Unicode Transformations: Finding Elusive Vulnerabilities

OWASP AppSecDC

November 2009

Chris Weber

chris@casabasecurity.com

 Casaba Security

What's this about?

- Visual spoofing and counterfeiting
- Text transformation attacks

What will you learn?

- Why you should care about Visual Integrity…
  - Branding
  - Identity
  - Cloud Computing – URI's!

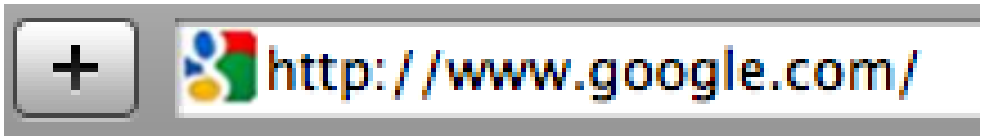
What will you learn?

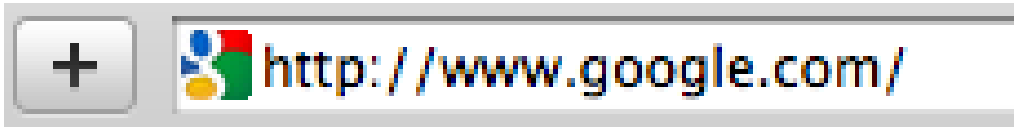
- Good techniques for finding bugs
  - Web-apps and clever XSS
  - Test cases for fuzzing

What about tools?

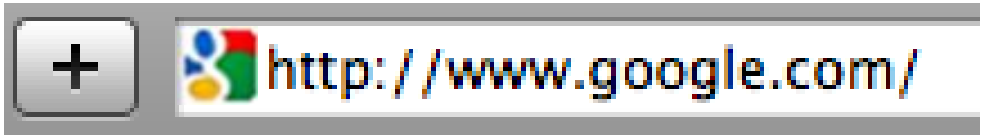
- Watcher
  - Microsoft SDL recommended tool
  - Passive Web-app testing for free
  - http://websecuritytool.codeplex.com/
- Unibomber
  - Deterministic auto-pwn XSS testing

Can you tell the difference?





How about now?



The Transformers

When good input turns bad

<script>

becomes

<script>

Agenda

Unicode Transformations

Agenda

- Unicode crash course
- Root Causes
- Attack Vectors
- Tools
  - Watcher
  - Unibomber

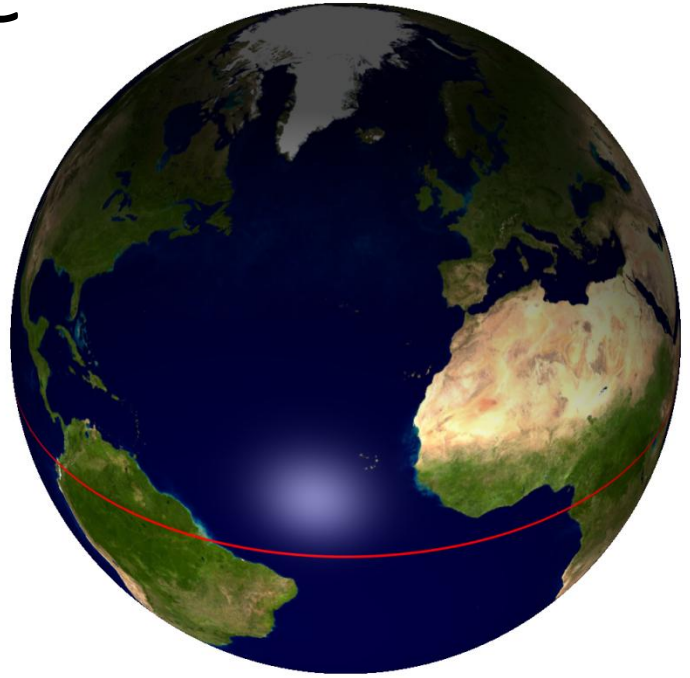
Unicode Transformations

Agenda

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

Unicode Crash Course

What is Unicode

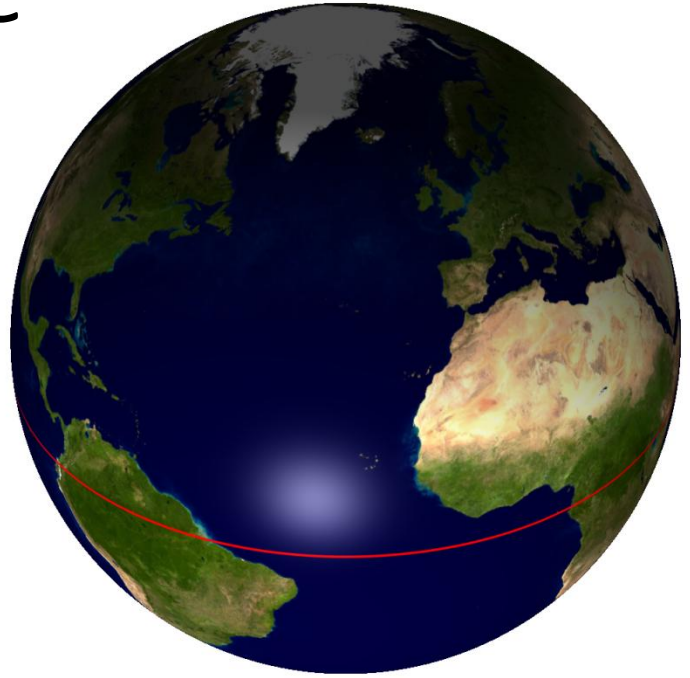


- Globalization
- One framework for all languages
- Storage and transmission of text
- A large database

