Attacking Internationalized Software

Scott Stender
scott@isecpartners.com

Black Hat
August 2, 2006

Information Security Partners, LLC
iSECPartners.com
Attacking Internationalized Software

• Introduction

• Background
  – Internationalization Basics
  – Platform Support
  – The Internationalization “Stack”

• Historical Attacks
  – Width calculation
  – Encoding attacks

• Current Attacks
  – Conversion to Unicode
  – Conversion from Unicode
  – Encoding Attacks

• Tools
  – I18Attack

• Q&A
Attacking Internationalized Software

Introduction

• **Who are you?**
  – Founding Partner of Information Security Partners, LLC (iSEC Partners)
  – Application security consultants and researchers

• **Why listen to this talk?**
  – Every application uses internationalization (whether you know it or not!)
  – A great deal of research potential

• **Platforms**
  – Much of this talk will use Windows for examples
  – *Internationalization is a cross-platform concern!*
Attacking Internationalized Software

• Introduction

• **Background**
  – Internationalization Basics
  – Platform Support
  – The Internationalization “Stack”

• **Historical Attacks**
  – Width calculation
  – Encoding attacks

• **Current Attacks**
  – Conversion to Unicode
  – Conversion from Unicode
  – Encoding Attacks

• **Tools**
  – I18Attack

• **Q&A**
Attacking Internationalized Software

Background – Internationalization Basics

• **Internationalization Defined**
  – Provides support for *potential* use across multiple languages and locale-specific preferences
  – Most of this talk will focus on character manipulation

• **Code Pages A-Plenty**
  – Single-Byte: Most pages for European languages, ISO-8859-*…
  – Multi-Byte: Japanese (Shift-JIS), Chinese, Korean
  – Unicode

• **Encodings to match A-Plenty**
  – EBCDIC, ASCII, UTF-7, UTF-8, UTF-16, UCS-2…
Attacking Internationalized Software

Background – Internationalization Basics

- Multi-Byte Character Sets
  - 0x41 = U+0041 = LATIN CAPITAL LETTER A
  - 0x81 0x8C = U+2032 = PRIME

See http://www.microsoft.com/globaldev for others
Attacking Internationalized Software

Background – Internationalization Basics

• **Unicode**
  – One code page to rule them all!
  – Current standards specify a 21-bit character space

• **Encodings vs. Code Points**
  – Code pages describe sets of points, encodings translate those points to 1s and 0s
  – Though Unicode is often associated with 8 or 16-bit chars, these are just the most common encodings
  – Many encodings available: UTF-32, UTF-16, UCS-2, UTF-8, UTF-7
  – UTF-16 surrogate pairs: U+D800 to U+DBFF high & U+DC00 to U+DFFF low
Attacking Internationalized Software

Background – Platform Support

• **Almost every platform has support for internationalization**
  – Results depend on Unicode standard supported by platform

• **Newer platforms tend to play nicer with Unicode**
  – .Net & Java use native Unicode encodings, though they can convert to others

• **Cool, I use one of those!**
  – Not so fast – you still depend on internationalization support of underlying OS, servers they interact with, etc.

*Also “Damn, they use one of those!”*
Attacking Internationalized Software

Background – Windows

• **Windows is built with Unicode at its core**
  – Most native API functions take UTF-16 strings
  – In many cases, this requires that SBCS and MBCS code pages be converted, often several times

• **Broad, generalized support though OS and applications**
  – Serves as a good example for today’s demos
  – Not all localized builds support the same code pages out of the box
  – Install language packs, and test with native builds if you *really* want coverage

• **Character set conversion has two core APIs**
  – Though we are Win32-specific here, the idea translates to other platforms
Attacking Internationalized Software

Background – Windows

- **MultiByteToWideChar – Convert to Unicode**
  - CodePage - can use default which will vary by system
  - Note all of the length specifiers!

```c
int MultiByteToWideChar(
    UINT CodePage,             // code page
    DWORD dwFlags,             // character-type options
    LPCSTR lpMultiByteStr,     // string to map
    int cbMultiByte,           // number of bytes in string
    LPWSTR lpWideCharStr,      // wide-character buffer
    int cchWideChar            // size of buffer
);
```
Attacking Internationalized Software

