Host/Split

Exploitable Antipatterns in Unicode Normalization

Jonathan Birch, Microsoft Office Security
jobirch@Microsoft.com

Microsoft
This talk is about new Unicode vulnerabilities

• Not homographs.

• A lot of software needs to get fixed. Maybe your software.
Agenda

• Background: How Unicode Hostnames (IDN’s) work
• The HostSplit attack – making URL’s that switch domains
• Example exploit – stealing OAUTH tokens from O365
• What was vulnerable / what’s still vulnerable
• HostBond - a variant attack
• How to test / Best practices
Why attack Unicode URL’s?

• Story time: Pen-Test Lunch at Microsoft

• In 2017 Egyptian hieroglyph URL’s became available.
I got this one

.net

• Yes, that’s a man simultaneously riding two giraffes.
• But how does that work?
Background: How do IDN’s work?
It’s all ASCII Underneath

0x.net  ToASCII  xn--fq7d.net

ToUnicode

U-Label  A-Label
Unicode ➔ ASCII – A Two Step Process

1. Normalization
   Convert characters to a “standardized form”.

2. Punycoding
   Turn Unicode into ASCII.
Unicode ➔ ASCII: Normalization

Å (U+00C5)

Å (U+212B)

Å (U+0041, U+030A)

Normalize To

å (U+00E5)
Unicode ➔ ASCII: Puny coding

FISKMÅS

xn--fiskms-mua

ACE
(Means “this is Puny code”)

ASCII Stuff

State-machine instructions
ASCII $\rightarrow$ Unicode is Simple

• Just run the Punycode state machine and rehydrate the Unicode.

• RFC 3490 (IDNA) says the resulting U-label should have “ToASCII” run against it and we should fail if the result doesn’t match our input…
Three different specs for how this works

- IDNA2003: The original
- IDNA2008: The second try
- IDNA2008 + UTS46: A compatibility patch
The HostSplit Vulnerability
Revisiting Unicode Normalization

• Remember that “normalization” step?

• “Å” became “â”

• What if there were Unicode characters that normalized to ASCII characters with syntax-significance?
Unicode Normalizing to Control Characters

\( \text{a/c} \) Normalizes To \( \text{a/c} \)

(U+2100) “Account Of”

ASCII: a, forward slash, c
Splitting Hostnames

https://evil.c%Example.com

What happens if we perform ToASCII against this?
Splitting Hostnames

https://evil.ca%c.Example.com

Normalizes To

https://evil.ca/c.Example.com

No need to Punycode anything – it’s all ASCII now!
Does this really work?

• Yes, though not as broadly as it did when I started.

• The first vulnerability I found like this was in Edge and IE.

• Similar issues existed in .Net, Python, and Java.
The Edge / IE Vulnerability

When Edge received a redirect with this location header:

https://evil.c\%/c.Example.com

It redirected to:

https://evil.ca/c.Example.com

But why does this matter? How do we exploit it?
Attacking OAuth with HostSplit
Attacking OAuth

OAUTH Authorization Code Flow (RFC 6749 4.1)

- This is only secure because the authorization server has an allow list pattern for the redirect URI’s it accepts for any given application ID.
- But how does this allow list pattern work?
OAuth Redirect Allow Lists

- In OAuth 1.0 patterns like “*.office.com” were common.

- A URI like “http://evil.ca%e.office.com” would work.

- Nobody normalizes URL’s before comparing - the check just says, “Does the string end with .office.com?”
HostSplit vs. OAuth 1.0

Attacker web site

OFFICE_APP_ID, https://evil.ca/c.office.com

Authorization Server

Hostname ends with 
“.office.com”
Looks good!

Redirect is https://evil.ca/c.office.com
Edge goes to https://evil.ca/c.office.com

Authorization Code

Evil.ca
What about modern OAuth?

• Recent OAuth implementations have more restrictive allow list patterns.

• A specific hostname is usually required (no wildcards).

• The attack I’ve described so far only let’s us fool a subdomain check.
Redirects to the rescue?

• In 2014, a researcher named Wang Jing publicly disclosed a vulnerability he called “Covert Redirect”.

• Essentially: “If you can get an OAUTH token sent to an open redirect, you can sometimes steal it.”

• This met with a mixed reception.