Unicode Crash Course

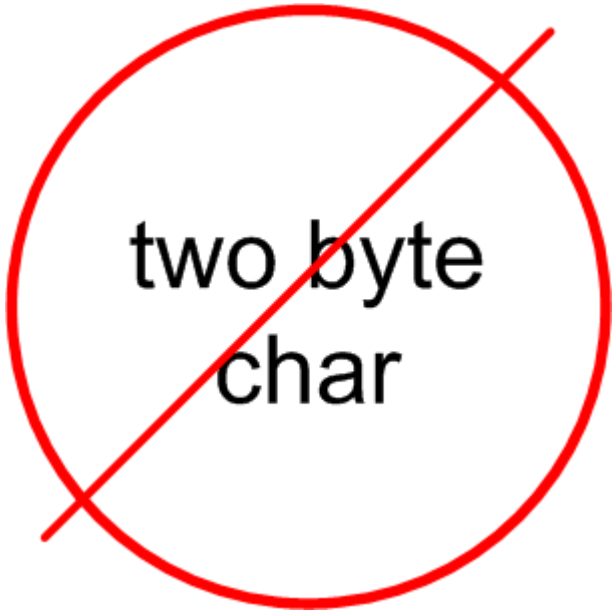
The Unicode Attack Surface

- All software
- All users



Unicode Crash Course

Unthink it



Unicode Crash Course

- 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**

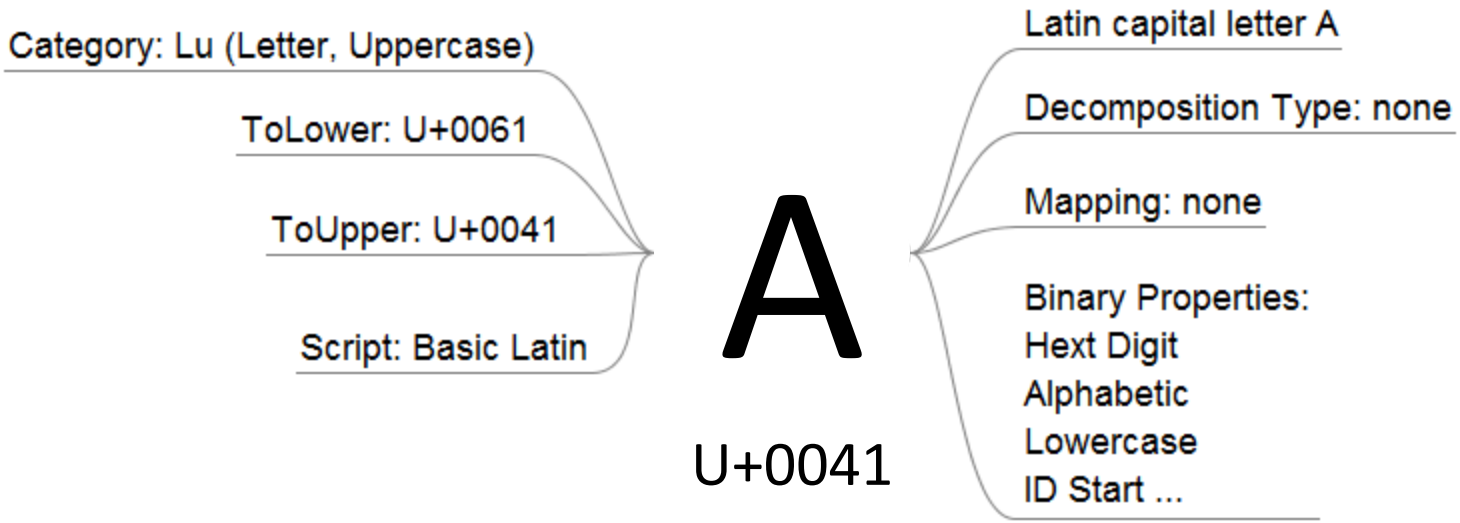
Unicode Crash Course

Code Points

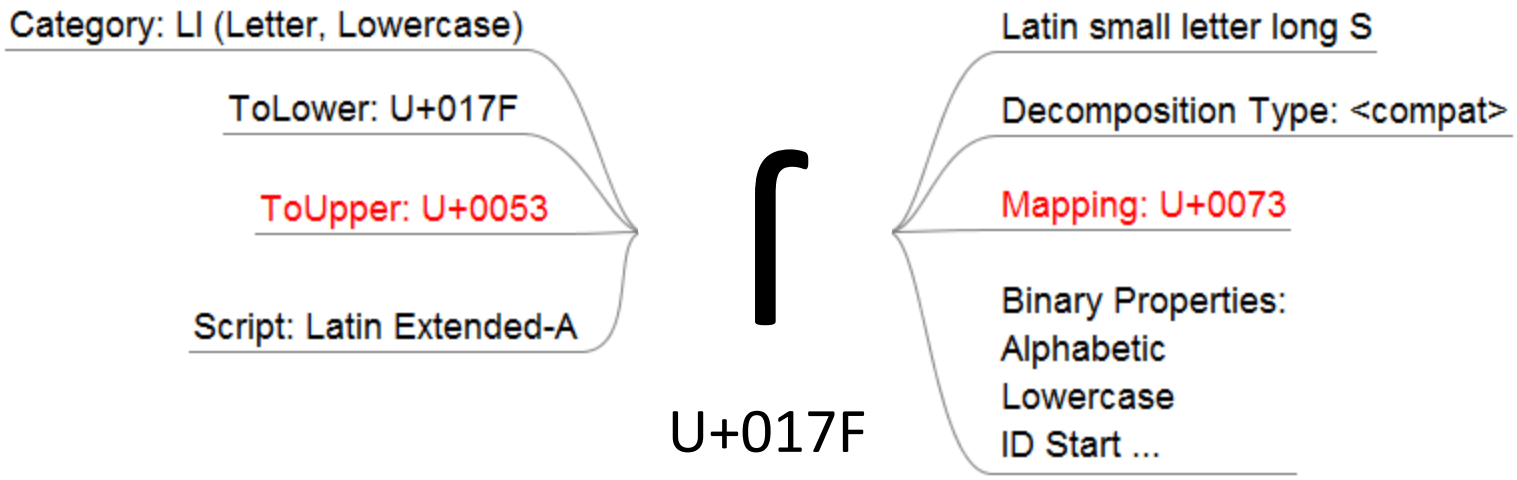
A = U+0041

Every character has a unique number

Unicode Crash Course



Unicode Crash Course



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

Ａ

Ａ

\xEF\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

Visual Spoofing and IDN's

Root Causes

Visual Spoofing

- Over 100,000 assigned characters
- **Many lookalikes** within and across scripts

Open your Web browser

And follow along

http://nottrusted.com/idn.php

Attack Vectors

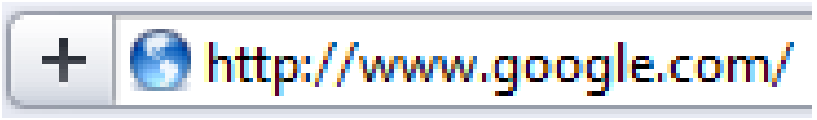
IDN homograph attacks

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

g g

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.

Root Causes

IDN – so what's the problem?