Background – Windows

- **WideCharToMultiByte – Convert from Unicode**
  - `dwFlags` – modifies conversion properties
    - `WC_NO_BEST_FIT_CHARS` is your friend!
  - `lpDefaultChar` – allows you to specify error character

```c
int WideCharToMultiByte(
    UINT CodePage,            // code page
    DWORD dwFlags,            // performance and mapping flags
    LPCWSTR lpWideCharStr,    // wide-character string
    int cchWideChar,          // number of chars in string
    LPSTR lpMultiByteStr,     // buffer for new string
    int cbMultiByte,          // size of buffer
    LPCSTR lpDefaultChar,     // default for unmappable chars
    LPBOOL lpUsedDefaultChar  // set when default char used
);
```
Attacking Internationalized Software

Background – *nix

- General support assumptions are hard to make
  - POSIX Locale offers some standardization
  - Many libraries and application-specific approaches fill the void

- Pushes i18n concerns “up the stack”
  - Less internationalization support offered “for free” to developers
  - For example – using non-English or non-UTF-8 characters often requires using alternate editors/shells/etc. See open18n.org.

- This is good and bad
  - Less pixie dust means that internationalization support is often intentional
  - Then again, it’s complicated, error prone, and often implemented insecurely.
Attacking Internationalized Software

Background – *nix

• **Common Utilities/Libraries that offer support**
  – International Components for Unicode – open source library, cross-language
  – iconv – common utility on most linux distros. Converts files across many encodings
  – Libiconv: API for the same
  – Roll your own – everybody else does!*

• **Standardization**
  – [www.opengroup.org](http://www.opengroup.org) – POSIX locale guidelines
  – [www.open18n.org](http://www.open18n.org) – Internationalization guidelines defined in LSB

*Please don’t!
Attacking Internationalized Software

Background – Everything Else

• **Support isn’t just from the OS**
  – Programming language
  – Virtual machines
  – Application only

• **This offers a unique attack surface**
  – Cross-OS, Language, Application Class, and Implementation
  – A great place to start is with standards that stipulate I18N support
  – In short, this hits almost every application out there
Attacking Internationalized Software

Background – The Internationalization Stack

• Every application has internationalization dependencies
  – Development platform
  – External libraries
  – Operating System
  – Application Server
  – Database Server - collations!
  – Clients
Attacking Internationalized Software

Background – The Internationalization Stack

• **Web applications**
  – Code page can be set on both HTTP request and response
  – Code page is set on first line of every XML document

• **The Default Code Page**
  – Remember CP_ACP?
  – Change system and user locales
  – Ever tried to test your app on Japanese…you’ll see why you should!
Attacking Internationalized Software

Background – The Internationalization Stack

- HTTP Parser
- XML Parser
- Application Logic
- Database Access Library
- Database
- Operating System

- Please don’t check here
- Most practical point of control for devs
- Great research potential!
Attacking Internationalized Software

• Introduction

• Background
  – Internationalization Basics
  – Platform Support
  – The Internationalization “Stack”

• Historical Attacks
  – Width calculation
  – Encoding attacks

• Current Attacks
  – Conversion to Unicode
  – Conversion from Unicode
  – Encoding Attacks

• Tools
  – I18Attack

• Q&A
Attacking Internationalized Software

*Historical Attacks*

- **Security and Internationalization** has seen some attention…
  - Chalk these up as “lesson learned,” for the most part

- **Width Calculation**
  - Conversion functions
  - Count of bytes vs. Count of characters
    - `sizeof(array) vs. sizeof(array)/sizeof(array[0])`
  - Compile-time function specifiers (`lstr*`, `tchars`)

- **Non-minimal UTF-8 encodings in NT4 IIS**
  - `http://.../web/index.html`
  - `http://.../web/../../blah`
  - `http://.../web/%2E%2E%2F%2E%2E%2F/blah`
  - `http://.../web/%C0%AE%C0%AE%C0%AF%C0%AE%C0%AE%CO%AF/blah`
Attacking Internationalized Software