• But HostSplit makes it much easier to find open redirects.
Attacking Office OAuth

- Office.live.com received Office OAUTH tokens and had a redirect that went to dropbox.com or any subdomain of it.
- This URL as an OAUTH target would let you steal tokens:
  ```
  ```
- Well, almost.
Saved by a bug

• Middleware in front office.live.com double-encoded UTF-8 response headers.

• The redirect actually went to: https://evil.cÃ¢â€œâ€¬.dropbox.com/wopi_edit/document1.docx

• This put us in an awkward position.
What’s vulnerable to HostSplit?
More than the example

- Not just OAUTH
- Not just Edge
- Not just .Raise

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>U+2048 ?!</td>
<td>U+FF1A :</td>
</tr>
<tr>
<td>U+FF0F /</td>
<td>U+2488 1.</td>
</tr>
<tr>
<td>U+FF03 #</td>
<td>U+FE47 →</td>
</tr>
<tr>
<td>U+FF20 @</td>
<td>And many others...</td>
</tr>
</tbody>
</table>
IDNA Version Matters

• IDNA2008 blocks HostSplit-enabling characters.
• IDNA2003 and IDNA2008 + UTS46 are vulnerable.
• The "UseSTD3ASCIIRules" flag fixes these by blocking the bad characters.
• Why doesn’t everyone use either IDNA2008 or this flag?
Blame Underscores

• UseSTD3ASCIIIRules only allows alphanumerics, dashes, and periods.
• Lots of real-world hostnames contain underscores (mostly old intranet stuff).
• Many implementations use IDNA2008 + UTS46 without STD3 rules so that they can still connect to hosts with underscores in them.
Browsers are safe now

- Edge and IE vulnerabilities were fixed as CVE-2019-0654 in February of 2019.
- They now refuse to follow HostSplit HTTP redirects.
- Firefox and Chrome were already safe.
- Safari is probably also safe, but it percent-encodes dangerous Unicode characters for some reason.
.NET was vulnerable

```csharp
string url = @"http://canada.ca/c%products.office.com/test.exe";
UriBuilder uriBuilder = new UriBuilder(url);
IdnMapping idnMapping = new IdnMapping();
uriBuilder.Host = idnMapping.ASCII(uriBuilder.Host);
url = uriBuilder.ConnectionString();
System.Console.WriteLine(url);
```

This used to output “http://canada.ca/c.products.office.com/test.exe”
.NET is fixed

- patched as [CVE-2019-0657](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0657) in February 2019
- The logic used now is:
  1. Identify that the URI contains a host name that will be resolved via DNS.
  2. Isolate the host name, removing port + authentication.
  3. Encode only the host using `IdnToAscii`
  4. Check the output to ensure no URI control characters have been added to the host.

(M. Kerr, 2018)

UriBuilder now throws a `System.UriFormatException` if you give it a URL like the one on the previous slide.
Fiddler was vulnerable (because of .NET)

• Telerik’s Fiddler tool runs on .NET, and if you had it intercepting traffic, it would “fix” all location headers so that the Edge/IE bug worked in every browser.

• This got better when Microsoft fixed .NET.
Python was vulnerable

```python
>>> from urllib.parse import urlsplit, urlunsplit
>>> url = 'http://canada.c%.microsoft.com/some.txt'
>>> parts = list(urlsplit(url))
>>> host = parts[1]
>>> host
'canada.c%.microsoft.com'
>>> newhost = []
>>> for h in host.split('. '):
...     newhost.append(h.encode('idna').decode('utf-8'))
...
>>> parts[1] = '.'.join(newhost)
>>> finalUrl = urlunsplit(parts)
>>> finalUrl
'http://canada.ca/c.microsoft.com/some.txt'
```

• Credit for this vulnerability is shared with Panayiotis Panayiotou
Python had an extra variant

```python
>>> from urllib.parse import urlparse
>>> r='http://bing.com'+u'http://bing.com'+u'http://bing.com':password@products.office.com'
>>> o = urlparse(r)
>>> o.hostname
'products.office.com'
>>> a = r.encode("IDNA").decode("ASCII")
>>> a
'http://bing.com#':password@products.office.com'
>>> o = urlparse(a)
>>> o.hostname
'bing.com'
```
Python is fixed

