Fun with Unicode
- an overview about Unicode dangers

by Thomas Skora

Overview

- Short Introduction to Unicode/UTF-8
- Fooling charset detection
- Ambigiuous Encoding
- Ambigiuous Characters
- Normalization overflows your buffer
- Casing breaks your XSS filter
- Unicode in domain names – how to short payloads
- Text Direction

Unicode/UTF-8

- Unicode = Character set
- Encodings:
  - UTF-8: Common standard in web, …
  - UTF-16: Often used as internal representation
  - UTF-7: if the 8th bit is not safe
  - UTF-32: yes, it exists...

UTF-8

- Often used in Internet communication, e.g. the web.
- Efficient: minimum length 1 byte
- Variable length, up to 7 bytes (theoretical).
- Downwards-compatible: First 127 chars use ASCII encoding
- 1 Byte: 0xxxxxxx
- 2 Bytes: 110xxxxx 10xxxxxx
- 3 Bytes: 1110xxxx 10xxxxxx 10xxxxxx
- ...got it? ;-)

UTF-16

- Often used for internal representation: Java, .NET, Windows, …
- Inefficient: minimum length per char is 2 bytes.
- Byte Order? Byte Order Mark! → U+FEFF
  - BOM at HTML beginning overrides character set definition in IE.
- Y\x00o\x00u\x00 \x00k\x00n\x00o\x00w\x00 \x00t\x00h\x00i\x00s\x00?\x00

UTF-7

- Unicode chars in not 8 bit-safe environments. Used in SMTP, NNTP, …
- Personal opinion: browser support was an inside job of the security industry.
- Why? Because:
  <script>alert(1)</script> ==
  +Adw-script+AD4-alert(1)+ADw-/script+AD4-
- Fortunately (for the defender) support is dropped by browser vendors.

Byte Order Mark

- U+FEFF
- Appears as: ï»¿
- W3C says: BOM has priority over declaration
  - IE 10+11 just dropped this insecure behavior, we should expect that it comes back.
  - http://www.w3.org/International/tests/html-css/character-encoding/results-basics#precedence
  - http://www.w3.org/International/questions/qa-byte-order-mark.en#bomhow
- If you control the first character of a HTML document, then you also control its character set. → XSS!

Ambiguous Encoding

- UTF-8 demands shortest encoding, but parsers/filters possibly do something different.
- Example „<“:
  - 0x3C = 00111100
  - 0xC0 0xBC = 11000000 10111100 = 000 00111100
  - Second byte encodes leading zeros!
  - Now think about dumb, encoding-unaware XSS filters which filter 0x3C, but not the overlong sequences.
  - Fortunately browsers ignore it, so further steps on server side are needed.
  - Test it with Burp Intruder, Payload Type „Illegal Unicode“

Ambiguous Characters

- Ä != Ä
  - Ä = U+00C4
  - = LATIN CAPITAL LETTER A WITH DIAERESIS (= NFC)
  - Ä = U+0041 + U+0308
    = LATIN CAPITAL LETTER A + COMBINING DIAERESIS (= NFD)
- Normalize before compare
- Normalization Forms:
  http://www.unicode.org/reports/tr15/#Norm_Forms
- Useful Link:
  http://software.hixie.ch/utilities/cgi/unicode-decoder/utf8-decoder

Decomposition

- Previous examples decomposition expands to 2 characters
- U+FDFA: ARABIC LIGATURE SALLALLAHOU ALAYHE WASALLAM
- This is: „صلى الله عليه وسلم“
- 18 characters!
- Remember while writing native code: some Unicode normalization blows your input up! → Buffer Overflows!
- http://fallout.skora.net/security/unicode-lists.html

Upper-/Lower-Casing

- XSS-“filters“ seen in audits: first filtering „SCRIPT“ etc., then upper casing
- This breaks our „<script src=...></script>“ attack payload to < SRC=...></>
- Unicode rushes to the rescue!
  - Some (for us germans) strange chars case to latin chars
  - uc(ſ)=S (U+017f), uc(ı)=I (U+0131)
  - <ſcrıpt ſrc=...></ſcrıpt> => <SCRIPT SRC=...></SCRIPT>
- http://prompt.ml/9
- http://fallout.skora.net/security/unicode-lists.html