- Divergent user-agent implementations
- Lookalikes everywhere
- IDNA and Nameprep based on Unicode 3.2
  - We're up to Unicode 5.1 (larger repertoire)

Root Causes

The state of International Domain Names

- Registrars still allow
  - Confusables
  - Combining marks
  - Single, Whole and Mixed-script
- Registrars can't control
  - Syntax spoofing in sub domain labels

Attack Vectors

Visual spoofing Vectors

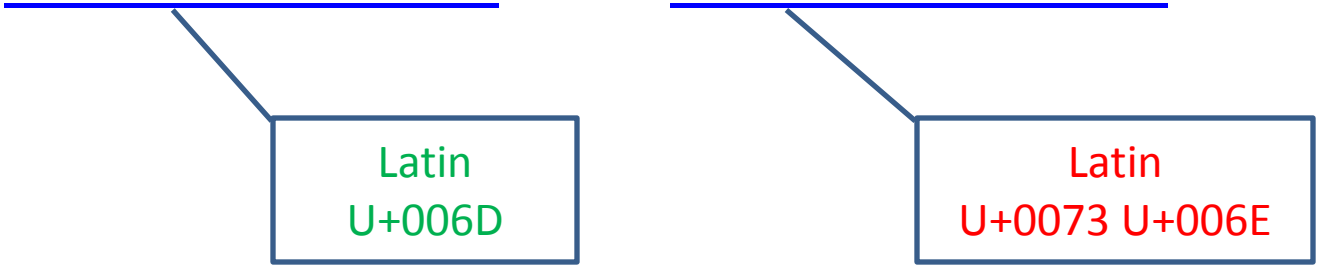
- Non-Unicode attacks
- Confusables
- Invisibles
- Problematic font-rendering
- Manipulating Combining Marks
- Bidi and syntax spoofing

Attack Vectors

Non-Unicode homograph attacks

rn can look like m in certain fonts

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



Attack Vectors

Non-Unicode homograph attacks

Are you using mono-width fonts?

```
0 and O
1 and l
5 and S
```

Attack Vectors

Non-Unicode homograph attacks

Classic long URL's

```
http://login.facebook.intvitation.videomessageid-
   h048892r39.sessionnfbid.com/home.htm?/disbursements/
```

