



Unicode comme primitive d'attaque de bas niveau

HACKFEST

Alexandre ZANNI (@noraj)

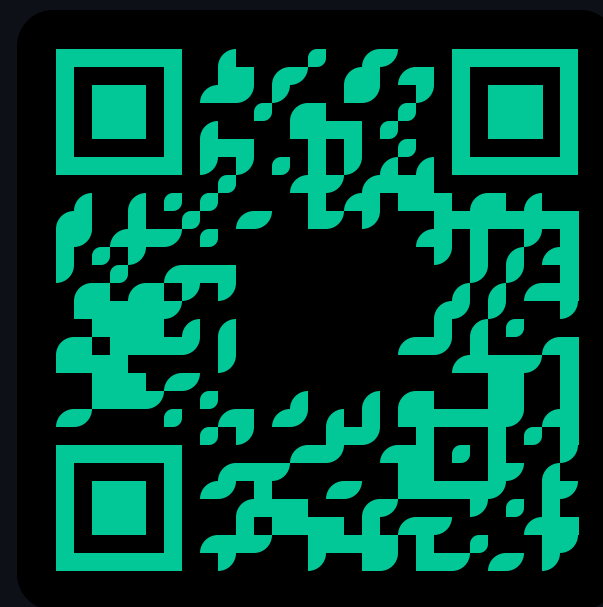
17/10/2025

Qui suis-je ?

