

# 5 Unicode Vulnerabilities That Could Byte You

(U+0079 U+0365)



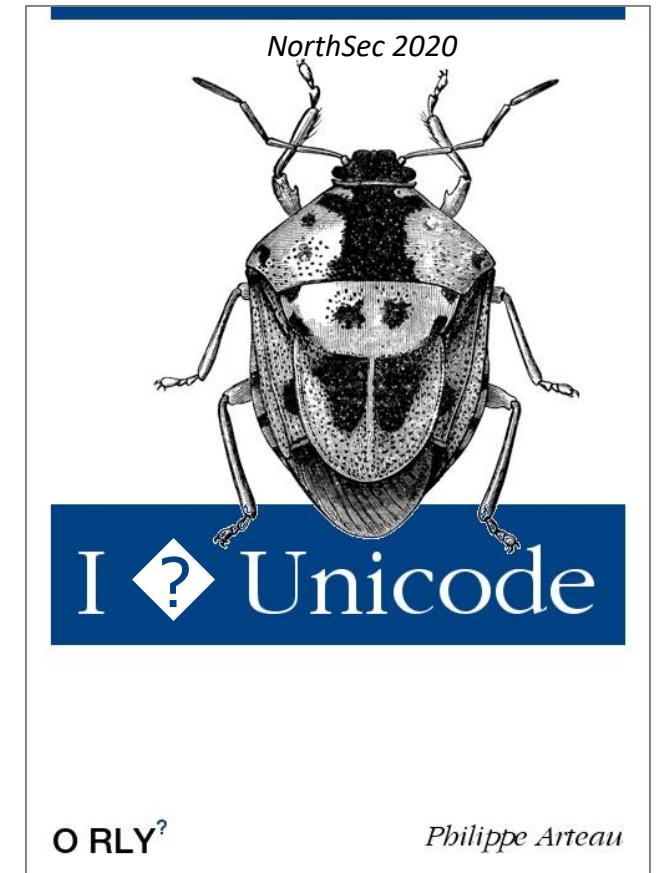
GoSECURE

nsec

Presented by Philippe Arteau from GoSecure

# Agenda

- Small history of encoding
- Normalization
- Case modification
- Bypassing filters
- Homograph attacks
- Data integrity



# Bio

- Philippe Arteau
- Security Researcher at  GoSECURE
- Open-source developer
  - Find Security Bugs (SpotBugs - Static Analysis for Java)
  - Security Code Scan (Roslyn – Static Analysis for .NET)
  - Burp and ZAP Plugins (Retire.js, CSP Auditor)
- Volunteer for the  nsec conference and former trainer

# Small history of encoding

# ASCII

Created in the 1960s

- Control Characters (0-31)
  - Null (0), Bell (7), Backspace (8), EOF (26)
- Standard Character Set (32-127)
  - A...Z, a-z,
- Extended character Set (128-255)
  - Accented characters, symbols, box characters



# Code Pages

Attempt to standardize ascii implementation to support special characters

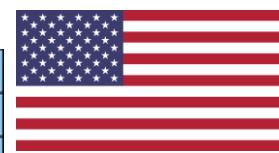
- IBM-PC / DOS Latin US : Code page 437
  - Greek : Code page 737
  - Cyrillic : Code page 855
  - ANSI / Windows-1252 : Code page 1252
  - ...

	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	^	_
60	t	a	b	c	d	e	f	g	h	i	j	k	l	m	n
70	p	q	r	s	t	u	v	w	x	y	z	{		3	~
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BAA	BBA	BBC	BBD	BBE	BFF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
	=	≥	≤	≡	J	÷	≈	F <sup>o</sup>	•	√	n	f <sup>o</sup>	2	■	FF

