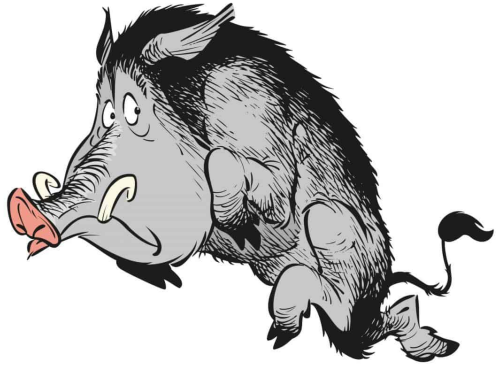
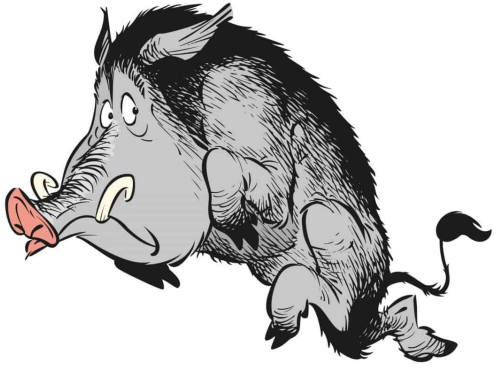
« Don't roll your own crypto » : les méthodes formelles au service de la sécurité

Théophile Wallez, *Inria Paris*

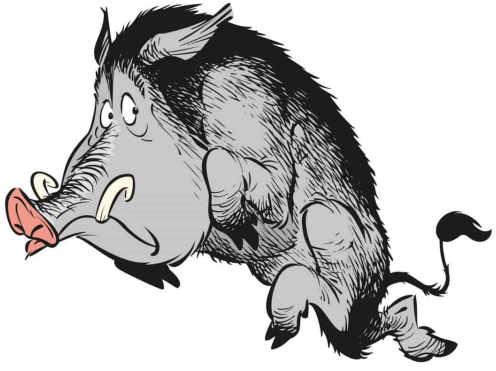
Qui suis-je



Qui suis-je



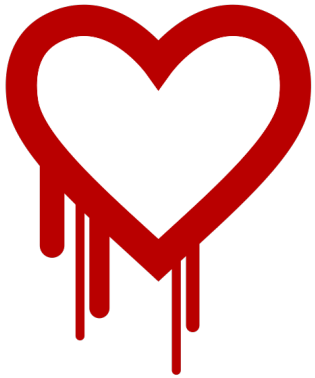
Qui suis-je



Des bugs en cryptographie

Heartbleed

Heartbleed



OpenSSL
Cryptography and SSL/TLS Toolkit

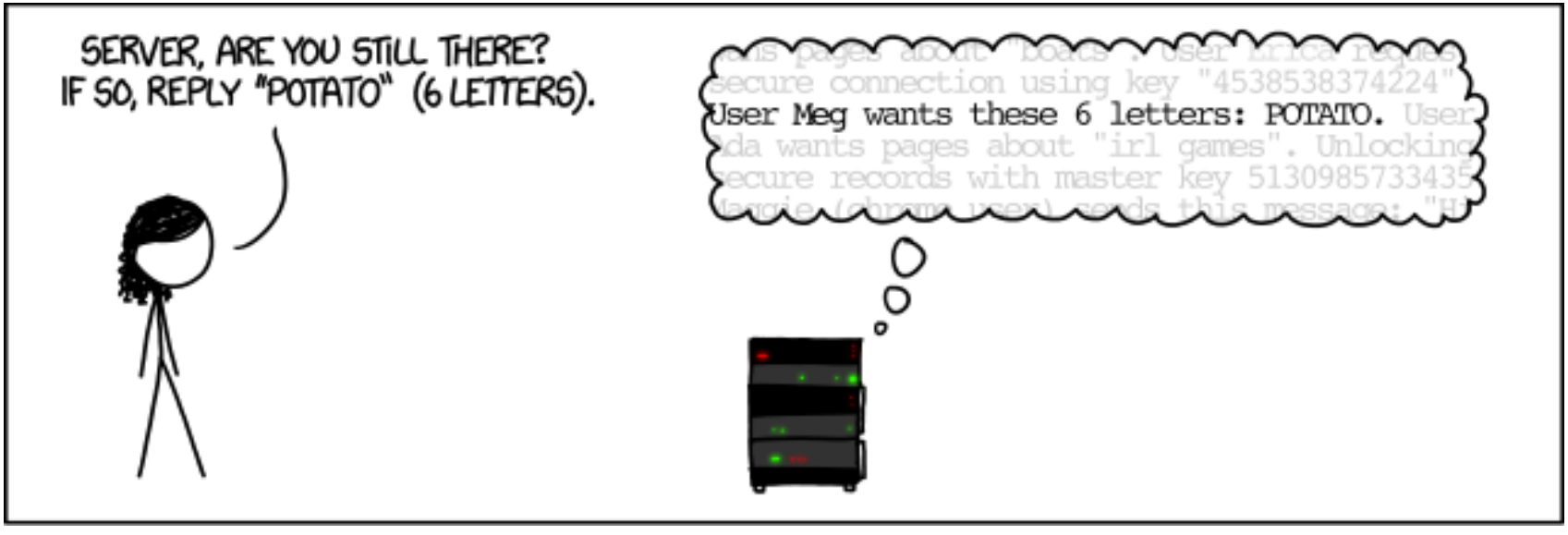
https://www.