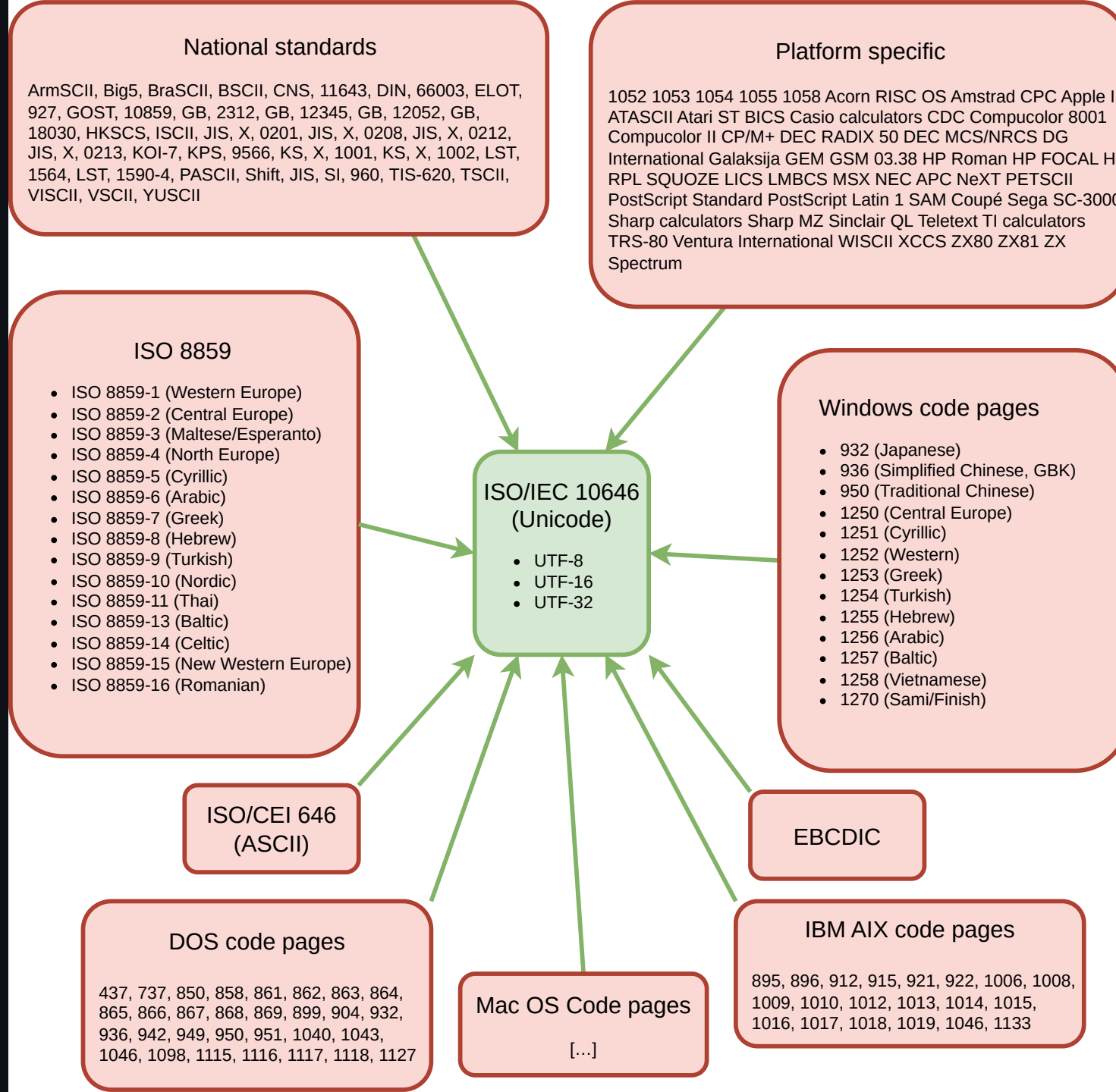
Alexandre ZANNI a.k.a **noraj**

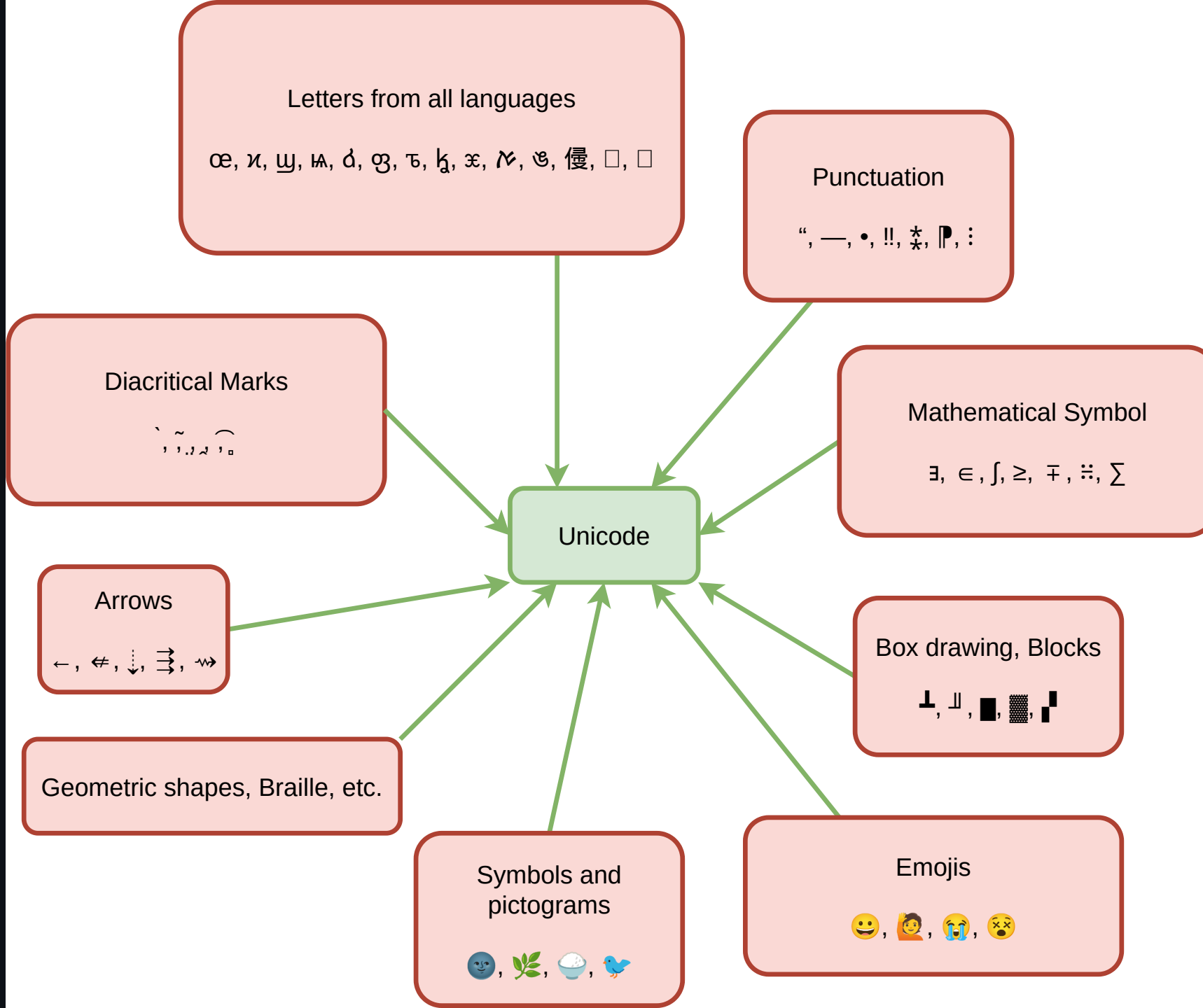
Pentester @**Synacktiv**

Mainteneur de **BlackArch** Linux

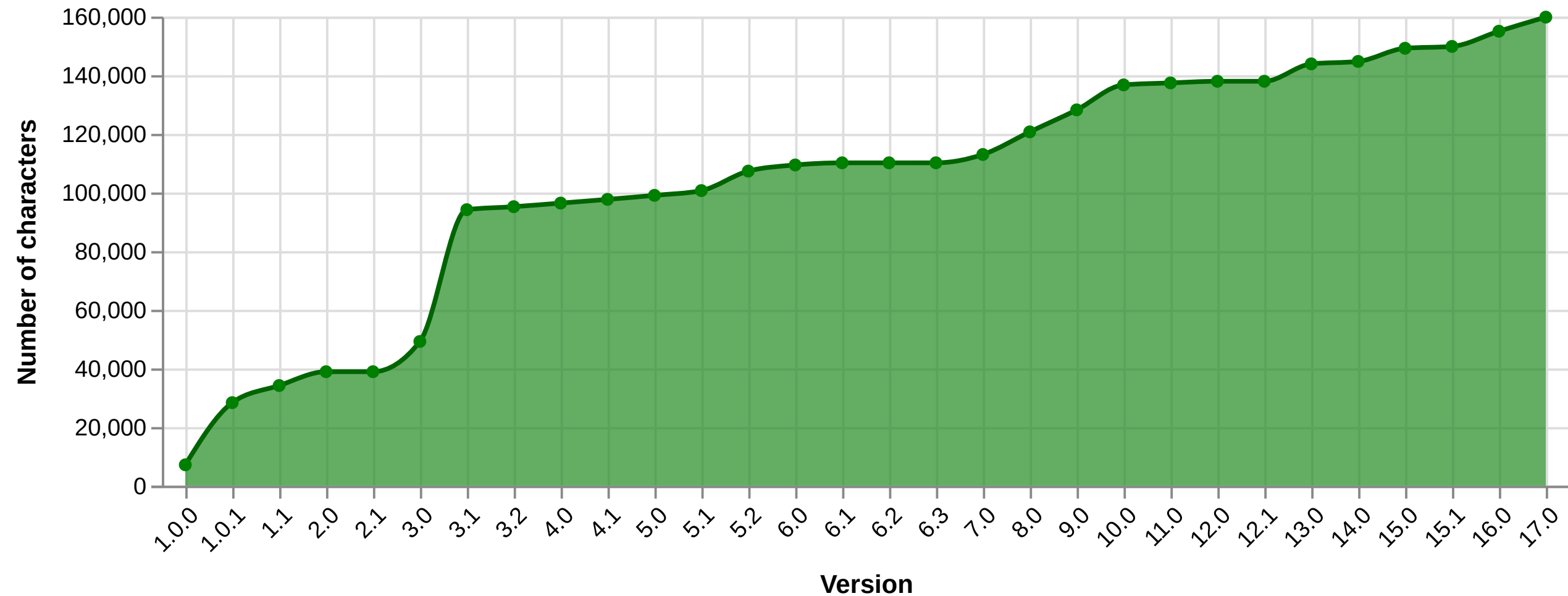


Introduction à Unicode





Number of Unicode characters per version.



- Les code de points (**Code points**) : valeurs numériques comprises entre 0 (0x0) et 1114111 (0x10FFFF), ~ 1.1M valeurs.
- Cette plage est nommée l'espace de code (**codespace**).
- Les valeurs sont notées `U+<hex_code_point>`.
- Les code points sont l'**identifiant** ou la **référence** numérique unique d'un caractère Unicode.