# Unexpected Encoding



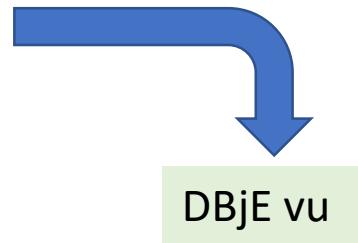
## Déjà vu



# CP-437

# DOS Latin US

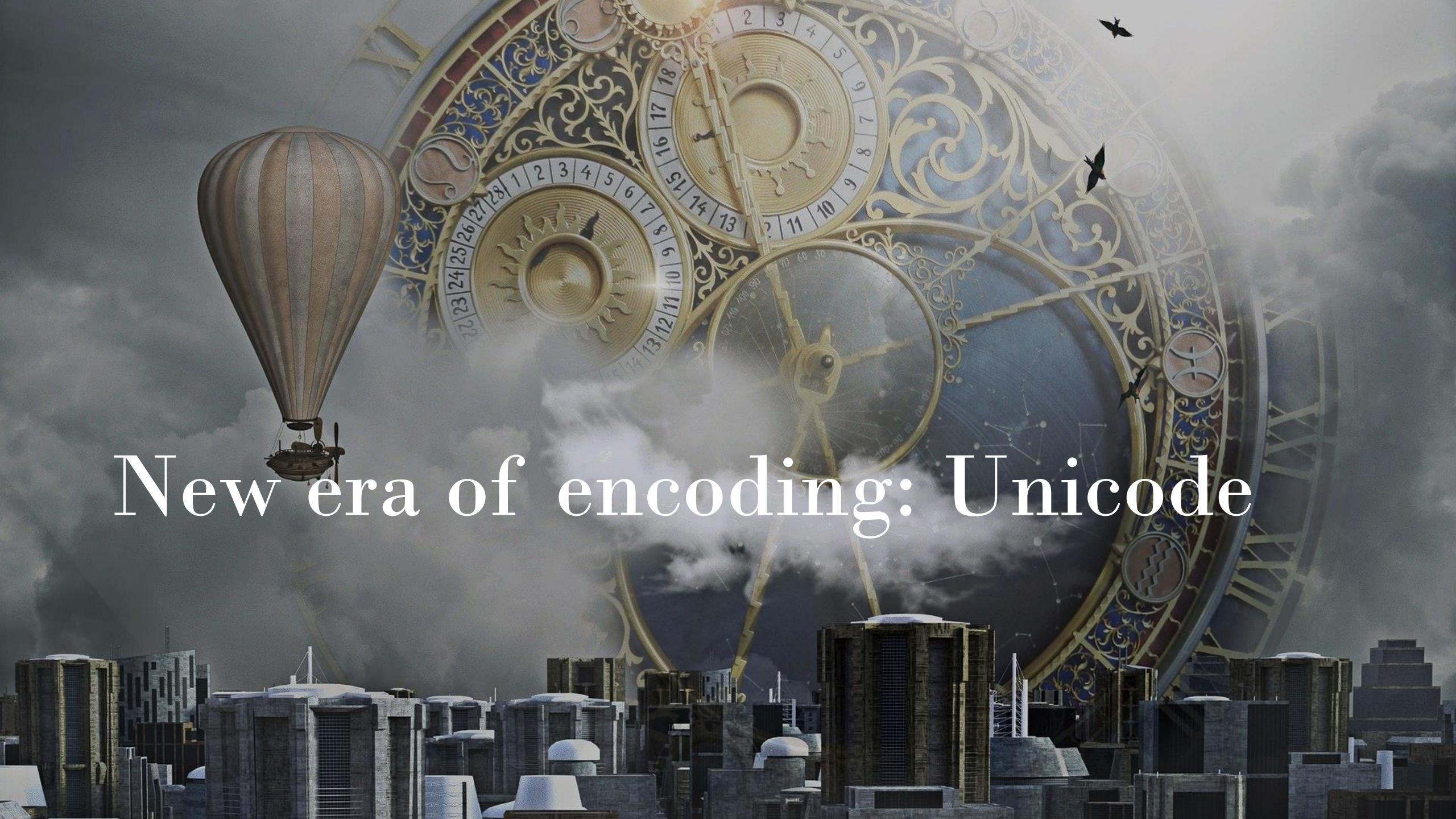
D\x82j\x85 vu



DBjE vu



**CP-866  
DOS Cyrillic  
Russian**



New era of encoding: Unicode

# Unicode Code Points

- Uniquely index “all” characters : Latin, Cyrillic, Asian, African script, Mathematical symbol, Measuring Unit, Egyptian Hieroglyphs and Emojis.

A    é    水     $\Sigma$     💩

U+0041    U+00e9    U+6C34    U+2211    U+ 1f4a9

- Code Points **do not** define how it should be **encoded** (bytes representation)

To get detailed CP info:

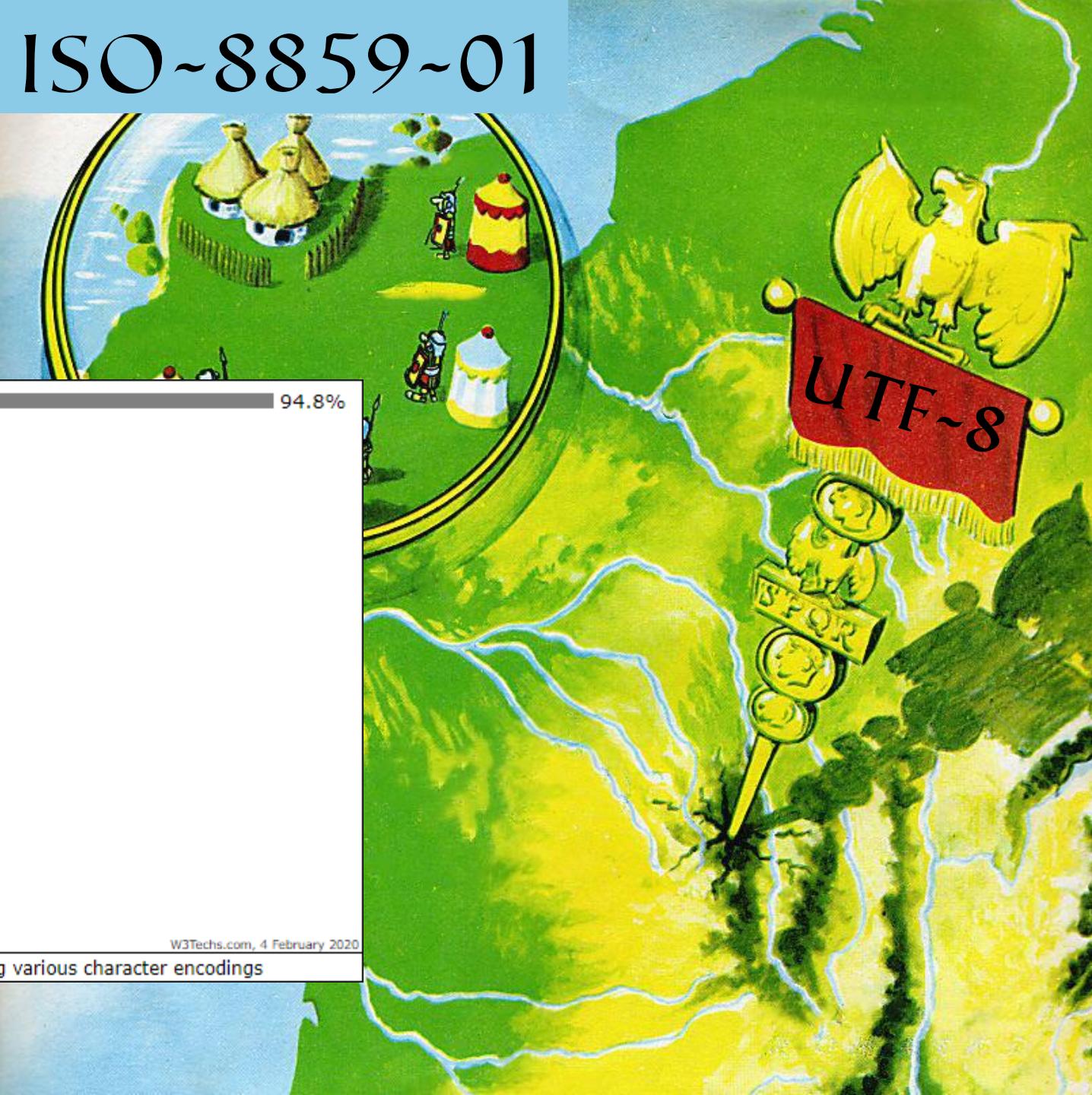
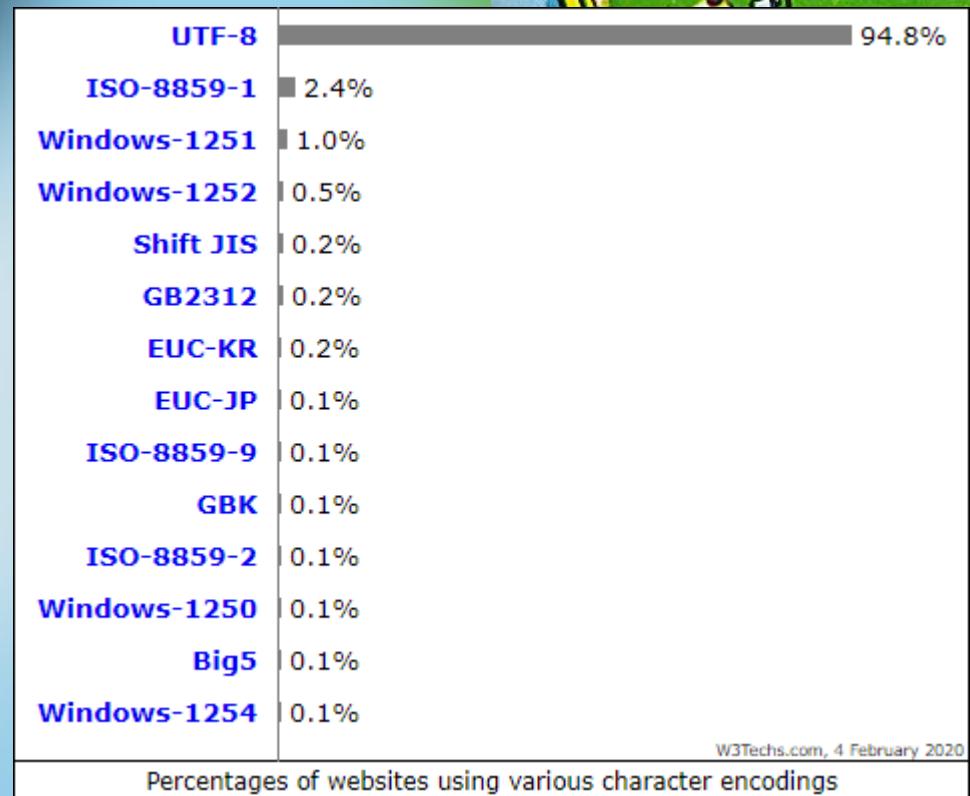
<https://www.fileformat.info/info/unicode/char/1f4a9/index.htm>