Attack Vectors

*Unicode and **The Confusables***

www.apple.com

// All Latin using Latin small letter Alpha 'ɑ'

www.facebook.com

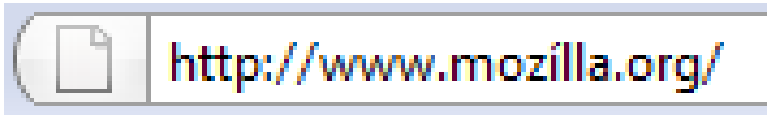
// Mixed Latin/Greek with lunate sigma symbol '**c**'

www.aЬc.com

// All Cyrillic 'abc'

Attack Vectors

IDN homograph attacks



Browsers whitelist .ORG

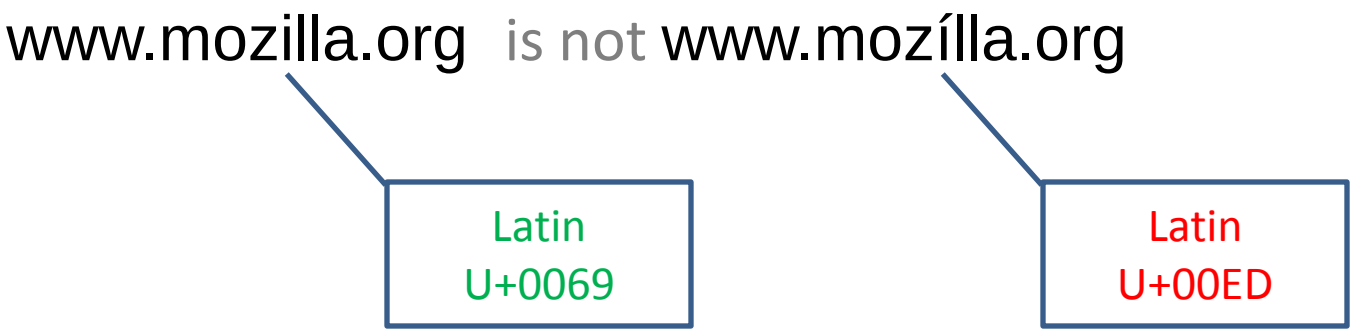
Attack Vectors

IDN homograph attacks

Others don’t necessarily but…

Attack Vectors

IDN homograph attacks



Attack Vectors

IDN Syntax Spoofing with / lookalikes

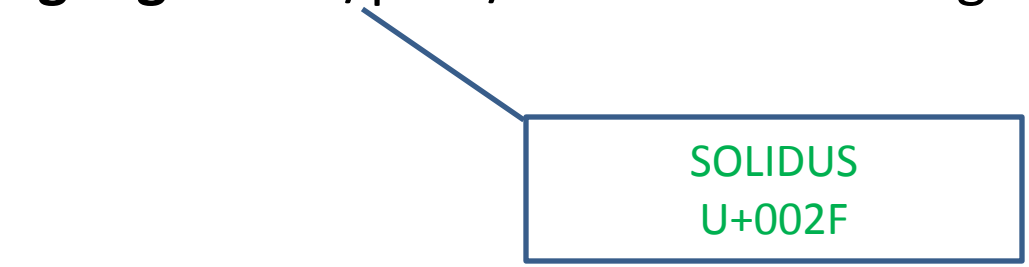
http://www.google.com／path／file?.nottrusted.org