Unicode planes, and code point ranges used [hide]									
Basic		Supplementary							
Plane 0		Plane 1		Plane 2		Plane 3	Planes 4–13	Plane 14	Planes 15–16
0000–FFFF		10000–1FFFF		20000–2FFFF		30000–3FFFF	40000–DFFFF	E0000–EFFFF	F0000–10FFFF
Basic Multilingual Plane		Supplementary Multilingual Plane		Supplementary Ideographic Plane		Tertiary Ideographic Plane	unassigned	Supplementary Special-purpose Plane	Supplementary Private Use Area planes
BMP		SMP		SIP		TIP	—	SSP	SPUA-A/B
0000–0FFF	8000–8FFF	10000–10FFF	18000–18FFF	20000–20FFF	28000–28FFF	30000–30FFF		E0000–E0FFF	15: SPUA-A
1000–1FFF	9000–9FFF	11000–11FFF		21000–21FFF	29000–29FFF	31000–31FFF			F0000–FFFFF
2000–2FFF	A000–AFFF	12000–12FFF	1A000–1AFFF	22000–22FFF	2A000–2AFFF	32000–32FFF			
3000–3FFF	B000–BFFF	13000–13FFF	1B000–1BFFF	23000–23FFF	2B000–2BFFF	33000–33FFF			16: SPUA-B
4000–4FFF	C000–CFFF	14000–14FFF	1C000–1CFFF	24000–24FFF	2C000–2CFFF				100000–10FFFF
5000–5FFF	D000–DFFF		1D000–1DFFF	25000–25FFF	2D000–2DFFF				
6000–6FFF	E000–EFFF	16000–16FFF	1E000–1EFFF	26000–26FFF	2E000–2EFFF				
7000–7FFF	F000–FFFF	17000–17FFF	1F000–1FFFF	27000–27FFF	2F000–2FFFF				

- Représentation : A
 - Nom : Latin Capital Letter A
 - Codepoint : U+0041 (65)
- Représentation : □
 - Nom : CJK Unified Ideograph-2000B
 - Codepoint : U+2000B (131083)

Format	Bits	Encodage	Correspondance	Taille
Octet	8	UTF-8	variable (1 $\Rightarrow$ 1 to 4)	1, 2, 3 ou 4 octets
Mot	16	UTF-16	variable (1 $\Rightarrow$ 1 to 2)	2 ou 4 octets
Double-Mot	32	UTF-32	fixe (1 $\Rightarrow$ 1)	4 octets

- UTF-32 : 1 unité = code point + remplissage avec des zéro
- UTF-16 : si < BMP 1 unité = code point, sinon substitués (*surrogates*)
- UTF-8 : mécanisme plus compliqué

Substituts UTF-16

- Caractères : **A** (dans BMP)
 - Code point : U+0041 (0x41 ou 0d65)
 - UTF-16BE : **0041**
- Caractères : **𐄂** (hors BMP)
 - Code point : U+10082 (0x10082 ou 0d65666)
 - UTF-16BE : **D800 DC82**

Substituts UTF-16

```
def high_surrogate(codepoint)
  (((codepoint - 0x10000) / 0x400).floor + 0xD800)
end

def low_surrogate(codepoint)
  ((codepoint - 0x10000) % 0x400 + 0xDC00)
end

def code_point(hs, ls)
  ((hs - 0xD800) * 0x400 + ls - 0xDC00 + 0x10000)
end
```

Composition

caractères pré-composé vs caractères composite

```
"\u00E9" # => "é"  
"e\u0301" # => "é"  
"\u00E9" == "e\u0301" # => false
```

Assemblage

- modifier la forme des caractères avec des **selecteurs de variations** ou des **modifieurs**
- joindre / assembler des caractères avec des **assembleurs**

👉 + 🟰 = 🤞 (`\u{1F44C}\u{1F3FB}`)

❤️ + 🔥 = 🔥💔 (`\u{2764}\uFE0F\u200D\u{1F525}`)

🚩 + 💀 = 💀🚩 (`\u{1F3F4}\u200D\u2620\uFE0F`)

Primitive d'attaque de bas niveau

D'où viennent les problèmes ?

- Erreurs de mise en œuvre / implémentation
- Complexité des mécanismes Unicode
- Manque de sensibilisation / prise de conscience

Homoglyphes

Des glyphes qui se ressemblent mais qui sont visuellement différentes à quelques détails près alors que d'un point de vue logique ce sont des caractères entièrement différents.

Cela dépend de la police utilisée.

The image shows two overlapping outlines of the lowercase letter 'a'. The blue outline represents the Latin Small Letter A (Unicode U+0061), which has a more traditional, slightly rounded shape. The red outline represents the Mathematical Sans-Serif Small A (Unicode U+1D5BA), which is more geometric and angular, with a distinct horizontal bar at the top and a sharp, pointed bottom.

Latin Small Letter A
Unicode U+0061



Mathematical Sans-Serif Small A
Unicode U+1D5BA

Both set in Helvetica Neue

The image shows two overlapping outlines of the lowercase letter 'a'. The blue outline represents the U+0061 Latin Small Letter A, which is a standard, slightly rounded letter. The red outline represents the U+0430 Cyrillic Small Letter A, which is more angular and has a distinct horizontal bar at the top and a sharp, pointed bottom.