# UTF-8

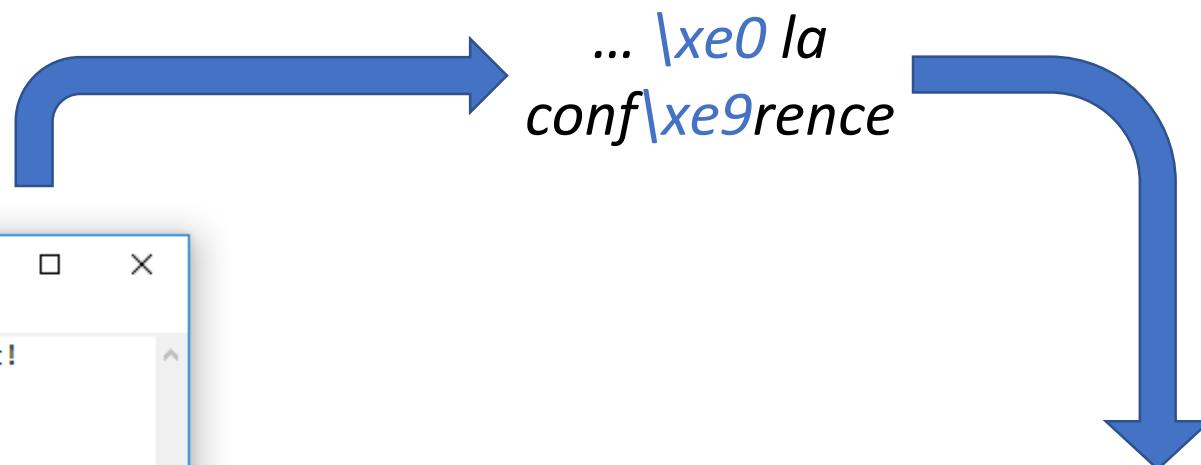
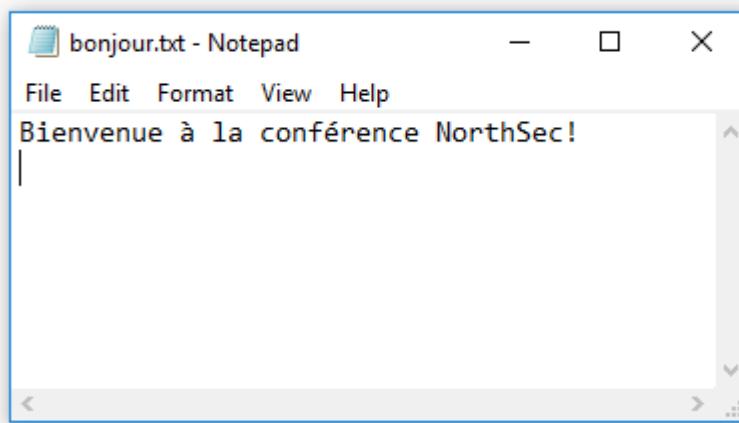
- Encoding format supporting Unicode Code Points
- The most popular encoding format
- First presented in 1993, Officialized in 2003 ([RFC 3629](#))
- Variable width length encoding ... **1 byte != 1 character**

0____					
110_ __	10_ __				
1110 __	10_ __	10_ __	10_ __		
[...]					
1111 110_	10_ __	10_ __	10_ __	10_ __	10_ __

# ISO~8859~01



# Unexpected Encoding



```
philippe@GS00RD01:~/encoding$ cat bonjour.txt
Bienvenue à la conférence NorthSec!
philippe@GS00RD01:~/encoding$ █
```

**ISO-8859-1**

**UTF-8**



Normalization issues

$\zeta$	$\leftrightarrow$	$C + \dot{Q}$
$q + \dot{o} + \dot{Q}$	$\leftrightarrow$	$q + \dot{Q} + \dot{o}$
가	$\leftrightarrow$	ㄱ + ㅏ
$\Omega$	$\leftrightarrow$	$\Omega$

# NFC/NFKC Normalisation

Objective :

- Compare two Unicode strings
- Use to build username, name, ids, description in URL or path.
- *Converting any Unicode string to Basic Latin (ASCII) only string.*



# NFC vs NFKC

## Canonical Equivalence (NFC)

Figure 1. Examples of Canonical Equivalence

Subtype	Examples
Combining sequence	Ç ← C+ç
Ordering of combining marks	q+ö+ø ← q+ø+ö
Hangul & conjoining jamo	가 ← ㄱ + ㅏ
Singleton equivalence	Ω ← Ω

## Compatibility Equivalence (NFKC)

Figure 2. Examples of Compatibility Equivalence

Subtype	Examples
Font variants	ſ → H Ĳ → H
Linebreaking differences	[NBSP] → [SPACE]
Superscripts/subscripts	i <sup>9</sup> → i9 i <sub>9</sub> → i9
Squared characters	アバート → アパート
Fractions	¼ → 1/4
Other	dž → dž

# Normalisation

When building URL or path, normalization can lead to security issues.