Strange chars in URLs

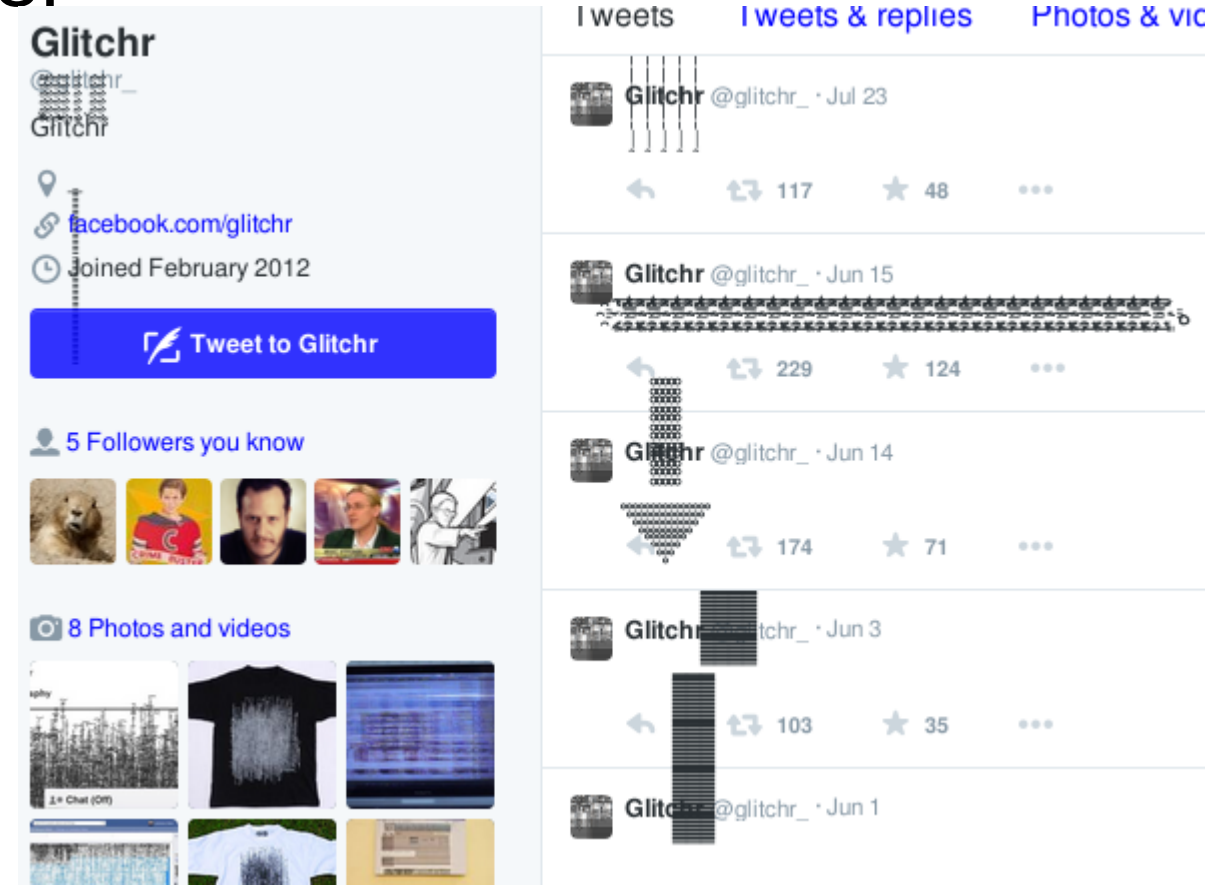
- These are single characters:
  ffi,Rs,FAX,TEL, ℳℓ , cc , mol ,...many others
- In URLs, browsers convert them to:
  ffi,Rs,FAX,TEL,ml,cc,mol etc.
- So <script src="//ffi. ℳℓ" > requests script code from ffi.ml.
  - URL shrinks from 6 to 3 chars.
- Also nice for phishing attacks, like Punycode domains:
  - xn--pypal-4ve = paypal
    = p + CYRILLIC SMALL LETTER A + aypal

Text Direction

- <RIGHT-TO-LEFT OVERRIDE> + gpj.exe = exe.jpg
- Used by spammers to hide content from filters.
- Circumvent wordfilters: <RIGHT-TO-LEFT OVERRIDE> + elohssa = asshole

Rendering Glitches

- Unicode combining characters which break out of their expected rendering context.
- @glitchr_ on Twitter



Conclusion

- Unicode adds much complexity
- New attack surfaces
- Links:
  - Unicode Security Considerations: http://www.unicode.org/reports/tr36/
  - Unicode Security Mechanisms: http://www.unicode.org/reports/tr39/
  - Unicode Security Guide: https://github.com/cweb/unicode-security-guide