U+0061 LATIN SMALL LETTER A



U+0430 CYRILLIC SMALL LETTER A

Both glyphs set in Helvetica LT Std Roman at identical weight, size, and baseline.

Attaques visuelles basées sur l'homoglyphe / la confusion ?

- Nom de Domaine Internationalisé (IDN)
- Attaques spécifiques aux polices de caractères
 - Support de rendu inadéquat
 - Glyphe manquant
- Placer des portes dérobées dans du code source

Exemple piégage de code source de Certitude

```
const [ ENV_PROD, ENV_DEV ] = [ 'PRODUCTION', 'DEVELOPMENT' ];  
/* ... */  
const environment = 'PRODUCTION';  
/* ... */  
function isUserAdmin(user) {  
    if(environment!=ENV_PROD){  
        // contourner la vérification de l'auth en DEV  
        return true;  
    }  
  
    /* ... */  
    return false;  
}
```

Voyez-vous un problème ?

Dans ce contexte (`!=`)

- n'est **PAS** l'opérateur de négation `!` (point d'exclamation, U+0021)
- mais en réalité `!` (LATIN LETTER RETROFLEX CLICK, U+01C3)

Il n'a rien de spécial mais peut être utilisé dans les identificateurs (variables).

```
-environment != ENV_PROD  
+environment! = ENV_PROD
```

Caractères invisibles

Ils ne sont pas destinés à être vus par les humains, comme les caractères d'espacement, de contrôle, de remplissage.

about:newtab

no

OK



Console



Filtrer



Erreurs

Avertissements

Informations

Jo

>> var = alert;

← undefined

>> var yay = 'yes';

← undefined

>> if (yay === `no`) {}

Hangul Filler, U+3164

```
var <U+3164> = alert;
```

```
var yay = 'yes'
```

```
if (yay === <U+3164> `no`) {}  
// => alert
```


Des attaques visuelles basées sur des caractères invisibles ?

- Placer des portes dérobées dans du code source
 - Réordonnancement
 - Retours anticipés
 - Sortie de commentaires
 - Chaines de caractères étirées
 - Mélange d'isolats
 - Espaces blancs

Taille

Quel est la taille de 🧑🧑 ?

1? 6? 8? 16? 20?

La taille de quoi ?



code points : U+1F469 U+200D U+2764 U+FE0F U+200D U+1F468

- Taille en graphèmes (visuel) : **1**
- Taille de la chaîne de caractères / nombre de code unit (logique) : **6**
- Taille en octet (matériel) :
 - UTF-8: **20 bytes**
 - Hexadécimale : `f0 9f 91 a9 e2 80 8d e2 9d a4 ef b8 8f e2 80 8d f0 9f 91 a8`
 - UTF-16 (sans BOM): **16 bytes**
 - Gros-boutiste : `d83d dc69 200d 2764 fe0f 200d d83d dc68`
 - Petit-boutiste : `3dd8 69dc 0d20 6427 0ffe 0d20 3dd8 68dc`

- Taille en octet :
 - UTF-32 (sans BOM) : **24 bytes**
 - Gros-boutiste : `0001f469 0000200d 00002764 0000fe0f 0000200d 0001f468`
 - Petit-boutiste : `69f40100 0d200000 64270000 0ffe0000 0d200000 68f40100`

```
// nodejs - javascript - UTF-16
a.length // 8, à cause des surrogates

[...a].length // 6

// Retourne 20 à la place de 16
// nodejs
Buffer.byteLength(a, 'utf16le') // 16
Buffer.byteLength(a, 'utf16be') // 20
// javascript (UTF-8)
(new TextEncoder().encode(a)).length // 20

[...new Intl.Segmenter().segment(a)].length // 1
```

```
a = '👤'  
a.size # 6 (nombre de code points)  
a.bytesize # 20 (nombre d'octets en UTF-8)  
a.grapheme_clusters.size # 1 (nombre de graphèmes)
```

Impactes ?