%	/
(U+8453)	(U+8454)
@	/
(U+FE6B)	(U+FF0F)
?	..
(U+FE56)	(U+2025)
a	©
(U+FF41)	(U+9426)

# Normalization in action (1/5)

**https://www.evil.c[%c](#).microsoft.com**



**https://www.evil.ca[/c](#).microsoft.com**

# Normalization in action (2/5)

**https://hola@www.microsoft.com**



**https://holaawww.microsoft.com**

# Normalization in action (3/5)

[...]/documents/1?view=admin&?view=user



[...]/documents/1?view=admin&?view=user

# Normalization in action (4/5)

[...]/user/**admin**\_/info



[...]/user/**admin**//info

# Normalization in action (5/5)

[...]/documents/download/../../../../admin/secret



[...]/documents/download/../../../../admin/secret

# Normalization Recommendations

- If you need to do normalization, normalized prior to security validation
- Review your library (HTTP and Network)
- Prefer whitelist over blacklist
- Strict validation on user input stored or pass to other API



Case modification

# Case modification

What happens when « to upper case » and « to lower case » are called?

- Unicode define the behavior when characters are converted to a lower case or uppercase variant.
- Not all character have a variant.
- Many characters share the same variant [1]

[1] Reference: Why there are no uppercase character for X? (collision)  
[http://unicode.org/faq/casemap\\_charprop.html#10](http://unicode.org/faq/casemap_charprop.html#10)

# Upper case modification examples

a  
U+0061

.toUpperCase()



A  
U+0041

ß  
U+00DF

.toUpperCase()



SS  
U+0053 U+0053

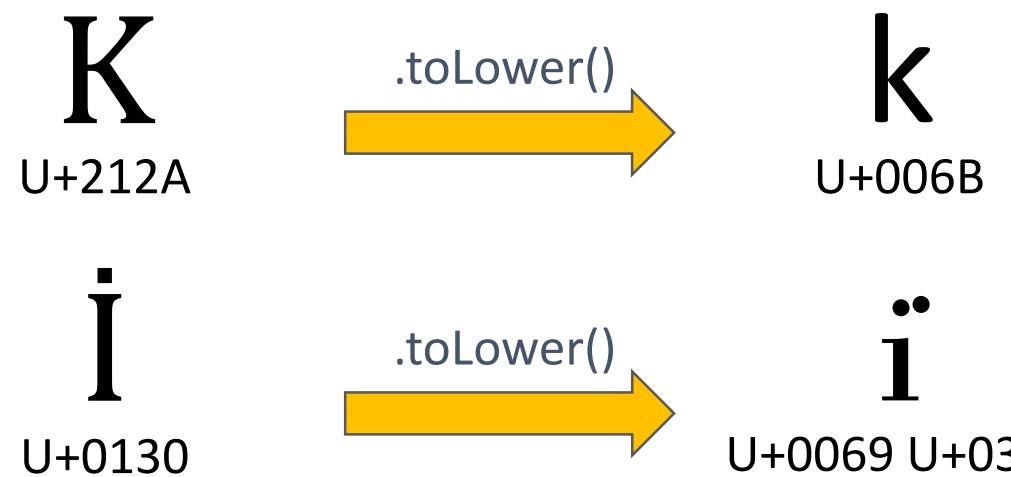
fi  
U+FB01

.toUpperCase()



FI  
U+0046 U+0049

# Lower case modification examples



# Potential issues

- Character collision
  - When doing comparison with Hostname, Username, Roles, etc.
- Bypass WAF, whitelist or blacklist

```
if current_user.upper() == "ADMIN":  
    "adm\u0131n".upper() == "ADMIN"
```

# Certificate validation weakness

```
private static boolean matchAllWildcards(String name,
    String template) {
    name = name.toLowerCase();
    template = template.toLowerCase();
    StringTokenizer nameSt = new StringTokenizer(name, ".");
    StringTokenizer templateSt = new StringTokenizer(template, ".");

    if (nameSt.countTokens() != templateSt.countTokens()) {
        return false;
    }

    while (nameSt.hasMoreTokens()) {
        if (!matchWildCards(nameSt.nextToken(),
            templateSt.nextToken())) {
            return false;
        }
    }
    return true;
}
```

# Forgot password in Django (CVE-2019-19844)

Vulnerable implementation

```
278     def save(self, domain_override=None,
279             subject_template_name='registration/password_reset_subject.txt',
280             email_template_name='registration/password_reset_email.html',
281             use_https=False, token_generator=default_token_generator,
282             from_email=None, request=None, html_email_template_name=None,
283             extra_email_context=None):
284         """
285             Generate a one-use only link for resetting password and send it to the
286             user.
287         """
288         email = self.cleaned_data["email"]
289         for user in self.get_users(email):
290             if not domain_override:
291                 current_site = get_current_site(request)
292                 site_name = current_site.name
293                 domain = current_site.domain
294             else:
295                 site_name = domain = domain_override
296             context = {
297                 'email': email,
298                 'domain': domain,
299                 'site_name': site_name,
300                 'uid': urlsafe_base64_encode(force_bytes(user.pk)),
301                 'user': user,
302                 'token': token_generator.make_token(user),
303                 'protocol': 'https' if use_https else 'http',
304                 **(extra_email_context or {}),
305             }
306             self.send_mail(
307                 subject_template_name, email_template_name, context, from_email,
308                 email, html_email_template_name=html_email_template_name,
309             )
```

super-admin@email.com

super-adm\u0130n@email.com

# Forgot password in Django (CVE-2019-19844)

Correct implementation

```
292     def save(self, domain_override=None,
293             subject_template_name='registration/password_reset_subject.txt',
294             email_template_name='registration/password_reset_email.html',
295             use_https=False, token_generator=default_token_generator,
296             from_email=None, request=None, html_email_template_name=None,
297             extra_email_context=None):
298         """
299             Generate a one-use only link for resetting password and send it to the
300             user.
301         """
302         email = self.cleaned_data["email"]
303         email_field_name = UserModel.get_email_field_name()
304         for user in self.get_users(email):
305             if not domain_override:
306                 current_site = get_current_site(request)
307                 site_name = current_site.name
308                 domain = current_site.domain
309             else:
310                 site_name = domain = domain_override
311             user_email = getattr(user, email_field_name)
312             context = {
313                 'email': user_email,
314                 'domain': domain,
315                 'site_name': site_name,
316                 'uid': urlsafe_base64_encode(force_bytes(user.pk)),
317                 'user': user,
318                 'token': token_generator.make_token(user),
319                 'protocol': 'https' if use_https else 'http',
320                 **(extra_email_context or {}),
321             }
322             self.send_mail(
323                 subject_template_name, email_template_name, context, from_email,
324                 user_email, html_email_template_name=html_email_template_name,
325             )
```

