A program to detect mojibake that results from a UTF-8-encoded file being misinterpreted as code page 1252

devblogs.microsoft.com/oldnewthing/20190701-00

July 1, 2019

Raymond Chen

It is not uncommon to see UTF-8-encoded data misinterpreted as code page 1252, due to file content lacking an explicit encoding declaration, such as you might encounter with the Resource Compiler. So let's write a program to detect that specific kind of mojibake.

The first observation is that code page 1252 agrees with Unicode's first 256 code points, with the exception of the range U+0080 through U+009F, which code page 1252 assigns to various types of punctuation. Therefore, the conversion is *almost* trivial.

```
wchar_t To1252Table[32];

void Init1252Table()
{
    char as8bit[32];
    for (int i = 0; i < 32; i++) {
        as8bit[i] = (char)(i + 0x80);
    }
    MultiByteToWideChar(1252, 0, as8bit, ARRAYSIZE(as8bit),
                        To1252Table, ARRAYSIZE(To1252Table));
}

BYTE To1252(wchar_t ch)
{
    if (ch < 0x100) return (BYTE)ch;
    for (int i = 0; i < ARRAYSIZE(To1252Table); i++) {
        if (To1252Table[i] == ch) return (BYTE)(i + 0x80);
    }
    return 0;
}
```

We start by using the `MultiByteToWideChar` function to obtain the reverse conversion table for those troublesome 32 bytes. If the code point is below U+0100, then we just throw away the high-order bits and declare victory. Otherwise, we look inside our table, and if we find a match, we convert the index into the corresponding 1252 character. If that fails, then we declare failure.

There is a bit of a hole here: Unicode code points U+0080 through U+009F are erroneously converted to 1252 characters 0x80 through 0x9F. But for our purposes, this is close enough.

```
BOOL CALLBACK StringEnumProc(
    HMODULE module,
    LPCWSTR type,
    LPWSTR name,
    LONG_PTR param)
{
    PCWSTR filename = (PCWSTR)param;
    auto hrsrc = FindResource(module, name, type);
    auto res = LoadResource(module, hrsrc);
    int base = (int)((INT_PTR)name * 16);
    auto p = reinterpret_cast<wchar_t*>(LockResource(res));
    for (int i = 0; i < 16; i++) {
        // See if there is anything that looks mis-encoded.
        bool isSuspicious = false;
        for (int j = 0; j < *p; j++) {
            if (p[j + 1] == 0xFFFD) {
                // Ugh, REPLACEMENT CHARACTER.
                // Original *.rc was corrupted.
                isSuspicious = true;
                break;
            } else {
                BYTE ch = To1252(p[j + 1]);
                if (ch > 0x7f) {
                    // Does this look like UTF-8?
                    if ((ch & 0xE0) == 0xC0 &&
                        (To1252(p[j + 2]) & 0xC0) == 0x80) {
                        isSuspicious = true;
                        break;
                    }
                    if ((ch & 0xf0) == 0xE0 &&
                        (To1252(p[j + 2]) & 0xC0) == 0x80 &&
                        (To1252(p[j + 3]) & 0xC0) == 0x80) {
                        isSuspicious = true;
                        break;
                    }
                    if ((ch & 0xf8) == 0xF0 &&
                        (To1252(p[j + 2]) & 0xC0) == 0x80 &&
                        (To1252(p[j + 3]) & 0xC0) == 0x80 &&
                        (To1252(p[j + 4]) & 0xC0) == 0x80) {
                        isSuspicious = true;
                        break;
                    }
                }
            }
        }
        if (isSuspicious) {
            printf("%ls: %d: ", filename, base + i);
            for (int j = 0; j < *p; j++) {
                wchar_t ch = p[j + 1];
                if (ch < 0x80) {
                    putwc(p[j + 1], stdout);
                } else {
```

```
                    printf("\\u%04x", ch);
                }
            }
            printf("\n");
        }
        p += *p + 1;
    }
    return true;
}

void ProcessFile(PCWSTR filename)
{
    auto module = LoadLibraryEx(filename,
                    nullptr, LOAD_LIBRARY_AS_DATAFILE);
    if (!module) {
        fprintf(stderr, "Error %d loading %ls\n",
                GetLastError(), filename);
    }
    EnumResourceNames(module, RT_STRING,
                    StringEnumProc, (LONG_PTR)filename);
    FreeLibrary(module);
}
```

To process a file, we load it as a data file, and then enumerate the `RT_ STRING` resources out of it.

We learned some time ago that string resources are stored in bundles of 16 strings each, with each string in the bundle prefixed by its length in code points. We walk through each string of the bundle, and look for suspicious strings.

If we see a U+FFFD, then that's a SUBSTITUTION CHARACTER,[1] which means that the original file contained an encoding error, and there as obviously a problem with the original resource file.

Otherwise, we try to reverse-decode the character into code page 1252, and if it looks like the start of a UTF-8 multi-byte sequence, we peek ahead at the next few characters, and if they also reverse-decode into the completion of a UTF-8 multi-byte sequence, then we declare that the string is suspicious.

If we find a suspicious string, we print its resource ID and the string itself, taking care to escape any character outside the ASCII character set so it can be easier to identify in the original resource file.

Feed a file to this function, and it'll report any strings that it thinks may have been the result of UTF-8-as-1252 mojibake.

[1] I find it oddly quaint that Unicode code point names are written in ALL CAPS. I don't know why, but I never asked either.

Raymond Chen

Follow