(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

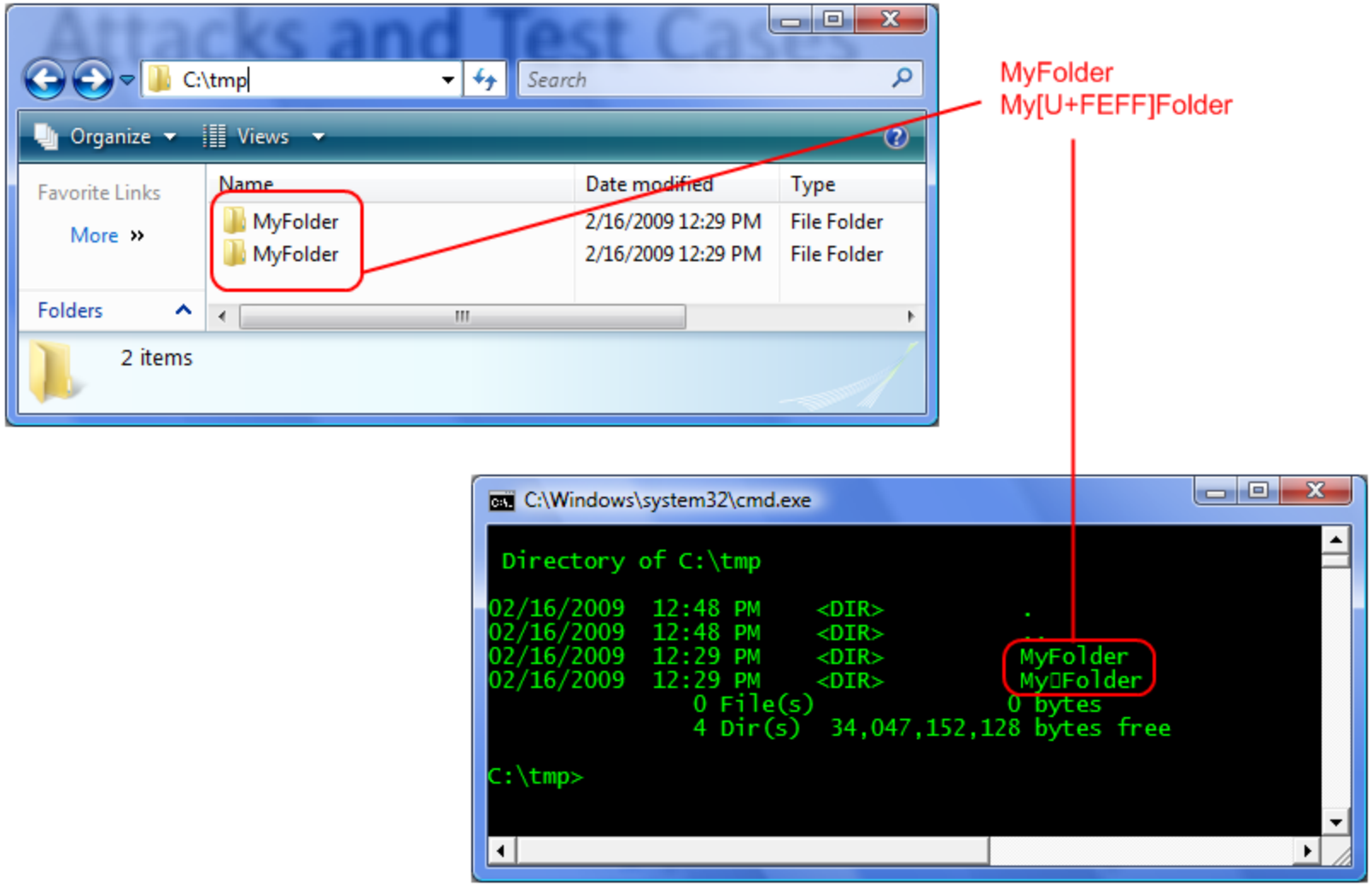
IDN Syntax Spoofing with / lookalikes

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

(However punctuation not required…)

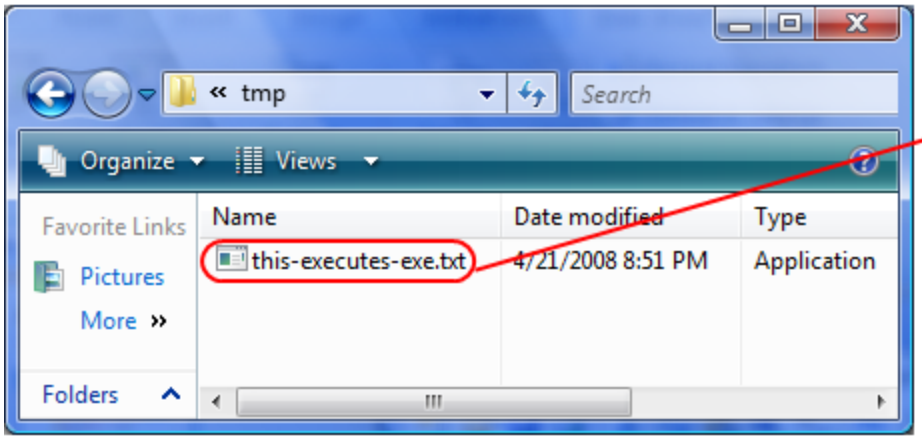
Attack Vectors

The Invisibles



Attack Vectors

Visual Spoofing with Bidi Explicit Directional Overrides



this-executes-[U+202E]txt.exe

Best-fit Mappings

Root Causes

Best-fit mappings

Commonly occur in charset transformations and even innocuous API's

Impact: Filter evasion, Enable code execution

When σ becomes s

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!