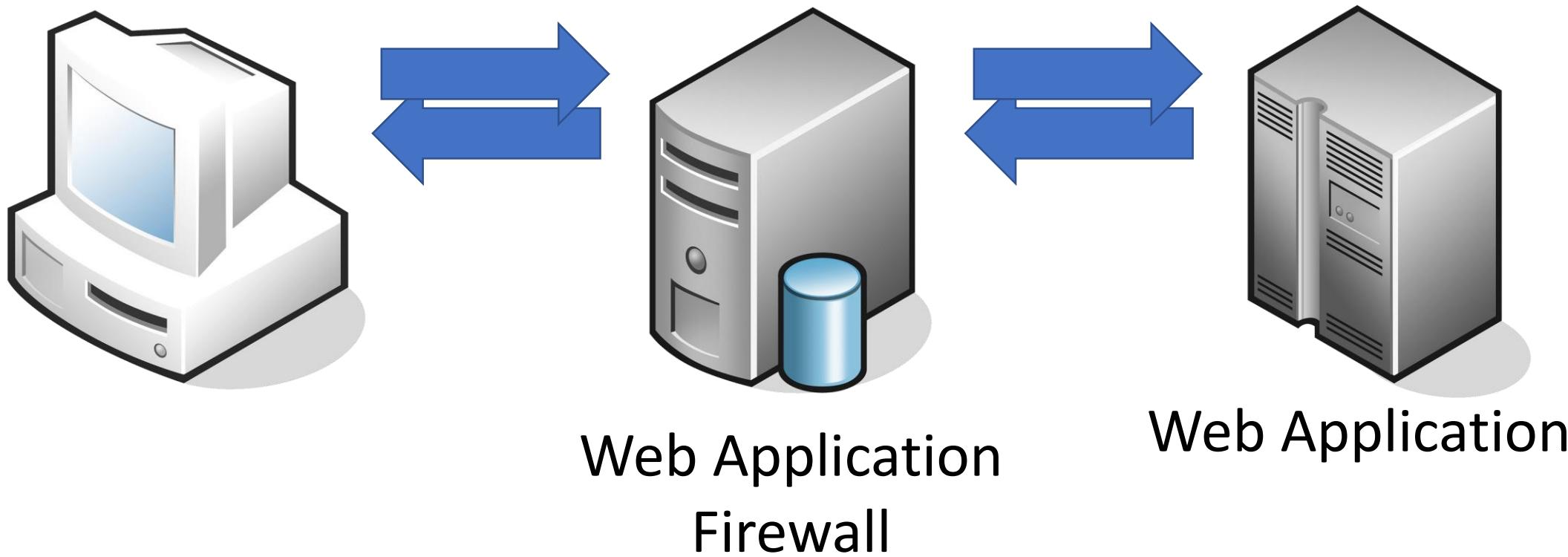
# Mitigations for case modification

- Use ASCII only lower/upper functions
- Use invariant variation if available in your language.
  - In CSharp: “user input”.ToLowerInvariant()

A close-up photograph of a soccer ball hitting a chain-link fence. The ball is in sharp focus, showing its textured surface and the impact point where it has just hit the metal mesh. The background is blurred with warm, golden-yellow lights, suggesting a stadium at dusk or night. The chain-link fence runs diagonally across the frame.

When encoding allows security  
bypasses

# Reason why security filtering can fail



# Alternative to UTF-8

Unicode is not the only way to encode characters

Here are standard encoding commonly supported

- UTF-7
- UTF-16LE
- UTF-16BE
- UTF-32...

# Malicious XML document in UTF-16

In UTF-16, every character is stored on two bytes

<book: 3c 62 6f 6f 6b => **003c 0062 006f 006f 006b**

Byte Order Mark:

- First two bytes are used to define the endianness

Encoding	Byte Order Mark (bytes in hex)
UTF-8	EF BB BF
UTF-16BE	FE FF
UTF-16LE	FF FE

*These bytes are more likely to be honored when a file is read than an HTTP request.*

# Malicious XML document in UTF-16

```
> cat book_utf16be.xml
<book xml:id="simple_book" xmlns="http://docbook.org/ns/docbook"
    version="5.0">
    <title>Very simple book</title>
    <chapter xml:id="chapter_1">
        <title>Chapter 1</title>
        <para>Hello world!</para>
        <para>
            I hope that your day is proceeding <emphasis>splendidly</emphasis>!
        </para>
    </chapter>
    <chapter xml:id="chapter_2">
        <title>Chapter 2</title>
        <para>Hello again, world!</para>
    </chapter>
</book>
```

How your application will see it

How the bytes are stored

The diagram illustrates the flow of data. On the left, a terminal window shows the XML source code. An arrow points from this window to a code editor window on the right. The code editor window displays the same XML code, but the text is highlighted in various colors (e.g., pink for tags, green for attributes, blue for URLs). A second arrow points from the code editor window back to the terminal window, indicating that the application sees the XML source code.

```
1  <book xml:id="simple_book" xmlns="http://docbook.org/ns/docbook" version="5.0">
2  |   <title>Very simple book</title>
3  |   <chapter xml:id="chapter_1">
4  |       <title>Chapter 1</title>
5  |       <para>Hello world!</para>
6  |       <para>
7  |           I hope that your day is proceeding <emphasis>splendidly</emphasis>!
8  |       </para>
9  |   </chapter>
10 |   <chapter xml:id="chapter_2">
11 |       <title>Chapter 2</title>
12 |       <para>Hello again, world!</para>
13 |   </chapter>
14 | </book>
```

# Malicious XML document with mixed encoding

- In protocols where encoding is defined, Byte Order Mark is not enough.
- XML supports changing encoding in the XML declaration section.

00000000:	3c3f 786d 6c20 7665 7273 696f 6e3d 2231	<?xml version="1
00000010:	2e30 2220 656e 636f 6469 6e67 3d22 5554	.0" encoding="UT
00000020:	462d 3136 4245 2200 3f00 3e00 3c00 6100	F-16BE".?.>.<.a.
00000030:	3e00 3100 3300 3300 3700 3c00 2f00 6100	>.1.3.3.7.<./.a.
00000040:	3e00 0a	>..

