Unraveling Unicode:
A Bag of Tricks for Bug Hunting

Black Hat USA
July 2009

Chris Weber
www.lookout.net
chris@casabasecurity.com

Casaba Security
Can you tell the difference?

http://www.google.com/
How about now?

Spoofle

Hi Mom!
The Transformers

*When good input turns bad*

```html
<scri*p>t</scri*p>
```

becomes

```html
<scri*p>t</scri*p>
```
Agenda
Unicode Transformations

Agenda

• Unicode crash course
• Root Causes
• Attack Vectors
• Tools
  – Find Unicode issues in Web-testing
  – Visual Spoofing Detection
Unicode Transformations

Agenda

• Unicode crash course
• Root Causes
• Attack Vectors
• Tools
Unicode Crash Course

The Unicode Attack Surface

- End users
- Applications
- Databases
- Programming languages
- Operating Systems
Unicode Crash Course

Unthink it
A large and complex standard

- code points
- encodings
- categorization
- normalization
- binary properties
- case mapping
- conversion tables
- bi-directional properties
- canonical mappings
- decomposition types
- case folding
- best-fit mapping
- 17 planes
- private use ranges
- script blocks
- escapings
Unicode Crash Course

*Code pages and charsets*

Shift_jis
Gb2312
ISCII
Windows-1252
ISO-8859-1
EBCDIC 037
Unicode Crash Course

Ad Infinitum

• Unicode can represent them all
• ASCII range is preserved
  – U+0000 to U+007F are mapped to ASCII
Unicode Crash Course

*Code points*

- Unicode 5.1 uses a **21-bit scalar** value with space for over 1,100,000 **code points**:

  U+0000 to U+10FFFF
Unicode Crash Course

Code Points

A = U+0041

Every character has a unique number
Unicode Crash Course

Category: Lu (Letter, Uppercase)
- ToLower: U+0061
- ToUpper: U+0041
- Script: Basic Latin

Latin capital letter A
- Decomposition Type: none
- Mapping: none
- Binary Properties:
  - Hex Digit
  - Alphabetic
  - Lowercase
  - ID Start ...

U+0041
Unicode Crash Course

U+017F

Category: Li (Letter, Lowercase)
ToLower: U+017F
ToUpper: U+0053
Script: Latin Extended-A

Latin small letter long S
Decomposition Type: <compat>
Mapping: U+0073
Binary Properties:
Alphabetic
Lowercase
ID Start ...
Unicode Crash Course

Encodings

UTF-8
  – variable width **1 to 4 bytes** (*used to be 6*)

UTF-16
  – Endianess
  – Variable width **2 or 4 bytes**
  – Surrogate pairs!

UTF-32
  – Endianess
  – Fixed width **4 bytes**
  – Fixed mapping, no algorithms needed
Unicode Crash Course

Encodings and Escape sequences

U+FF21 FULLWIDTH LATIN CAPITAL LETTER A

\%EF\%BC\%A1

&amp;#xFF21;

&amp;#65313;

\xEFl\xBC\xA1

\uFF21
Unicode Transformations

Agenda

• Unicode crash course
• Root Causes
• Attack Vectors
• Tools
Unicode Transformations

Agenda

• Unicode crash course
• Root Causes
• Attack Vectors
• Tools
Unicode Transformations

Overview

• Unicode crash course

• Root Causes
  – Visual Spoofing and IDN’s
  – Best-fit mappings
  – Normalization
  – Overlong UTF-8
  – Over-consumption
  – Character substitution
  – Character deletion
  – Casing
  – Buffer overflows
  – Controlling Syntax
  – Charset transformations
  – Charset mismatches

• Tools
Root Causes

Visual Spoofing

• Over 100,000 assigned characters
• Many lookalikes within and across scripts
Attack Vectors

*IDN homograph attacks*

Some browsers allow .COM IDN’s based on script family

– (Latin has a big family)
Attack Vectors

*IDN homograph attacks*

Safari
Attack Vectors

*IDN homograph attacks*

Opera
Attack Vectors

*IDN homograph attacks*

www.google.com is not www.google.com

- Latin U+0069
- Latin U+0261

$$gg$$
Root Causes

The state of International Domain Names

ICANN guidelines v2.0

- Inclusion-based
- Script limitations
- Character limitations

Deny-all default seems to be the right concept.

A script can cross many blocks. Even with limited script choices, there’s plenty to choose from.

Great for domain labels, but sub domain labels still open to punctuation and syntax spoofing.
Attack Vectors

*Visual spoofing Vectors*

- Non-Unicode attacks
- Confusables
- Invisibles
- Problematic font-rendering
- Manipulating Combining Marks
- Bidi and syntax spoofing
Attack Vectors

Non-Unicode homograph attacks

rn can look like m in certain fonts

www.mullets.com is not www.rnullets.com

Latin
U+006D

Latin
U+0073 U+006E
Attack Vectors

Non-Unicode homograph attacks

Are you using mono-width fonts?