- Débordement de mémoire tampon
- Troncature des chaînes de caractères
- Accès incorrect à l'index
- Assainissement mal placé

Transformation de casse

- ~~bijectif : $1 \Rightarrow 1$~~
- $1 \Rightarrow n$
- $n \Rightarrow 1$
- contextuel (dépend de ce qui placé autour)
- sensible à la localisation (paramètres régionaux)

Peut entraîner :

- Modifier la longueur de la chaîne de caractères
- Création de collisions
- Inadéquation des entrées
- Contournement de la sécurité

Collisions par transformation de casse (majuscule)

```
a = 'ß'  
len(a) # 1  
a.upper() # SS  
len(a.upper()) # 2
```

ß (LATIN SMALL LETTER SHARP S, U+00DF)

↓

2 × S (LATIN CAPITAL LETTER S, U+0053).

Collisions par transformation de casse (minuscule)

```
kelvin = 'K'  
k = 'K'  
kelvin == k # False  
kelvin.lower() == k.lower() # True
```

K (KELVIN SIGN, U+212A)

⇓

k (LATIN SMALL LETTER K, U+006B) when converted to lowercase.

Transformation contextuelle (correspondance de casse sensible au contexte)

Σ (GREEK CAPITAL LETTER SIGMA, U+03A3)

Σ	⇒	σ
Σa	⇒	σa
aΣ	⇒	aς
ΣΣ	⇒	σς

- **σ** (GREEK SMALL LETTER SIGMA, U+03C3) si la lettre est suivie par une lettre min/maj (ou seule)
- **ς** (GREEK SMALL LETTER FINAL SIGMA, U+03C2) sinon

Transformation sensible à la langue (correspondance de casse sensible à la langue)

```
"I".toLocaleLowerCase("fr-FR")  
// 'i'  
"I".toLocaleLowerCase("az-AZ")  
// 'ı'  
"i".toLocaleUpperCase("fr-FR")  
// 'I'  
"i".toLocaleUpperCase("tr-TR")  
// 'İ'
```

Exemple de prise de contrôle d'un compte par réinitialisation du mot de passe en utilisant les collisions de transformation de la casse (minuscules)

```
app.post('/api/password/reset', function(req, res) {  
  var email = req.body.email;  
  db.get('SELECT id, email, FROM users WHERE email = ?',  
    [email.toLowerCase()],  
    (err, user) => {  
      if (err) {  
        console.error(err.message);  
        res.status(400).send();  
      } else {  
        generateTemporaryPassword((tempPassword) => {  
          accountRepository.resetPassword(user.id, tempPassword, () => {  
            messenger.sendPasswordResetEmail(email, tempPassword);  
            res.status(204).send();  
          });  
        });  
      }  
    });  
});
```




```
email.toLowerCase()
```

```
Admin@synacktiv.com ⇒ admin@synacktiv.com
```

```
> a = "K";  
'K'  
> b = "K";  
'K'  
> a == b  
false  
> a.toLowerCase() == b.toLowerCase()  
true
```



```
email.toLowerCase()
```

```
admin@synackTiv.com ⇒ admin@synacktiv.com
```

- Sélectionne `email.toLowerCase()` en BDD
 - Données de l'utilisateur transformées
 - L'adresse courriel de l'utilisateur légitime est sélectionnée en BDD (et donc son jeton de réinitialisation du mot de passe).
- Envoi un courriel à `email` ☠
 - Entrée utilisateur non modifiée



Implicit Unicode behaviors in database string functions

Article (p. 73) dans le magazine **Paged Out!** n°6

Autre exemple pour for MariaDB / MySQL

Normalisation

Canonique vs Compatible

Form	Description
Normalization Form D (NFD)	Canonical Decomposition
Normalization Form C (NFC)	Canonical Decomposition, followed by Canonical Composition
Normalization Form KD (NFKD)	Compatibility Decomposition
Normalization Form KC (NFKC)	Compatibility Decomposition, followed by Canonical Composition

Source		NFD		NFC		NFKD		NFKC
fi FB01	:	fi FB01		fi FB01		f i 0066 0069		f i 0066 0069
2 ⁵ 0032 2075	:	2 5 0032 2075		2 5 0032 2075		2 5 0032 0035		2 5 0032 0035
fi 1E9B 0323	:	f ̇ ̇ 017F 0323 0307		fi ̇ 1E9B 0323		s ̇ ̇ 0073 0323 0307		§ 1E69

2 exemples d'attaques utilisant la normalisation

- Fractionnement de l'hôte
- Contournement du filtre XSS

Fractionnement de l'hôte

Abus des modes de normalisation compatibles pour contourner la liste blanche des URL

Technique présentée par *Jonathan Birch* à *Black Hat USA 2019* (*Host/Split
Exploitable Antipatterns in Unicode Normalization*)