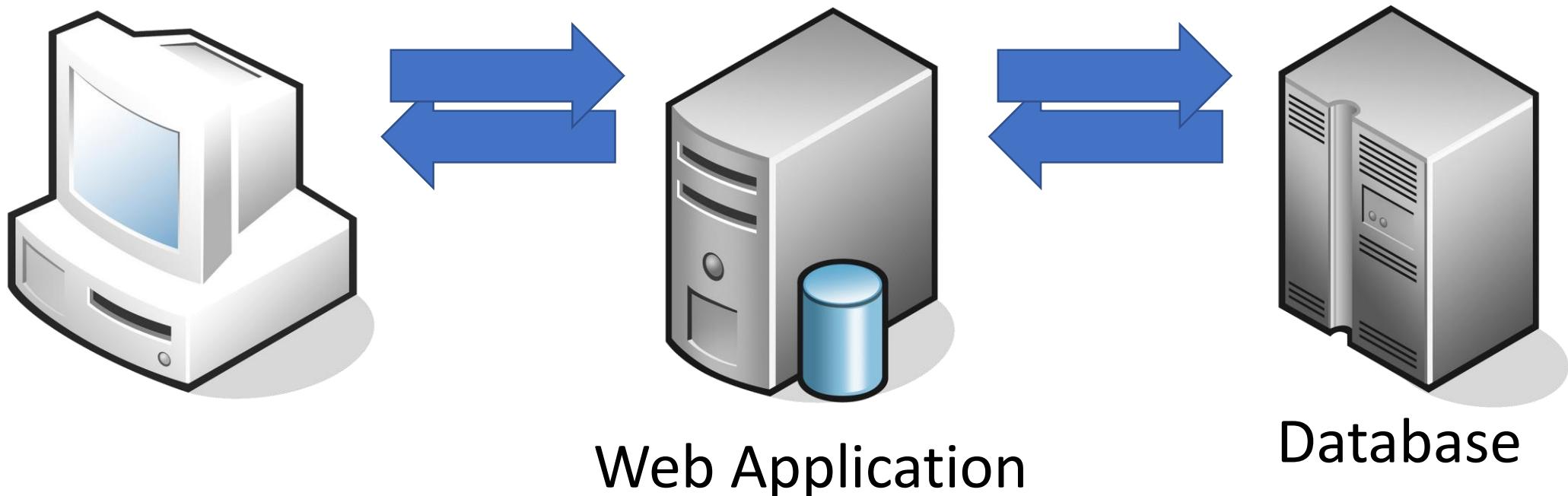
UTF-8                          UTF-16

# Other avenues

There are plenty obfuscation techniques that do not require fancy Unicode encoding (out-of-scope for this presentation)

- XML entities (inside XML) /&#x65;&#x74;&#x63;/passwd
- Double encoding %253C -> %3C -> <
- Non-printable characters java\tscript:
- Partial encoding /&#x65;tc/passwd
- Parser specific behavior

# Storing user input inside a database



# Persistent XSS inside database

*Query*

```
INSERT INTO ContentTable  
VALUES (...,'<img src=...')
```

*Data once stored*

<img src=...

U+003C

<img src=...

U+FF1C

# Homograph attacks

With punycode domains



# Punycode domains

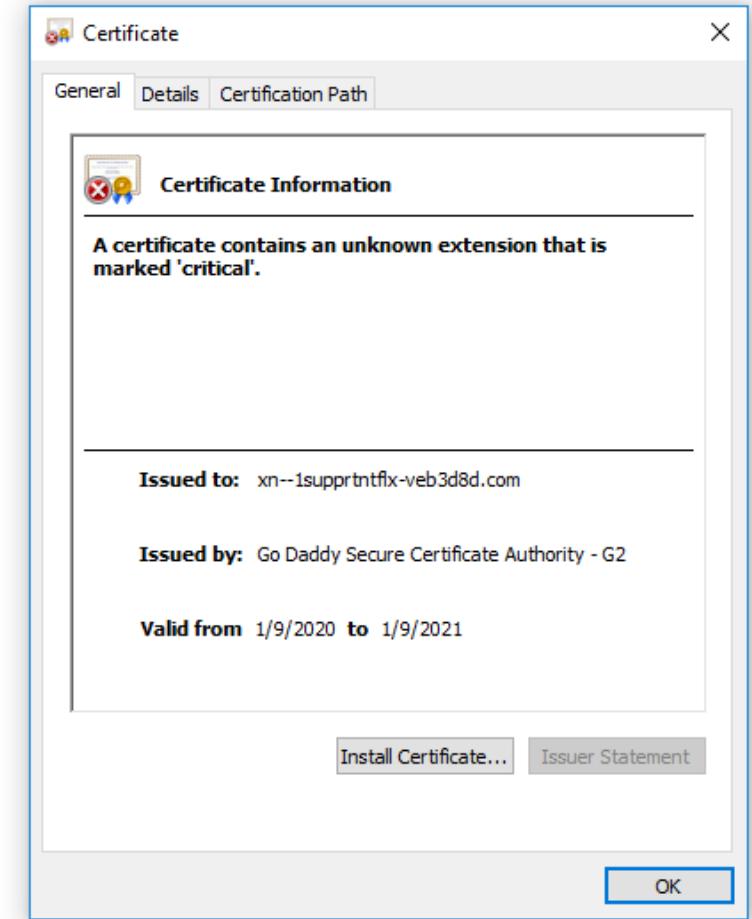
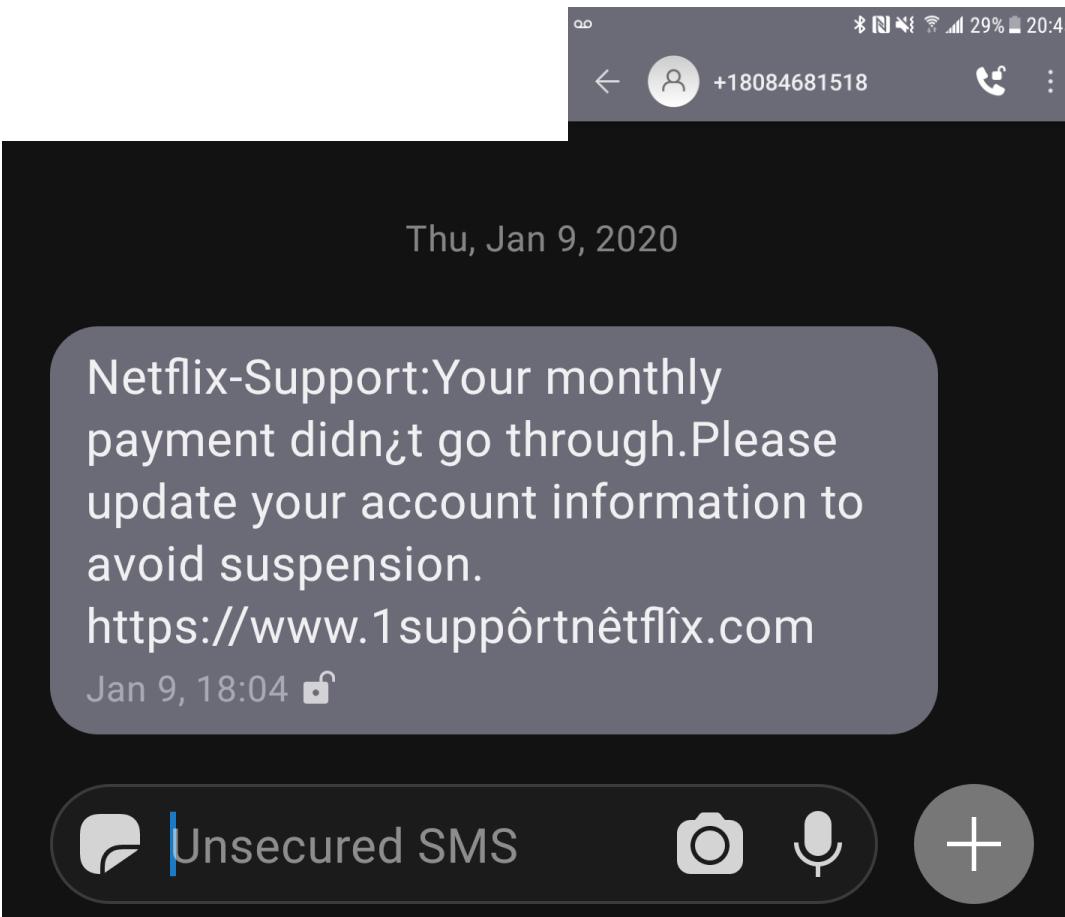
Punycode is a representation of Unicode characters using ASCII characters.