0 and 0
1 and 1
5 and S
Attack Vectors

Non-Unicode homograph attacks

Classic long URL’s

http://login.facebook.invitation.videomessageid-h048892r39.sessionnfbid.com/home.htm?/disbursements/
Attack Vectors

*Single-script and The Confusables*

www.apple.com
// All Latin using Latin small letter Alpha ‘ɑ’

www.facebook.com
// Mixed Latin/Greek with lunate sigma symbol ‘ᴄ’

www.абс.com
// All Cyrillic ‘абс’
Attack Vectors

*IDN homograph attacks*

Browsers whitelist .ORG
Attack Vectors

*IDN homograph attacks*

Others don’t necessarily but...
Attack Vectors

*IDN homograph attacks*

www.mozilla.org is not www.mozília.org

- Latin U+0069
- Latin U+00ED
Attack Vectors

*IDN Syntax Spoofing with / lookalikes*

http://www.google.com/\path/\file?.nottrusted.org

(This case doesn’t work anymore)
Attack Vectors

*IDN Syntax Spoofing with / lookalikes*

http://www.google.com/path/file.nottrusted.org

(Normalized to a / U+002F)
Attack Vectors

IDN Syntax Spoofing with / lookalikes

http://www.google.com/path/file.nottrusted.org

Katakana No U+FF89

(However punctuation not required...)

http://www.google.com/path/file.nottrusted.org
Attack Vectors

The Invisibles
Attack Vectors

Visual Spoofing with Bidi Explicit Directional Overrides

![Visual Spoofing Example](image)
Root Causes

*Best-fit mappings*

Commonly occur in charset transformations and even innocuous API’s

**Impact:** Filter evasion, Enable code execution

When $\sigma$ becomes $\varsigma$

U+03C3 GREEK SMALL LETTER SIGMA

When ' becomes '

U+2032 PRIME
Root Causes

Guidance for Best-Fit mappings

- Scrutinize character/charset manipulation API’s
- **Use** EncoderFallback with System.Text.Encoding
- **Set** WC_NO_BEST_FIT_CHARS flag with WideCharToMultiByte()
- **Use** Unicode end-to-end
Case Study: Social Networking

*Best-fit mappings*

- A popular social networking site in 2008
- Implemented complex filtering logic to prevent XSS
  - **Attack**: Filter evasion, code execution
  - **Exploit**: Bypass filtering logic with best-fit mappings to leverage cross-site scripting
  - **Root Cause**: best-fit mappings
Case Study: Social Networking

*Best-fit mappings*

`-moz-binding()`

was not allowed, but....

`-[U+ff4d]oz-binding()`

would best-fit map!
Root Causes

Normalization

Normalizing strings after validation is dangerous

Impact: Filter evasion, Enable code execution
Root Causes

Normalization

\[ \text{i becomes } \text{I} + \cdot \]

U+0130  U+0049  U+0307
Root Causes

*Normalization*

But are there dangerous characters?

You bet... with NFKC and NFKD you could control HTML or other parsing

\[ \leftarrow \text{becomes} \rightarrow \]

U+FE64  U+003C
Root Causes

Normalization

\(<\text{becomes}\)<

U+FE64  U+003C

toNFKC("<script>"") = "<script>"
Root Causes

Guidance for Normalization

Normalize strings **before** validation

**NFKC** first defense against Visual spoofing
Root Causes

Non-shortest form UTF-8

Non-shortest or overlong UTF-8

Impact: Filter evasion, Enable code execution

Application gets %C0%A7
OS/Framework sees %27
Database gets '
Root Causes

*Guidance for Non-shortest form UTF-8*

- Unicode **specification forbids**
  - *Generation* of non-shortest form
  - *Interpretation* of non-shortest form for BMP
- Validate UTF-8 encoding (throw on error)
Attack Vectors

Directory traversal

How many ways can you say ../
Normalization compatibility forms:
U+2024 U+2024 U+FF0F
%E2 %80 %A4 %E2 %80 %A4 %EF %83 %BF

UTF-8:
U+002E U+002E U+002F
%2E %2E %2F

Best-fit mapping Windows-1252:
U+FF0E U+FF0E U+2215
%EF %BC %8E %EF %BC %8E %E2 %88 %95

UTF-8 overlong:
U+002E U+002E U+002F
%C0 %AE %C0 %AE %C0 %AF%
Root Causes

Handling the Unexpected

• Unassigned code points
  – U+2073

• Illegal code points
  – Half a surrogate pair

• Code points with special meaning
  – U+FEFF is the BOM

• **Impact**: Filter evasion, Enable code execution
Root Causes
Handling the Unexpected: Over-consumption