• **Introduction**

• **Background**
  – Internationalization Basics
  – Platform Support
  – The Internationalization “Stack”

• **Historical Attacks**
  – Width calculation
  – Encoding attacks

• **Current Attacks**
  – Conversion to Unicode
  – Conversion from Unicode
  – Encoding Attacks

• **Tools**
  – I18Attack

• **Q&A**
Attacking Internationalized Software

Current Attacks – Conversion from Unicode

- Scenario – Validation is performed on input, later converted to locale-specific text

- Attack Class – “Eating Characters”
  - Especially damaging for any character string that “doubles up” to escape

- Eating a SQL quotation character
  - Shift-JIS MBCS Japanese Code Page
  - 0x8260 = U+FF21 = FULLWIDTH LATIN CAPITAL LETTER A
  - 0x8227 = nothing (but 0x27 is an apostrophe)
  - 0x822727 = nothing with an apostrophe
  - Converted to Unicode, this will likely become ‘?!
  - …where user =‘blah?’ or 1-1--…

Demo
Attacking Internationalized Software
*Current Attacks – Conversion to Unicode*

- **Scenario** – Validation is performed, changed to Unicode

- **Attack Class – “Character Conversion”**
  - Unicode’s character space is much larger than any locale-specific code page
  - Results in a many-to-one mapping for many characters
  - Code-page specific
  - Big reason why WC_NO_BEST_FIT_CHARS should *always* be specified

- **Sneaking an apostrophe in…**
  - U+2032 = PRIME
  - Converted to Latin-1252 it is 0x27 – Apostrophe
  - U+2032 isn’t the only apostrophe equivalent in Windows-1252!
  - Same thing happens for quotation marks, numbers, letters, etc.
  - Latin-1 isn’t the only code page, have you tried your JPN web client lately?

**Demo**
Attacking Internationalized Software

Current Attacks – Conversion to Unicode

• Attack Class – “Foiling Canonicalization”
  – Back in the day `%C0%AE` was interpreted as 0x2E or simply ‘.’
  – Unicode standard has been changed to explicitly disallow all such conversions
  – Most UTF-8 parsers today choose to omit such characters

• Attack - Directory Traversal
  – http://.../web/index.html
  – http://.../web/../../../blah
  – http://.../web/./././blah
  – ../../../ not found in input, so passed to file parser
  – File parser converts ./././ to unicode (as NtCreateFile requires)
  – Non-minimal encodings dropped - ../../../ remains

Demo
Attacking Internationalized Software

Current Attacks – Encoding Attacks

- **Attack Class – “Mistaken Identity”**
  - We have been spoiled by the most common Unicode encodings
  - Unicode is just a set of code points, encoding is up to the parser
  - UTF-8, UTF-16, and UCS-2 all resemble ASCII

- **Sneak “garbage” data past validators**
  - Most interesting characters exist in ASCII – ‘, “, <, >, =…
  - Validation routines often take advantage of the ASCII resemblance
  - Many encodings can easily bypass this approach
  - ASCII, EBCDIC, UTF7..

Demo
Attacking Internationalized Software

• Introduction

• Background
  – Internationalization Basics
  – Platform Support
  – The Internationalization “Stack”

• Historical Attacks
  – Width calculation
  – Encoding attacks

• Current Attacks
  – Conversion to Unicode
  – Conversion from Unicode
  – Encoding Attacks

• Tools
  – I18Attack

• Q&A
Attacking Internationalized Software

*Tools – I18NAttack*

- **Background**
  - Testing equivalence characters, “eaters,” alternate encodings is time consuming!
  - Goal is to provide a security-focused collection of characters and encodings that often trip up input validation routines
  - Using it is always going to be transport-dependent, but here is a tool to get you started…

- **I18NAttack**
  - HTTP POST/GET Parameter Fuzzer
  - Reference implementation for nasty character database
  - Will identify and fuzz problem characters across equivalents, unusual encodings, etc.
  - Use to bypass poor input validation

Demo
Attacking Internationalized Software

Q&A

Scott Stender

scott@isecpartners.com