It uses the bootstrap encoding system.

Example:

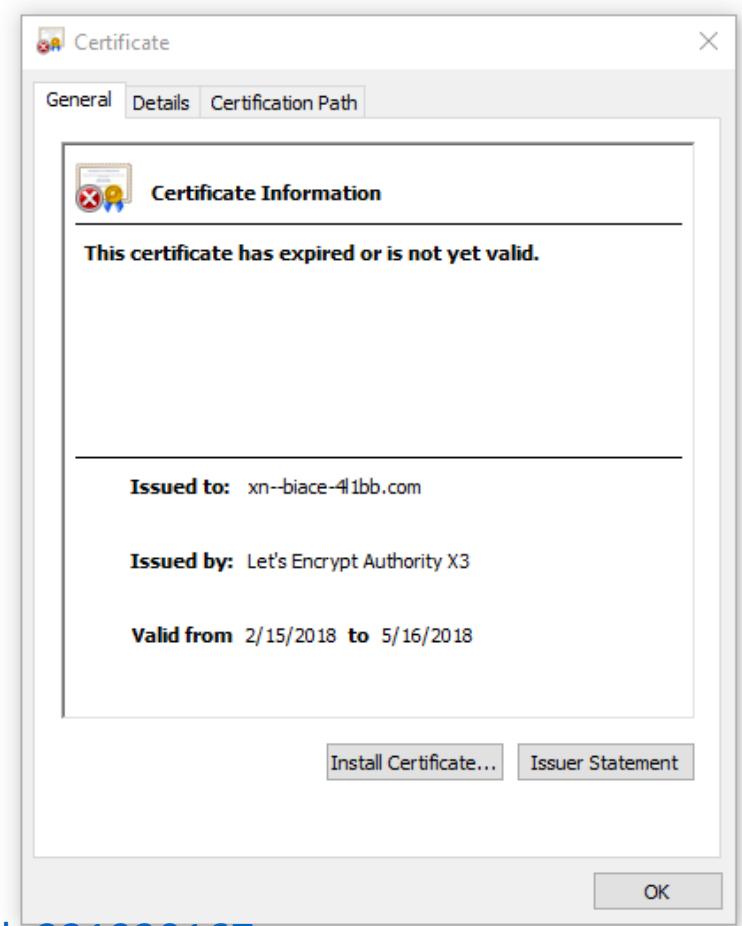
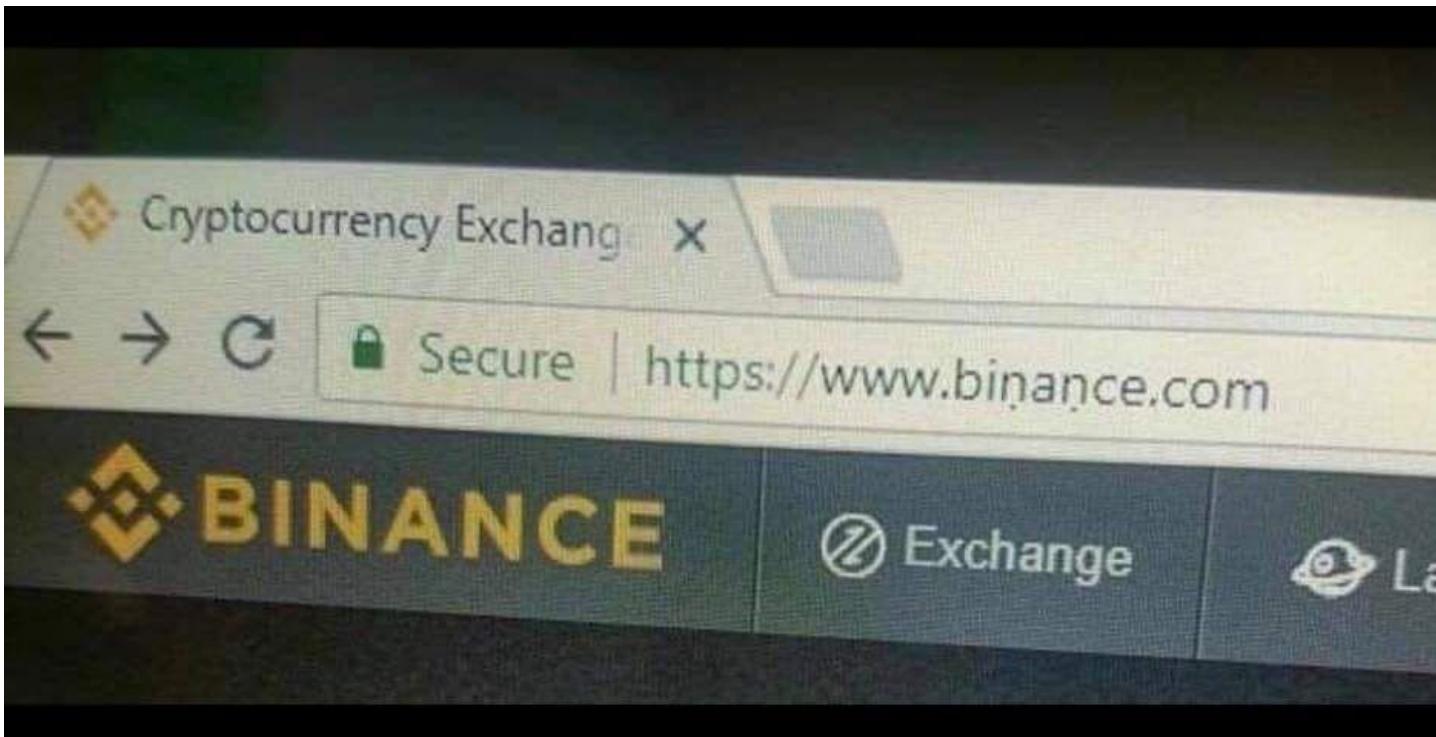
- [nörthsec.com](http://nörthsec.com)
- [xn--nrthsec-90a.com](http://xn--nrthsec-90a.com)

