L2/20-202

No U+FFFD Generation for Zero-Length Content between ISO-2022-JP Escape Sequences

Henri Sivonen (hsivonen@mozilla.com)

2020-08-06

UTR #36: Unicode Security Considerations says[1], in part:

> Character encoding conversion must also not simply skip an illegal input byte sequence. Instead, it must stop with an error or substitute a replacement character (such as U+FFFD (�) REPLACEMENT CHARACTER) or an escape sequence in the output. (See also *Section 3.5 Deletion of Code Points*.) It is important to do this not only for byte sequences that encode characters, but also for unrecognized or "empty" state-change sequences. For example:
>
> […]
>
> - ISO-2022 shift sequences without text characters before the next shift sequence. The formal syntaxes for HZ and most CJK ISO-2022 variants require at least one character in a text segment between shift sequences. Security software written to the formal specification may not detect malicious text (for example, "delete" with a shift-to-double-byte then an immediate shift-to-ASCII in the middle).

Background

Previously, Gecko, the engine of Firefox and Thunderbird, used a character encoding conversion library called uconv, which did not implement the above advice and, like Internet Explorer, did not generate an U+FFFD when two ISO-2022-JP escape sequences occurred without any content between them.

In Firefox 56, Gecko switched form uconv to encoding_rs, which is a newly-written character encoding conversion library that generates U+FFFD when no content occur between ISO-2022-JP escape sequences, because the above advice from UTR #36 is embodied in the algorithm[2] specified in the WHATWG Encoding Standard. Instead of matching Internet Explorer, this behavior matches Chrome and Safari, whose behavior I believe to arise from upstream ICU behavior.

1 https://www.unicode.org/reports/tr36/#Some_Output_For_All_Input

2 https://encoding.spec.whatwg.org/review-drafts/2020-06/#iso-2022-jp-decoder

After the change, the U+FFFD generation has been reported as a bug in the email context both when handling email subject[3] and when handling email body[4].

The Problem

The problem is that the requirement to generate U+FFFD when no content occurs between ISO-2022-JP escape sequences gives ISO-2022-JP the unusual property that the concatenating two output from a conforming ISO-2022-JP encoder and decoding the result does yield the same output as independently decoding the two ISO-2022-JP encoder output and then concatenating the results.

The specific case causing problems under concatenation is a transition *to* the ASCII state followed immediately by a transition *away form* the ASCII state.

ISO-2022-JP is the only encoding specified in the WHATWG Encoding Standard that has this unusual property.

Is the case of email subjects, the problem was addressed by treating each MIME *encoded-word*[5] in a header as an independent ISO-2022-JP stream. That is, each one is decoded independently and the result is concatenated after decoding.

Still, the U+FFFD generation requirement when there is no content between ISO-2022-JP escape sequences poses an interoperability problem when decoding the body of emails that have been generated using a mechanism that has involved concatenating ISO-2022-JP encoder outputs.

Effectiveness U+FFFD Generation as a Security Measure

The example given as motivation for the U+FFFD generation requirement in UTR #36 is: “Security software written to the formal specification may not detect malicious text (for example, "delete" with a shift-to-double-byte then an immediate shift-to-ASCII in the middle).”

However, the requirement is ineffective as a measure ensuring that ASCII strings cannot be masked from security software operating on bytes without performing ISO-2022-JP decoding.

Instead of injecting a transition to two-byte mode immediately followed by a transition back to ASCII, the attacker could inject a transition from the ASCII state to the ASCII state between each character in the string to be masked or could insert transitions from the ASCII state to the Roman state and vice versa to make the string to be masked alternate between the ASCII and Roman states.

3 https://bugzilla.mozilla.org/show_bug.cgi?id=1374149

4 https://bugzilla.mozilla.org/show_bug.cgi?id=1508136

5 https://tools.ietf.org/html/rfc2047#section-2

It seems that what UTR #36 now says is ineffective. It would make sense to either give up completely and to remove the advice to generate U+FFFD for zero-length content between ISO-2022-JP escape sequences or to also address the other kind of state transitions that can mask ASCII strings.

The main problem with the latter is that it would require detecting unnecessary transitions between the ASCII and Roman states. If a legitimate encoder has generated these transitions when not strictly necessary, a new U+FFFD generation interoperability problem could occur.

End state 1: Drop ISO-2022-JP State Transition-Related U+FFFD Generation Completely

Upside: No user-unwanted U+FFFD generation for non-malicious inputs. Simple specification and implementation.

Downside: Unlike with ASCII-compatible encodings, including multibyte ones like Shift_JIS, EUC-JP, EUC-KR, GBK and Big5, a scan for bad ASCII strings on the pre-decode byte level is meaningless in terms of the security properties of the post-decode Unicode data.

End state 2: Drop U+FFFD Generation for Zero-Length Content in the ASCII State and Add U+FFFD Generation for Other Unnecessary Transitions

Upside: Upholds the security properties that UTR #36 guidance tries to uphold while removing the interoperability problem in the case of concatenated conforming encoder outputs.

Downside: Generates U+FFFD if the ISO-2022-JP encoder has chosen to use the Roman state when not logically necessary or has chosen to switch to or from the Roman state earlier than logically necessary. Complexity of specification and implementation.

Details

Generate U+FFFD if:

- A state transition was made such that the previous state had no content and the previous state was not the ASCII state. (I.e. stop generating U+FFFD if the zero-length state is the ASCII state.)
- A state transition to the ASCII state was preceded by the Roman state and the next byte was not 0x5C, 0x7E or the end of the stream.

- A state transition to the Roman state was made and the next byte was not 0x5C, 0x7E, 0x1B or the end of the stream. (0x1B is on this list to avoid a case where both this rule and the first rule would apply at the same time resulting in two U+FFFDs.)

Commonality of the End States

It's unclear if end state 2 is achievable considering the expectations it places on how the Roman state is used. If it isn't achievable, it would make sense to go all the way to end state 1. However, both end states have a thing in common that addresses the problem that users see at present. Both involve stopping generating U+FFFD if the state with zero-length content is the ASCII state.

Conclusion

I suggest adopting End state 1 and refining the UTR #36 advice about ISO-2022-JP to exclude zero-lengh states from the U+FFFD generation advice. That is, a transition to a state followed immediately by another transition should not generate U+FFFD.

Security software should be advised to perform actual decoding before security analysis for encodings that rely on states entered into using escape sequences. After all, trying to analyze the bytes without decoding them does not work at all e.g. for UTF-7.