Normalization

Root Causes

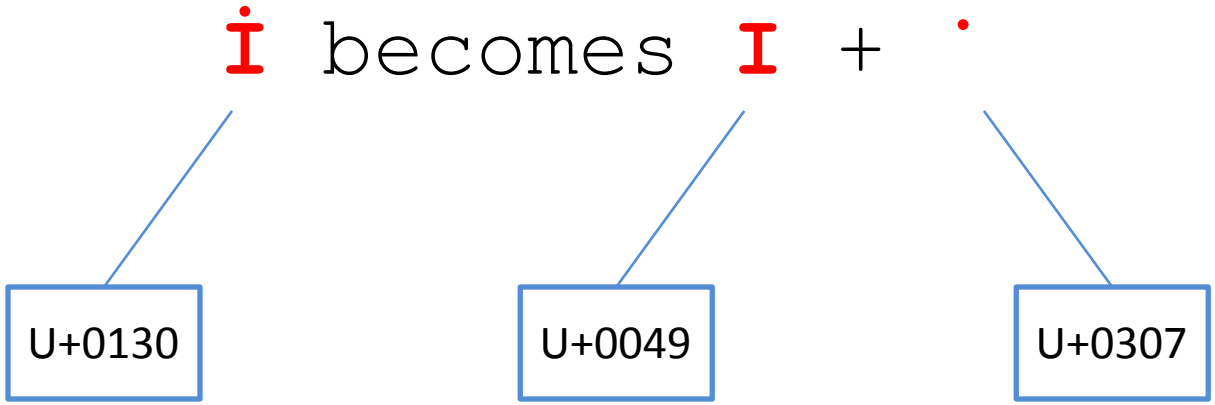
Normalization

Normalizing strings **after** validation is dangerous

Impact: Filter evasion, Enable code execution

Root Causes

Normalization



Root Causes

Normalization

But are there dangerous characters?

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

Root Causes

Normalization

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

Overlong UTF-8 (Canonicalization)

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

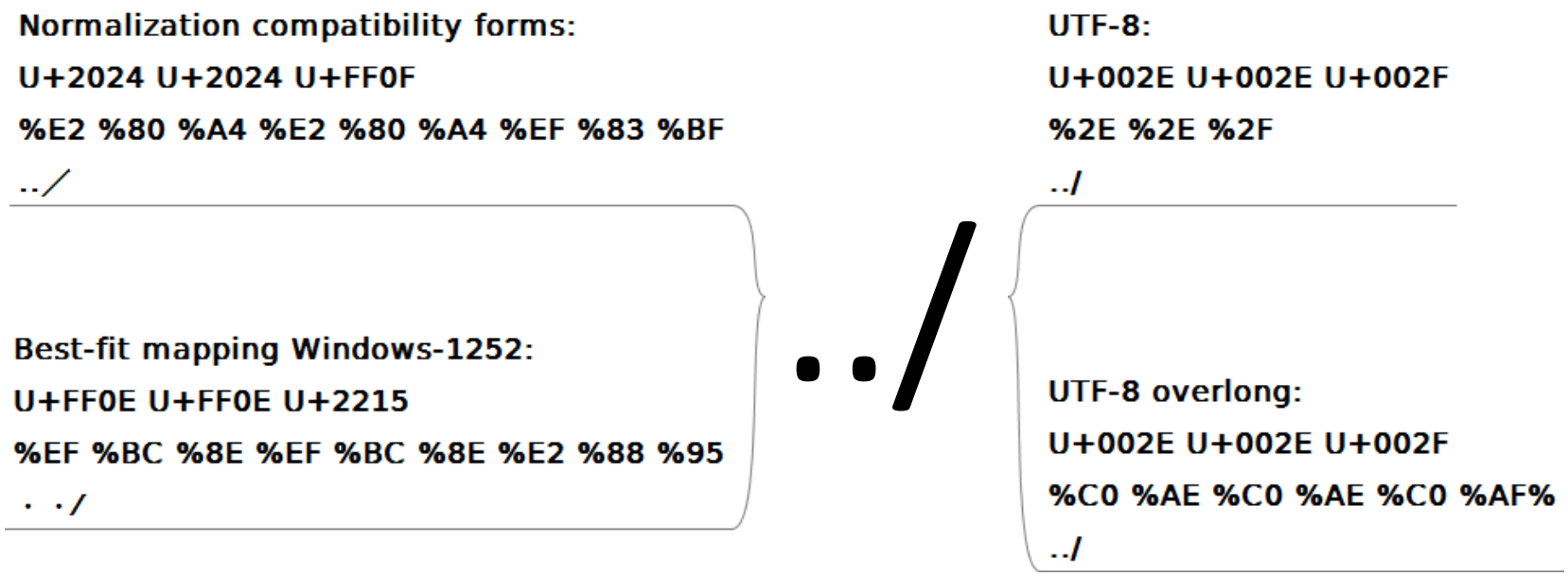
- Unicode **specification forbids**
  - **Generation** of non-shortest form
  - **Interpretation** of non-shortest form for BMP
- Validate UTF-8 encoding (throw on error)

Attack Vectors

Directory traversal

How many ways can you say ../

Attack Vectors



Handling the Unexpected
(Good fuzzing test cases)

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

Over-Consumption

Root Causes

Handling the Unexpected: Over-consumption

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

** Big problem with MBCS lead bytes*

```
<41 C2 3E 41>   becomes
<41 41>
```

Root Causes

Handling the Unexpected: Over-consumption

```
<img src="#0xC2"> "onerror="alert(1)"<br />
```

Browser sees:

```
<img src="#>" onerror="alert(1)" /><br />
```

Character Substitution

Root Causes

Handling the Unexpected: Character-substitution

Correcting insecurely rather than failing

- Substituting a '.' or a '/' would be bad

Character Deletion

Root Causes

Handling the Unexpected: Character-deletion

"deletion of noncharacters"