# Homograph SMS Phishing



Source : <https://crt.sh/?id=2310598224>

# Homograph phishing attack

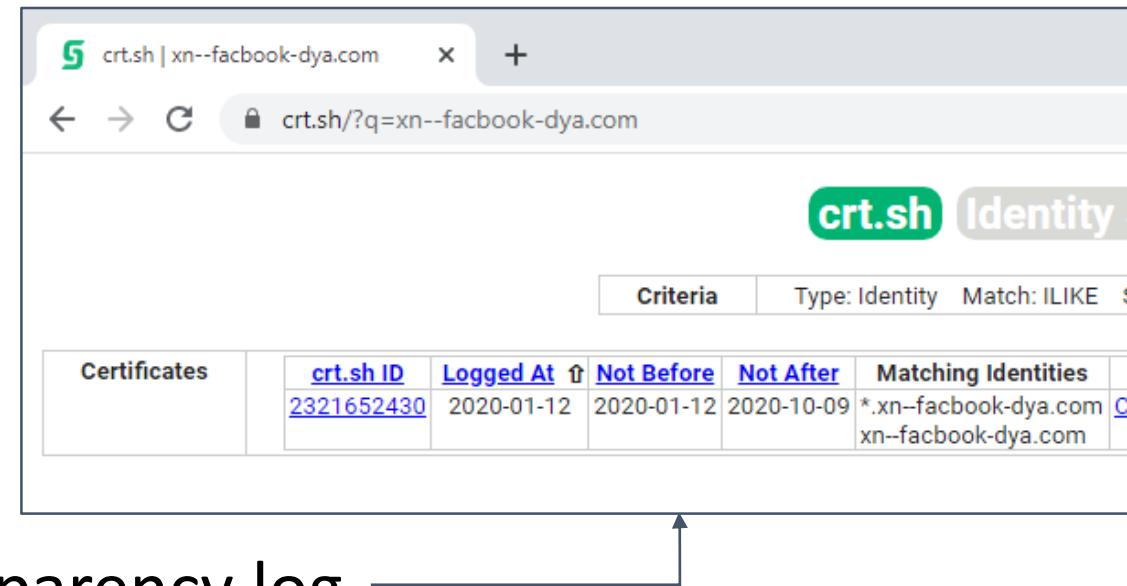


Source : <https://bitcointalk.org/index.php?topic=5184169.0> & <https://crt.sh/?id=331930167>

# Mitigations

## Defenses

- Browser defense (mostly Chrome)
- Sender Policy Framework (SPF)
- Awareness training
- Find phishing domains on the TLS transparency log



A screenshot of a web browser displaying the crt.sh website. The address bar shows 'crt.sh | xn--facbook-dya.com'. The main content area shows a table with one row of data. The table has columns for Certificates, crt.sh ID, Logged At, Not Before, Not After, and Matching Identities. The crt.sh ID column contains the value '2321652430'. An arrow points from the text 'Find phishing domains on the TLS transparency log' to the 'Not Before' column of the table.

Certificates	crt.sh ID	Logged At	Not Before	Not After	Matching Identities
	<a href="#">2321652430</a>	2020-01-12	2020-01-12	2020-10-09	*.xn--facbook-dya.com xn--facbook-dya.com

Additional Reference:

<https://us.norton.com/internetsecurity-how-to-cyber-security-best-practices-for-employees.html>

# Data integrity

Losing data because of encoding..

# Data integrity

- Improper conversion can lead to partial data loss
- If the decoder does not recognize the encoding...
  - it may raise an exception
  - or it may replace the characters silently.

'conf\xe9rence NSec' → 'conf?rence NSec'

# Do NOT use

- Java : new String(binary\_file\_content, Encoding.UTF8)
- Python : binary\_file\_content.decode("utf-8", errors="ignore")

# You SHOULD use

- Java:
  - CharsetDecoder decoder =  
Charsets.UTF\_8.newDecoder().onMalformedInput(CodingErrorAction.REPORT);  
CharBuffer decoded = decoder.decode(ByteBuffer.wrap(input));
- Python
  - binary\_file\_content.decode("utf-8")



*Conclusion*

# Recommendation

- Be cautious when using
  - Case changing function (.upper(), .lower())
  - Normalization function (UTF-8 to ASCII)
  - Internalization function (Punycode to UTF-8)
- Restrict to ASCII (CP < 128) (username, service name, id)
- Declare explicitly the encoding to avoid implicit default that vary

# Questions

## Contact information

- [parteau@gosecure.ca](mailto:parteau@gosecure.ca)
- @GoSecure\_Inc
- @h3xStream

## Slides

<https://gosecure.github.io/presentations/>





# References



# References for normalization

- Black Hat presentation of HostSplit:  
<https://i.blackhat.com/USA-19/Thursday/us-19-Birch-HostSplit-Exploitable-Antipatterns-In-Unicode-Normalization.pdf>
- GitHub vulnerabilities with lowercase issue:  
<https://eng.getwisdom.io/hacking-github-with-unicode-dotless-i/>

# References for Punycode

- Good presentation on Punycode visual attack:

<http://www.irongeek.com/i.php?page=security/out-of-character-use-of-punycode-and-homoglyph-attacks-to-obfuscate-urls-for-phishing>