Over-consuming **ill-formed byte sequences**

* Big problem with MBCS lead bytes

\[<41 \text{ C2 } 3E \text{ 41}> \text{ becomes } <41 \text{ 41}>\]
Root Causes

Handling the Unexpected: Over-consumption

\[
\text{<img src="\#[0xC2]\" onerror="alert(1)"\>}
\]

becomes

\[
\text{<img src="\#\" onerror="alert(1)"\>}
\]
Root Causes

Handling the Unexpected: Character-substitution

Correcting insecurely rather than failing

– Substituting a ‘.’ or a ‘/’ would be bad
Root Causes

Handling the Unexpected: Character-deletion

“deletion of noncharacters” (UTR-36)
Root Causes

Handling the Unexpected: Character-deletion

\[ <\text{script}> \] becomes \[ <\text{ipt}> \]
Root Causes

Solutions for Handling the Unexpected

• Fail or error

• Use U+FFFD instead
  – A common alternative is ‘?’ which can be safe
Attack Vectors

Filter evasion

• Bypass filters, WAF’s, NIDS, and validation
• Exploit delivery techniques
  – E.g. Cross-site scripting (buffer overflow of the Web)
Case Study: Apple and Mozilla

Safari and Firefox BOM consumption

- **Attack**: Filter evasion, code execution
- **Exploit**: Bypass filtering logic with specially crafted strings to leverage cross-site scripting
- **Root Cause**: Character deletion

```
<a href="java[U+FEFF]script:alert(’XSS’)">
```

Can be nastier:

```
```
A Closer Look: The BOM

**BOM**

U+FEFF

**ZERO WIDTH NO-BREAK SPACE**

(BYTE ORDER MARK)

- **Binary Properties:**
  - Default Ignorable Code Point

**UTF-8:** EF BB BF

**UTF-16LE:** FF FE

**UTF-16BE:** FE FF

**Category:** Cf [Other, Format]

**Script:** Common

**Line Break:** WJ [Word Joiner]
Root Causes

Casing

• Attackers manipulate casing operations to inject otherwise prohibited characters
• Casing can multiply the buffer sizes needed
• **Impact:** Filter evasion, Enable code execution
Root Causes

Casing

toLower(“İ”) == “i”

toLower(“script”) == “script”
Root Causes

Casing

\[ \text{len}(x) \neq \text{len} (\text{toLowerCase}(x)) \]
Root Causes

Guidance for Casing

• Perform casing operations **before** validation
• Leverage existing frameworks and API’s
  – ICU, .Net
Root Causes

Buffer Overflows

- Incorrect assumptions about string sizes (chars vs. bytes)
- Improper width calculations
- **Impact**: Enable code execution
## Root Causes

**Buffer Overflows**

### Casing - maximum expansion factors

<table>
<thead>
<tr>
<th>Operation</th>
<th>UTF</th>
<th>Factor</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower</td>
<td>8</td>
<td>1.5</td>
<td>Ä</td>
</tr>
<tr>
<td></td>
<td>16, 32</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Upper</td>
<td>8, 16, 32</td>
<td>3</td>
<td>i̇</td>
</tr>
</tbody>
</table>

Source: *Unicode Technical Report #36*
## Root Causes

### Buffer Overflows

**Normalization** - maximum expansion factors

<table>
<thead>
<tr>
<th>Operation</th>
<th>UTF</th>
<th>Factor</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFC</td>
<td>8</td>
<td>3X</td>
<td>☐</td>
</tr>
<tr>
<td></td>
<td>16, 32</td>
<td>3X</td>
<td>☧</td>
</tr>
<tr>
<td>NFD</td>
<td>8</td>
<td>3X</td>
<td>ë</td>
</tr>
<tr>
<td></td>
<td>16, 32</td>
<td>4X</td>
<td>ð</td>
</tr>
<tr>
<td>NFKC/NFKD</td>
<td>8</td>
<td>11X</td>
<td>صلى الله عليه وسلم</td>
</tr>
<tr>
<td></td>
<td>16, 32</td>
<td>18X</td>
<td></td>
</tr>
</tbody>
</table>

Source: *Unicode Technical Report #36*
Root Causes

* Guidance for Buffer Overflows*

- Know the difference between bytes and chars
- Secure coding
- Leverage existing frameworks and API’s  
  – ICU, .Net
Root Causes

*Controlling Syntax*

- White space and line breaks
  - E.g. when U+180E acts like U+0020
- Quotation marks
- **Impact:** Filter evasion, Enable code execution
Attacks and Exploits

*Controlling syntax*

- Manipulate HTML parsers and javascript interpreters
- Control protocols
Case Study: Opera