Root Causes

Handling the Unexpected: Character-deletion

`<scr[U+FEFF]ipt>` becomes `<script>`

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

Case Study: Apple and Mozilla

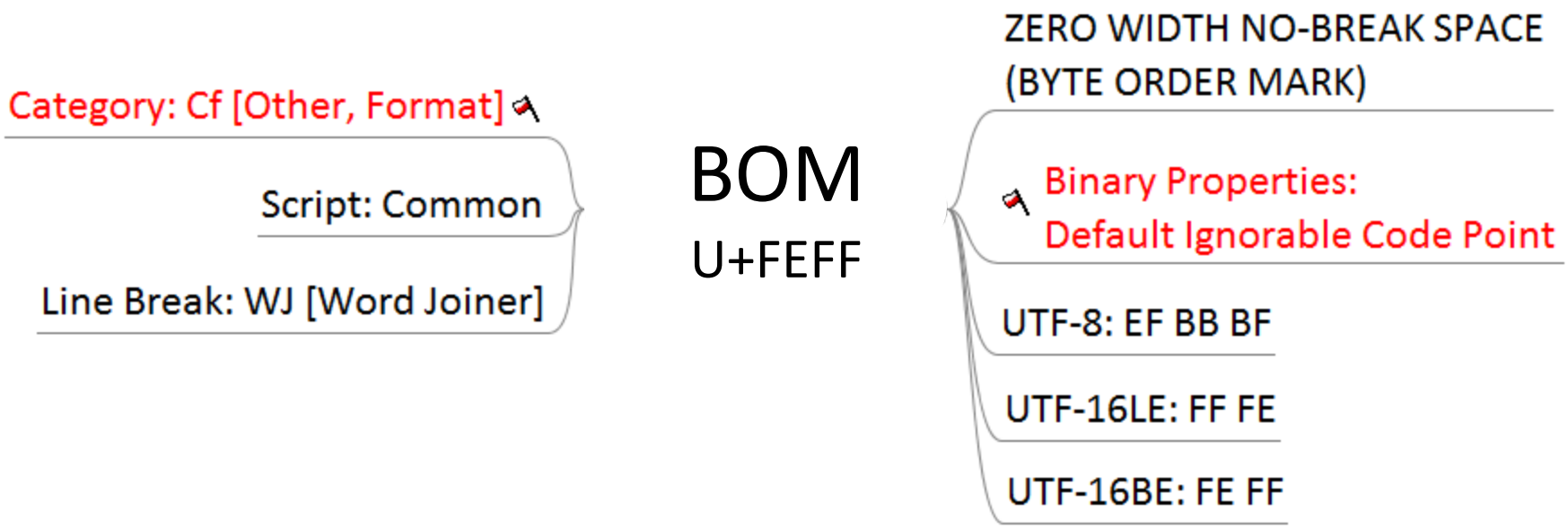
Safari and Firefox BOM consumption

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

Case Study: Apple and Mozilla

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

A Closer Look: The BOM



Casing

Root Causes

Casing

- Upper and lower-casing can produce dangerous text
- Casing can multiply the buffer sizes needed
- **Impact**: Filter evasion, Enable code execution

Root Causes

Casing

toLower(“**İ**”) == “**i**”

toLower(“scr**İ**pt”) == “scr**i**pt”

Root Causes

Casing

```
len(x) != len(toLower(x))
```

Root Causes

Guidance for Casing

- Perform casing operations **before** validation
- Leverage existing frameworks and API's
  - ICU, .Net

Buffer Overflows

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

Operation	UTF	Factor	Sample	
Lower	8	**1.5**	Ⱥ	U+023A
	16, 32	**1**	A	U+0041
Upper	8, 16, 32	**3**	ΐ	U+0390

Source: *Unicode Technical Report #36*

Root Causes

Buffer Overflows

Normalization- maximum expansion factors

Operation	UTF	Factor	Sample	
NFC	8	3X	□	U+1D160
	16, 32	3X	ﷺ	U+FB2C
NFD	8	3X	ΐ	U+0390
	16, 32	4X	ᾂ	U+1F82
NFKC/NFKD	8	11X	صلى الله عليه وسلم	U+FDFA
	16, 32	18X		