• Original issue was patched as CVE-2019-9636
• Variant using basic auth patched as CVE-2019-10160
Java was vulnerable

```java
import java.net.*;

public class IDNTest {
    public static void main(String[] args) throws Exception {
        String idnTest = "evil.C\u2100B.microsoft.com";
        String result = IDN.toASCII(idnTest);
        System.out.println(result + "\n");

        URL myUrl = new URL("http://evil.C\u2100B.microsoft.com");
        System.out.println(myUrl.getHost() + "\n");
    }
}
```

This output [http://evil.CA/B.Microsoft.com](http://evil.CA/B.Microsoft.com)
Java is fixed

• Patched as CVE-2019-2816 / S8221518 in July 2019
Your Windows code might still be vulnerable

• The Windows API `IdnToASCII` converting “a/c” to a/c.
• This is necessary – `IdnToASCII` isn’t only used for hostnames, so keeping it compliant with the standard is critical.
• `IDN_USE_STD3_ASCII_RULES` flag makes `IdnToASCII` safe. This also forbids underscores though.
Your Linux code might still be vulnerable

• LibIDN and LibIDN2 also convert “_kwargs” to a/c.
• The usestd3asciirules flag makes LibIDN safe.
• The no-tr46 flag makes LibIDN2 safe.
• I contacted bug-libidn@gnu.org about this in February of 2019, and they said it was by design.
HostBond: a variant attack
Characters only allowed in IDNA2008

• “ZERO-WIDTH JOINER” (U+200D) and “ZERO-WIDTH NON-JOINER” (U+200C) are deleted during normalization in IDNA2003.

• They’re invisible and make visual spoofing too easy.

• But these characters are important for some languages as a way of changing ligatures. (क्ष vs क्ष)

• IDNA2008 allows these, but only if the characters on either side of them would be visually changed.
ToASCII vs ToUnicode

- In IDNA2008, a string like “micro” + zero-width-joiner + ”soft” has the zero-width-joiner (ZWJ) thrown away during normalization.
- Many implementations of ToUnicode don’t make sure that the result has appropriate neighbors for decoded ZWJ’s.
- So the same string with the ZWJ already encoded is just fine:

  xn--microsoft-469d.com
Vanishing ZWJ’s

There are two problems with a URL like \texttt{xn--microsoft-469d.com}:

1. The zero-width joiner is invisible, so the U-Label for this is \texttt{microsoft.com} (there’s a ZWJ there – you just can’t see it).

2. Since the zero-width joiner isn’t technically allowed, it won’t survive being converted to Unicode and then back to ASCII.
The Hostbond Vulnerability

• Someone has a mail server at email.somecloudhost.net, and we want to impersonate them.

• We register our own domain with a Punycoded zero-width joiner in between the “e” and the “m”:

  xn--email-xt3b.somecloudhost.net

• What happens if we send email from our server?
HostBond Exploit - Gmail

• If we send email to Gmail from “admin@xn--email-xt3b.somecloudhost.net”, they decode it.

• The email shows up as having come from admin@email.somecloudhost.net

• SPF and DKIM are checked against the xn--email-xt3b version.

• If you reply, Gmail throws away the ZWJ, so it goes to the real admin@email.somecloudhost.net
HostBond Exploit - Gmail

• I reported this vulnerability to Google in February of 2019.

• They have acknowledged the report but have yet to make any fixes.

• They are actively working on the issue and monitoring for exploits.
LibIDN2 was vulnerable to HostBond
echo “xn--microsoft-469d.com” | idn2 -d microsoft.com

Zero-Width Joiner

- Fixed as [CVE-2019-12290](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12290) in version 2.2.0 of LibIDN2
- Credit shared with Tim Ruehsen (GNU libidn), Florian Weimer (GNU glibc) and Nikos Mavrogiannopoulos (GnuTLS)
HostBond - Limitations

• Only works against IDNA2008.
• Only exploitable when you can provide a Punycoded URL that will get decoded – usually requires UI.
• Domain registrars block creation of hostnames like these.
• But providers of third-level domains generally allow these hostnames.
Testing for HostSplit and HostBond
How to test for HostSplit

• Insert a magic character into a URL field and see what it resolves to.

• It’s easy if you can monitor network traffic. Look at DNS.

• http://canada.c%e.bing.com works.
A better HostSplit test case

Some domain where you log requests.

http://a.com/X.b.com

(Fullwidth Solidus [U+FF0F])

Map all b.com subdomains to the same server, log requests.

