useful part of a security strategy, even though they are insufficient for complete input validation and sanitization.

Character information in Java is based on the Unicode Standard. The following table shows the version of Unicode supported by the latest three releases of Java SE.

| Java Version | Unicode Version |
|---|---|
| Java SE 6 | Unicode Standard, version 4.0 [Unicode 2003] |
| Java SE 7 | Unicode Standard, version 6.0.0 [Unicode 2011] |
| Java SE 8 | Unicode Standard, version 6.2.0 [Unicode 2012] |

Applications that accept untrusted input should normalize the input before validating it. Normalization is important because in Unicode, the same string can have many different representations. According to the Unicode Standard [Davis 2008], annex #15, Unicode Normalization Forms:

> *When implementations keep strings in a normalized form, they can be assured that equivalent strings have a unique binary representation.*

## Noncompliant Code Example

The `Normalizer.normalize()` method transforms Unicode text into the standard normalization forms described in Unicode Standard Annex #15 Unicode Normalization Forms. Frequently, the most suitable normalization form for performing input validation on arbitrarily encoded strings is KC (NFKC) .

This noncompliant code example attempts to validate the `String` before performing normalization.

```
// String s may be user controllable
// \uFE64 is normalized to < and \uFE65 is normalized to > using the NFKC normalization form
String s = "\uFE64" + "script" + "\uFE65";

// Validate
Pattern pattern = Pattern.compile("[<>]"); // Check for angle brackets
Matcher matcher = pattern.matcher(s);
if (matcher.find()) {
  // Found black listed tag
  throw new IllegalStateException();
} else {
  // ...
}

// Normalize
s = Normalizer.normalize(s, Form.NFKC);
```

The validation logic fails to detect the `<script>` tag because it is not normalized at the time. Therefore the system accepts the invalid input.

### Compliant Solution

This compliant solution normalizes the string before validating it. Alternative representations of the string are normalized to the canonical angle brackets. Consequently, input validation correctly detects the malicious input and throws an `IllegalStateException`.

```
String s = "\uFE64" + "script" + "\uFE65";

// Normalize
s = Normalizer.normalize(s, Form.NFKC);

// Validate
Pattern pattern = Pattern.compile("[<>]");
Matcher matcher = pattern.matcher(s);
if (matcher.find()) {
  // Found blacklisted tag
  throw new IllegalStateException();
} else {
  // ...
}
```

## Risk Assessment

Validating input before normalization affords attackers the opportunity to bypass filters and other security mechanisms. It can result in the execution of arbitrary code.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
|---|---|---|---|---|---|
| IDS01-J | High | Probable | Medium | P12 | L1 |

## Automated Detection

| Tool | Version | Checker | Description |
|---|---|---|---|
| The Checker Framework | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| Fortify | 1.0 | **Process_Control** | Implemented |

## Related Guidelines

| | |
|---|---|
| ISO/IEC TR 24772:2013 | Cross-site Scripting [XYT] |
| MITRE CWE | CWE-289, Authentication bypass by alternate name<br>CWE-180, Incorrect behavior order: Validate before canonicalize |

## Android Implementation Details

Android apps can receive string data from the outside and normalize it.

## Bibliography


| | |
|---|---|
| [API 2006] | Java Platform, Standard Edition 6 API Specification |
| [Davis 2008] | |
| [Weber 2009] | Exploiting Unicode-enabled Software |