Source: *Unicode Technical Report #36*

Controlling Syntax

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=#[U+180E]onclick=alert()>
```

Case Study: Opera

Specifications

Root Causes

Specifications

1) Character stability
   - IDNA/Nameprep based on Unicode 3.2
2) Designs
   - Specs are carefully designed but not always perfect
     - This was 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.

Charset Transformations

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

Charset Mismatches

Root Causes

Charset Mismatches

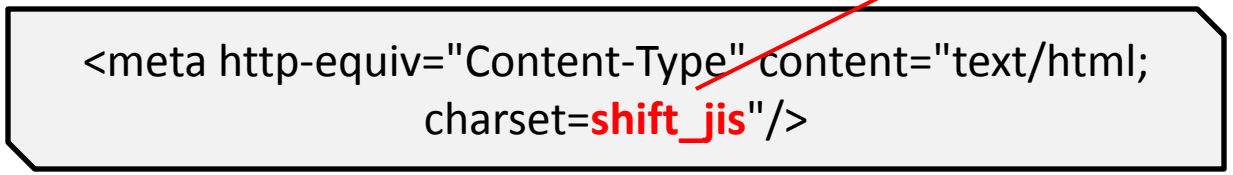
- Some charset identifiers are ill-defined
- Vendor implementations vary
- User-agents may sniff if confused
- Attackers manipulate behavior
- **Impact**: Filter evasion, Enable code execution

Root Causes

Charset Mismatches

Content-Type: charset=**ISO-8859-1**

Attacker-controlled input



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

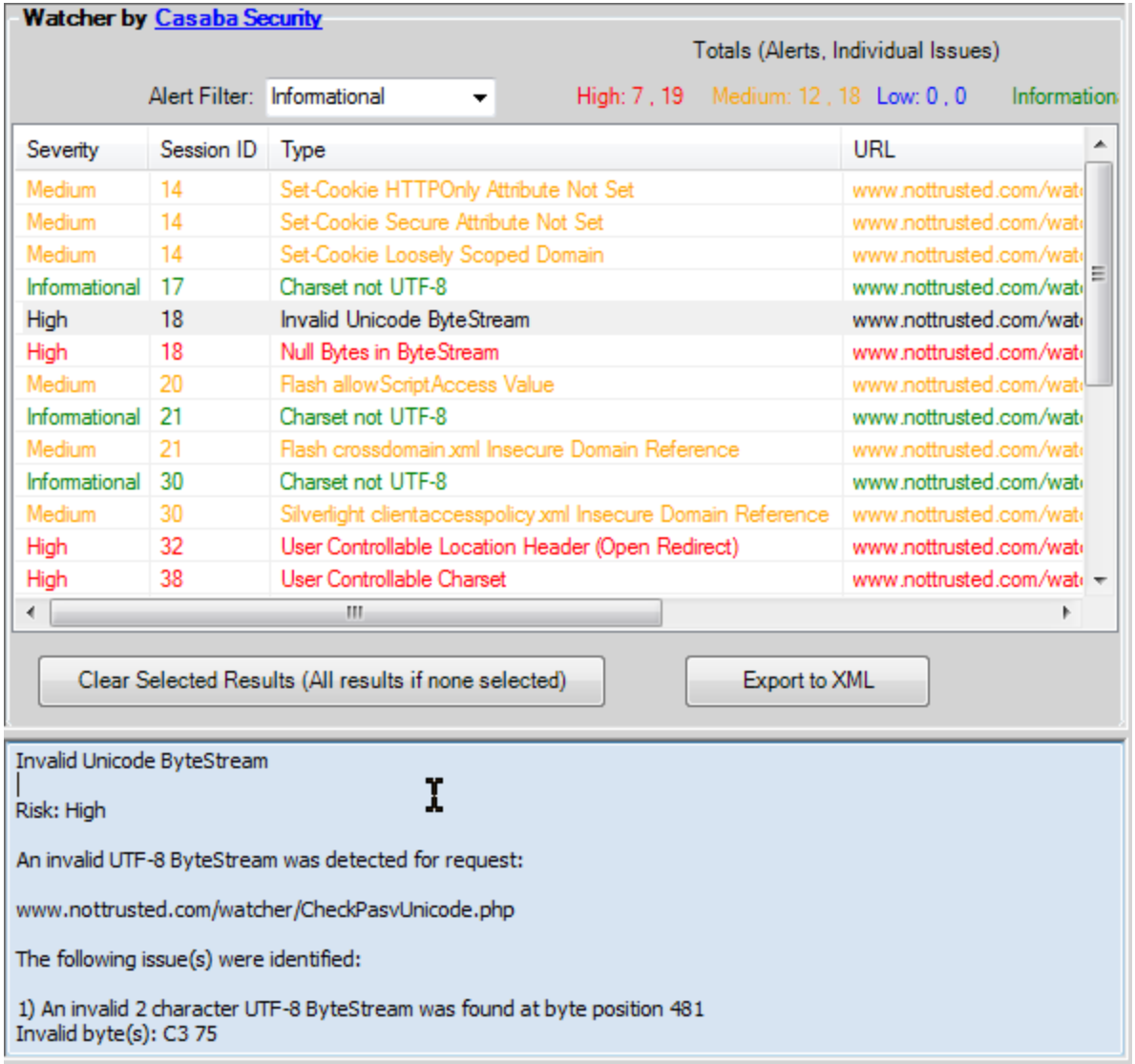
- Watcher
  - Microsoft SDL recommended tool
  - Passive Web-app security testing and auditing
- Unibomber
  - XSS autopwn testing tool

Tools

Watcher – Some of the Passive Checks Included

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

Tools



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

Thank you!

Casaba Security

www.casabasecurity.com

Chris Weber

Blog: www.lookout.net

Email: chris@casabasecurity.com

LinkedIn: http://www.linkedin.com/in/chrisweber