Requests that go to “a.com” are vulnerabilities. So are DNS lookups for a.com
How to test for HostBond

• If you have code that renders a user-provided hostname and might try to de-Punycode it, make sure it only decodes valid IDN’s.

• There are three test cases worth trying here:
  1. Zero-width joiner: “xn--TC-m1t.com” should not become “TC.com”.
  2. Bubble numbers: “x--orh.com” should not become “①.com” or “1.com”.
  3. Greek question mark: “xn--ab-y4b.com” should not decode to “a;b.com”.

HostBond isn’t just a web vulnerability

• Email (SMTP), IM (SIP/SIMPLE/MSRP), and other Internet protocols that use hostnames are also potentially vulnerable.

• For email: try sending mail with an extra recipient that has bad Punycode in the hostname, like test@xn--bing-676a.com – this should not render as test@bing.com
Best Practices for Unicode URL’s
Make all hostname decisions using ASCII

- Only A-labels should be used for security decisions.

- Software that tries to compare or filter hostnames should run some version of ToASCII on them first.

- Lots of platform code does this wrong.
UseSTD3ASCIIRules

• Use your platform’s version of “UseSTD3ASCIIRules”.

• This flag ensures that Unicode normalization does not introduce syntax-significant characters into a URL.

• This flag blocks hostnames with underscores, so don’t use underscores in hostnames.
Wrap bad platform code

• Lots of API’s are vulnerable to HostSplit.
• To fix these, extract the hostname first then compare after calling the API to make sure no new syntax characters appeared.
• Many API’s will decode invalid Punycode. Wrap these in a function that adds the round-trip check.
Go make this better, please.

• Test software for Unicode normalization vulnerabilities.

• Require URL’s be compared as ASCII.

• Hack stuff (ethically!) – that makes things better too.
Questions?

jobirch@Microsoft.com
CVE’s and Credits

- **CVE-2019-0654** Microsoft Browser Spoofing Vulnerability
- **CVE-2019-0657** .NET Framework and Visual Studio Spoofing Vulnerability
- **CVE-2019-9636** Python, `urlsplit` does not handle NFKC normalization
  - Credit shared with Panayiotis Panayiotou
- **CVE-2019-10160** Python, `urlsplit` NFKD normalization vulnerability in `user:password@`
- **CVE-2019-2816** Oracle Java SE/Java SE Embedded, “Normalize normalization”
- **CVE-2019-12290** LibIDN2, “Perform A-Label roundtrip for lookup functions by default”
  - Credit shared with Tim Ruehsen (GNU libidn), Florian Weimer (GNU glibc) and Nikos Mavrogiannopoulos (GnuTLS)
- Special thanks to Tina Zhang-Powell of MSVR, for helping to coordinate these fixes.
References


• Kerr, Max (Personal Communication August 27, 2018)

• Goldmann, Mikael, "Creative usernames and Spotify account hijacking", June 2013, <https://labs.spotify.com/2013/06/18/creative-usernames/>
Table of potential URL-splitting characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Unicode Code Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+2100, ™</td>
<td></td>
</tr>
<tr>
<td>U+2101, ™️</td>
<td></td>
</tr>
<tr>
<td>U+2105, ℅</td>
<td></td>
</tr>
<tr>
<td>U+2106, ℆</td>
<td></td>
</tr>
<tr>
<td>U+FF0F, /</td>
<td></td>
</tr>
<tr>
<td>U+2047, ??</td>
<td></td>
</tr>
<tr>
<td>U+2048, ?!</td>
<td></td>
</tr>
<tr>
<td>U+2049, ؤ</td>
<td></td>
</tr>
<tr>
<td>U+FE16, ?</td>
<td></td>
</tr>
<tr>
<td>U+FE56, ?</td>
<td></td>
</tr>
<tr>
<td>U+FF1F, ?</td>
<td></td>
</tr>
<tr>
<td>U+FE5F, #</td>
<td></td>
</tr>
<tr>
<td>U+FF03, #</td>
<td></td>
</tr>
<tr>
<td>U+FE6B, @</td>
<td></td>
</tr>
<tr>
<td>U+FF20, @</td>
<td></td>
</tr>
</tbody>
</table>