Avec NFKD et NFKC

Addressed To the Subject

`a/s` (U+2101) $\Rightarrow$ `a` (U+0061) + `/` (U+002F) + `s` (U+0073)

`a/s` $\Rightarrow$ `a/s`

```
url = 'https://synacktiv.c%support.target.com'  
url.unicode_normalize(:nfkd)  
# => "https://synacktiv.ca/support.target.com"  
url.unicode_normalize(:nfkc)  
# => "https://synacktiv.ca/support.target.com"
```

Va contourner `*.target.com` et mener à `synacktiv.ca`

Dans ce cas, il suffit à l'attaquant d'enregistrer un domaine avec un TLD se terminant par

a : `.ca` , `.media` , `.ninja` , `.pizza` , `.mba` , `.moda`

Séparateurs similaires :

- ₪ (U+2100)
- ₯ (U+2105)
- ₧ (U+2106)

Plus de caractères pour jouer avec les URL :

- `?! (U+2048) ⇒ ?!` , paramètres GET
- `/ (U+FF0F) ⇒ /` , similaire à `%` mais sans contrainte 🤪
- `# (U+FF03) ⇒ #` , neutralise des choses avec l'ancre
- `@ (U+FF20) ⇒ @` , nom d'utilisateur
 - `( utilisateur@cdn.cible.com )`

- `:` (U+FF1A) ⇒ `:`, identifiants
 - (`utilisateur:motdepasse@secret.cible.com`)
- `1.` (U+2488) ⇒ `1.` fabriquer des adresses IP

Cas d'usage :

- Parsers d'URL
- Validation avec regexp
- Navigateur web ou serveur avec l'en-tête HTTP `Location` & redirection vers l'entrée normalisée

Contournement de filtre XSS

Les fonctions d'échappement HTML permettent d'échapper aux caractères spéciaux HTML tels que `<`, `>`, `"`, `&`, etc.

Avec NFKD et NFKC

Contourner < (U+003C) avec :

- < (U+FE64)
- < (U+FF1C)

Contourner > avec :

- > (U+FE65)
- > (U+FF1E)

Contourner " (U+0022) avec :

- " (U+FF02)

Contourner & (U+0026) avec :

- & (U+FE60)
- & (U+FF06)

```
test = '<' # U+FE64
test.unicode_normalize(:nfkc) # => "<" (U+003C)
test.unicode_normalize(:nfkd) # => "<" (U+003C)
test = '>' # U+FF1E
test.unicode_normalize(:nfkc) # => ">" (U+003E)
test.unicode_normalize(:nfkd) # => ">" (U+003E)
```

Charge utile normale

```
<script>alert(document.cookie)</script>
```

↓ échappement HTML

```
&lt;script&gt;alert(document.cookie)&lt;/script&gt;
```

↓ charge utile échappée, pas vuln ✓

Charge utile Unicode

```
<script>alert(document.cookie)</script>
```

↓ échappement HTML

```
<script>alert(document.cookie)</script>
```

↓ charge utile pas échappée, mais caractères non interprétés en HTML, pas vuln 

Charge utile Unicode

```
<script>alert(document.cookie)</script>
```

↓ échappement HTML + (NFKC, NFKD)

```
<script>alert(document.cookie)</script>
```

↓ charge utile mutée (mXSS), caractères interprétés en HTML, vuln ✗

Cas d'usages :

- Ordre incorrect des étapes de filtrage par le développeur.
- Normalisation effectuée implicitement par le cadriciel du SGBD

Exemple : Dans toutes les applications qui font

```
normalizeNFKC(escapeHTML(user_param))
```

au lieu de

```
escapeHTML(normalizeNFKC(user_param))
```

Charge utile Unicode

```
<script>alert(document.cookie)</script>
```

↓ (NFKC, NFKD) + échappement HTML

```
&lt;script&gt;alert(document.cookie)&lt;/script&gt;
```

↓ charge utile échappée, pas vuln 

Exemple en Ruby (cadriceil Roda)

```
response.write CGI.escapeHTML(r.params['name']).unicode_normalize(:nfkc)
```

Exemple en Java (cadriceil FreeMaker)

```
${normalizeNFKC(request.queryParameters.name!?.html)}
```

Avec NFC

`>` suivi de U+0338 sera recomposé en `>`

⇒ annule la fin d'une balise HTML

```
<textarea id=desc>INJECTION_ICI</textarea>
<!-- ↓ injection -->
<textarea id=desc>U+0338 autofocus onfocus=alert(document.cookie) </textarea>
<!-- ↓ normalisé -->
<textarea id=desc>/ autofocus onfocus=alert(document.cookie) </textarea>
```

