When generating a random password, the result must still be a valid string

devblogs.microsoft.com/oldnewthing/20180713-00

July 13, 2018

Raymond Chen

A customer had a problem with auto-generated random passwords. Their password generator generated a string by choosing each character randomly with a code unit from U+0001 to U+FFFF. (They avoided U+0000 because that is the string terminator.) They didn't mind that the resulting passwords were untypeable, because the passwords were going to be entered programmatically.

Things seemed to work great. One computer created the account with the untypeable password, and another computer was able to log in with that password.

But occasionally, they would find that the first computer would create the account, but the second computer couldn't use it to sign in. If they re-ran the password generator, then everything worked again. If they went back to the original password, it stopped.

They were occasionally generating haunted passwords.

If you take a bunch of randomly-generated code units, the result may not be a legal Unicode string. This is true for UTF-16LE (which is the default encoding used by Windows) as well as UTF-8.

What is going on is that occasionally, the random number generator will produce an invalid Unicode string, like say, a high surrogate not followed by a low surrogate. When the account is created locally, the UTF-16LE string is passed directly to the underlying service, which creates the account with the specified password as-is.

The string is then transmitted to the other computer, and the other computer tries to sign in with that password. However, the network protocol for the service specifies that the password is encoded as UTF-8 before being hashed or encrypted or whatever it is that network protocols do to protect passwords.

The problem is that an invalid UTF-16LE string cannot be converted to Unicode code points, and therefore cannot be re-encoded as UTF-8 for transmission on the wire. At best, you get U+FFFD REPLACEMENT CHARACTER, which says "Um, there was something here, but it

wasn’t a valid Unicode code point, so I have no way of expressing it.”

The password goes out over the wire, and the server receives the UTF-8 string and transcodes it back to UTF-16LE, and the strings don’t match because invalid strings do not round trip from UTF-16LE to UTF-8 and back.

The solution to the problem is to stop generating garbage strings that aren’t even legal. They can generate the same amount of random data (preserving entropy), but convert it to Unicode via an encoding like base64 which is guaranteed to produce a legal string.

Raymond Chen

Follow