IDS11-J. Perform any string modifications before validation

It is important that a string not be modified after validation has occurred because doing so may allow an attacker to bypass validation. For example, a program may filter out the `<script>` tags from HTML input to avoid cross-site scripting (XSS) and other vulnerabilities. If exclamation marks (!) are deleted from the input following validation, an attacker may pass the string `"<scr!ipt>"` so that the validation check fails to detect the `<script>` tag, but the subsequent removal of the exclamation mark creates a `<script>` tag in the input.

A programmer might decide to exclude many different categories of characters. For example, *The Unicode Standard* [Unicode 2012] defines the following categories of characters, all of which can be matched using an appropriate regular expression:

Abbr	Long	Description
Cc	Control	A C0 or C1 control code
Cf	Format	A format control character
Cs	Surrogate	A surrogate code point
Co	Private_Use	A private-use character
Cn	Unassigned	A reserved unassigned code point or a noncharacter

Other programs may remove or replace any character belonging to a uniquely defined set of characters. Any string modifications must be performed before the string is validated.

Noncompliant Code Example (Noncharacter Code Points)

In some versions of *The Unicode Standard* prior to version 5.2, conformance clause C7 allows the deletion of noncharacter code points. For example, conformance clause C7 from Unicode 5.1 [Unicode 2007] states:

> *C7. When a process purports not to modify the interpretation of a valid coded character sequence, it shall make no change to that coded character sequence other than the possible replacement of character sequences by their canonical-equivalent sequences or the deletion of noncharacter code points.*

According to Unicode Technical Report #36, Unicode Security Considerations, Section 3.5, "Deletion of Code Points" [Davis 2008b]:

> *Whenever a character is invisibly deleted (instead of replaced), such as in this older version of C7, it may cause a security problem. The issue is the following: A gateway might be checking for a sensitive sequence of characters, say "delete". If what is passed in is "deXlete", where X is a noncharacter, the gateway lets it through: the sequence "deXlete" may be in and of itself harmless. However, suppose that later on, past the gateway, an internal process invisibly deletes the X. In that case, the sensitive sequence of characters is formed, and can lead to a security breach.*

The `filterString()` method in this noncompliant code example normalizes the input string, validates that the input does not contain a `<script>` tag, and then removes any noncharacter code points from the input string. Because input validation is performed before the removal of any noncharacter code points, an attacker can include noncharacter code points in the `<script>` tag to bypass the validation checks.

```
import java.text.Normalizer;
import java.text.Normalizer.Form;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class TagFilter {
  public static String filterString(String str) {
    String s = Normalizer.normalize(str, Form.NFKC);

    // Validate input
    Pattern pattern = Pattern.compile("<script>");
    Matcher matcher = pattern.matcher(s);
    if (matcher.find()) {
      throw new IllegalArgumentException("Invalid input");
    }

    // Deletes noncharacter code points
    s = s.replaceAll("[\\p{Cn}]", "");
    return s;
  }

  public static void main(String[] args) {
    // "\uFDEF" is a noncharacter code point
    String maliciousInput = "<scr" + "\uFDEF" + "ipt>";
    String sb = filterString(maliciousInput);
    // sb = "<script>"
  }
}
```

Compliant Solution (Noncharacter Code Points)

This compliant solution replaces the unknown or unrepresentable character with Unicode sequence `\uFFFD`, which is reserved to denote this condition. It also performs this replacement before doing any other sanitization, in particular, checking for `<script>`, to ensure that malicious input cannot bypass filters.

```
import java.text.Normalizer;
import java.text.Normalizer.Form;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class TagFilter {

  public static String filterString(String str) {
    String s = Normalizer.normalize(str, Form.NFKC);

    // Replaces all noncharacter code points with Unicode U+FFFD
    s = s.replaceAll("[\\p{Cn}]", "\uFFFD");

    // Validate input
    Pattern pattern = Pattern.compile("<script>");
    Matcher matcher = pattern.matcher(s);
    if (matcher.find()) {
      throw new IllegalArgumentException("Invalid input");
    }
    return s;
  }
  public static void main(String[] args) {
    // "\uFDEF" is a non-character code point
    String maliciousInput = "<scr" + "\uFDEF" + "ipt>";
    String s = filterString(maliciousInput);
    // s = <scr?ipt>
  }
```

According to Unicode Technical Report #36, Unicode Security Considerations [Davis 2008b], "`U+FFFD` is usually unproblematic, because it is designed expressly for this kind of purpose. That is, because it doesn't have syntactic meaning in programming languages or structured data, it will typically just cause a failure in parsing. Where the output character set is not Unicode, though, this character may not be available."

Risk Assessment

Validating input before removing or modifying characters in the input string can allow malicious input to bypass validation checks.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
IDS11-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)
Parasoft Jtest	2020.2	**BD.SECURITY.VPPD**	Validate all dangerous data

Related Guidelines

MITRE CWE	CWE-182, Collapse of Data into Unsafe Value

Bibliography

[API 2006]	
[Davis 2008b]	Section 3.5, "Deletion of Noncharacters"
[Seacord 2015]	IDS11-J. Perform any string modifications before validation LiveLesson
[Unicode 2007]	
[Unicode 2011]	
[Weber 2009]	"Handling the Unexpected: Character-deletion" (slides 72–74)