• Unicode formatter characters exploited for XSS
  – **Damage**: Filter evasion, controlling syntax
  – **Exploit**: Bypass filtering logic with specially crafted characters to leverage cross-site scripting.
  – **Root Cause**: Interpreting “white space”
  – A problem with HTML 4.0 spec?
Case Study: Opera

<a href=# onclick=alert()>"\U180E"</a>
Case Study: Opera

Category: Zs [Separator, Space]

Script: Mongolian

Line Break: GL [Non-breaking ("Glue")]

MVS
U+180E

MONGOLIAN VOWEL SEPARATOR

Binary Properties:
- White Space
- Grapheme Base
Root Causes

*Guidance for Controlling Syntax*

- Question specifications
- Be careful...
Root Causes

Specifications

1) Character stability
   – IDNA/Nameprep based on Unicode 3.2

2) Designs
   – Specs are carefully designed but not always perfect
     • This could have been a problem:
       – “When designing a markup language or data protocol, the use of U+FEFF can be restricted to that of Byte Order Mark. In that case, any U+FEFF occurring in the middle of the file can be ignored, or treated as an error.”
   – HTML 4.01
     • Defines four whitespace characters and explicitly leaves handling other characters up to implementer.
Root Causes

Charset Transformations

• Converting between charsets is dangerous
• Mapping tables and algorithms vary across platforms
• **Impact**: Filter evasion, Enable code execution, Data-loss
Root Causes

**Guidance for Charset Transformations**

- Avoid if possible
- Use Unicode as the broker
- Beware the PUA mappings
- Transform, case, and normalize prior to validation and redisplay
Root Causes

Charset Mismatches

• Some charset identifiers are ill-defined
• Vendor implementations vary
• User-agents may sniff if confused
• Attackers manipulate behavior

• Impact: Filter evasion, Enable code execution
Root Causes

Charset Mismatches

Content-Type: charset=ISO-8859-1

Attacker-controlled input

<meta http-equiv="Content-Type" content="text/html; charset=shift_jis"/>
Root Causes

Guidance for Charset Mismatches

• Force UTF-8
• Error if uncertain
Unicode Transformations

Agenda

• Unicode crash course
• Root Causes
• Attack Vectors
• Tools
Unicode Transformations

Agenda

• Unicode crash course
• Root Causes
• Attack Vectors
• Tools
Tools

• Watcher
  – Passive Web-app security testing and auditing

• Unibomber
  – XSS autopwn testing tool
Tools

Watcher – Some of the Passive Checks Included

• Unicode transformation hot-spots
• User-controlled HTML
• Cross-domain issues
• Insecure cookies
• Insecure HTTP/HTTPS transitions
• SSL protocol and certificate issues
• XSS hot-spots
• Flash issues
• Silverlight issues
• Information disclosure
## Tools

### Watcher by Casaba Security

#### Totals (Alerts, Individual Issues)

<table>
<thead>
<tr>
<th>Severity</th>
<th>Session ID</th>
<th>Type</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medium</td>
<td>14</td>
<td>Set-Cookie HTTPOnly Attribute Not Set</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Medium</td>
<td>14</td>
<td>Set-Cookie Secure Attribute Not Set</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Medium</td>
<td>14</td>
<td>Set-Cookie Loosely Scoped Domain</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Information</td>
<td>17</td>
<td>Charset not UTF-8</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>High</td>
<td>18</td>
<td>Invalid Unicode ByteStream</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>High</td>
<td>18</td>
<td>Null Bytes in ByteStream</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Medium</td>
<td>20</td>
<td>Flash allowScriptAccess Value</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Information</td>
<td>21</td>
<td>Charset not UTF-8</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Medium</td>
<td>21</td>
<td>Flash crossdomain.xml Insecure Domain Reference</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Information</td>
<td>30</td>
<td>Charset not UTF-8</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>Medium</td>
<td>30</td>
<td>Silverlight clientaccesspolicy.xml Insecure Domain Reference</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>High</td>
<td>32</td>
<td>User Controllable Location Header (Open Redirect)</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
<tr>
<td>High</td>
<td>38</td>
<td>User Controllable Charset</td>
<td><a href="http://www.nottrusted.com/watcher/CheckPassChallenge.php">www.nottrusted.com/watcher/CheckPassChallenge.php</a></td>
</tr>
</tbody>
</table>

---

**Invalid Unicode ByteStream**

**Risk: High**

An invalid UTF-8 ByteStream was detected for request:

www.nottrusted.com/watcher/CheckPassChallenge.php

The following issue(s) were identified:

1) An invalid 2 character UTF-8 ByteStream was found at byte position 481
   Invalid byte(s): C3 75
Tools

Watcher - Web-app Security Testing and Auditing

http://websecuritytool.codeplex.com
Tools

Unibomber—runtime XSS testing tool

• Deterministic testing
• Auto-inject payloads
• Unicode transformers
  — < > ‘ “, etc.
• Detect transformations and encoding hotspots