Exemple en Ruby (cadriciel Roda)

```
response.write ('<textarea id=noraj>' +  
                CGI.escapeHTML(r.params['param']) +  
                '</textarea>').unicode_normalize(:nfc)
```

Exemple en Java (cadriciel FreeMaker)

```
${normalizeNFC("<textarea id=noraj>" +  
               request.queryParameters.param!?.html +  
               "</textarea>")}
```

Bonus

unisec

Boite à outil pour la recherche en
sécurité sur Unicode

github.com/noraj/unisec



Mode SQL strict de MySQL

Article : *Que pourrait-il arriver de dangereux lorsque le mode SQL strict de MySQL est désactivé ?* à retrouver sur <https://www.synacktiv.com/publications>

Le mode strict contrôle la manière dont MySQL gère les **valeurs invalides** ou manquantes dans les **instructions de modification** de données telles que `INSERT` ou `UPDATE`.

Mode strict activé	Mode strict désactivé
Erreur	Avertissement (silencieux)
Rien n'est modifié	Une valeur « ajustée » (« valeur la plus proche ») est insérée

« convertir la valeur invalide en la valeur valide la plus proche » ?

- les chaînes de caractères plus longues que la taille de la colonne seront tronquées
- les entiers arrondis aux valeurs les plus proches
- etc.
- pour les valeurs qui ne peuvent être représentées dans l'encodage, chaque octet est remplacé par un ?

```
-- Créer une table avec une colonne utilisant volontairement
-- un encodage « étroit »
CREATE TABLE uni_sandbox (
  id INT AUTO_INCREMENT PRIMARY KEY,
  data VARCHAR(255) CHARACTER SET ascii
);

-- Mode strict activé
SET sql_mode='STRICT_ALL_TABLES';
INSERT INTO uni_sandbox (data) VALUES ('I ♥ Unicode');
-- => ERROR 1366 (22007): Incorrect string value: '\xE2\x99\xA5 Un...'
-- for column `unicode8`.`uni_sandbox`.`data` at row 1

-- Mode strict désactivé
SET sql_mode='';
INSERT INTO uni_sandbox (data) VALUES ('I ♥ Unicode');
SELECT * FROM uni_sandbox\G;
-- => data: I ? Unicode
```

Contexte

- App filtre avec regexp (accepte que lettres)
- App avec Unicode, BDD avec encodage « étroit »
- Mode SQL strict MySQL désactivé

classes de caractères POSIX étendues à Unicode

```
/[[:alnum:]]/.match('𐀀')  
# => #<MatchData "𐀀">
```

sélecteurs de classes de caractères personnalisés (non-POSIX)

```
"𐀀".match(/\p{Alpha}/u)  
// Array [ "𐀀" ]
```

Contournements

Expansion de nom de fichier shell

```
$ ls -d /???t  
/boot /root
```

Si lecture fichier avec IDOR et fichiers `UUID.txt` ➡ impossible de BF

lire `aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaaaaaa.txt`

Contournements

Quantificateur d'expression régulière

```
fichier\d?\.txt
```

```
fichier.txt  
fichier1.txt  
...  
fichier9.txt
```

```
import re

username = User.username # données de l'utilisateur extraites de la base de données
check(username, lib.alphanum) # une vérification alphanumérique Unicode
re.findall(f'confidential-{username}\.pdf', 'list-files-fetched-fromFS-or-DB', re.IGNORECASE)
```

aꠤbꠤ...yꠤzꠤ0ꠤ1...8ꠤ9ꠤ



a?b?...y?z?0?1?...8?9?

```
import re
import string

payload = "".join(map(lambda i: i + '?', string.ascii_lowercase + string.digits)) * 5
re.findall(f'confidential-{payload}\.pdf', 'confidential-noraj.pdf', re.IGNORECASE)
# => ['confidential-noraj.pdf']
```

Contournements

Paramètre de requête

`noraj␣debug` → `noraj?debug`

Contournements

Pare-feu applicatif (WAF)

`<␣php`  `<?php`

Histoire réelle

- 2012
- Wordpress désactivait le mode SQL strict lors de l'installation
- UTF-8 ↔ utf8mb3 (1 à 3 octets au lieu de 4)
- mode SQL strict désactivé = troncation (au lieu de remplacement par ?)
- 2014 : XSS stockée dans Wordpress dans les commentaires



<https://www.linkedin.com/company/synacktiv>



<https://x.com/synacktiv>



<https://synacktiv.com>