https://heartbleed.com/

Heartbleed

OpenSSL
Cryptography and SSL/TLS Toolkit

https://www.



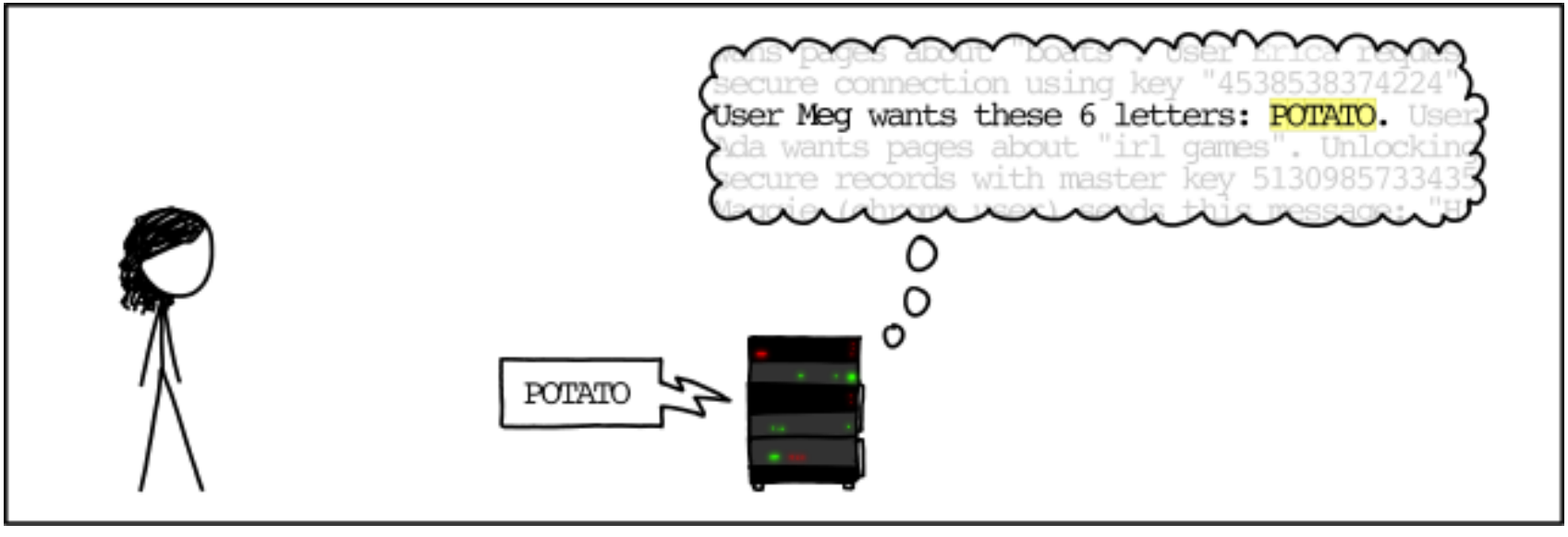
https://xkcd.com/1354/

https://heartbleed.com/

Heartbleed

OpenSSL
Cryptography and SSL/TLS Toolkit

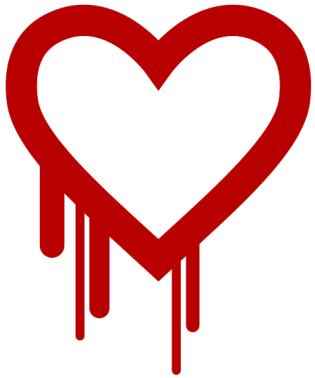
https://www.



https://xkcd.com/1354/

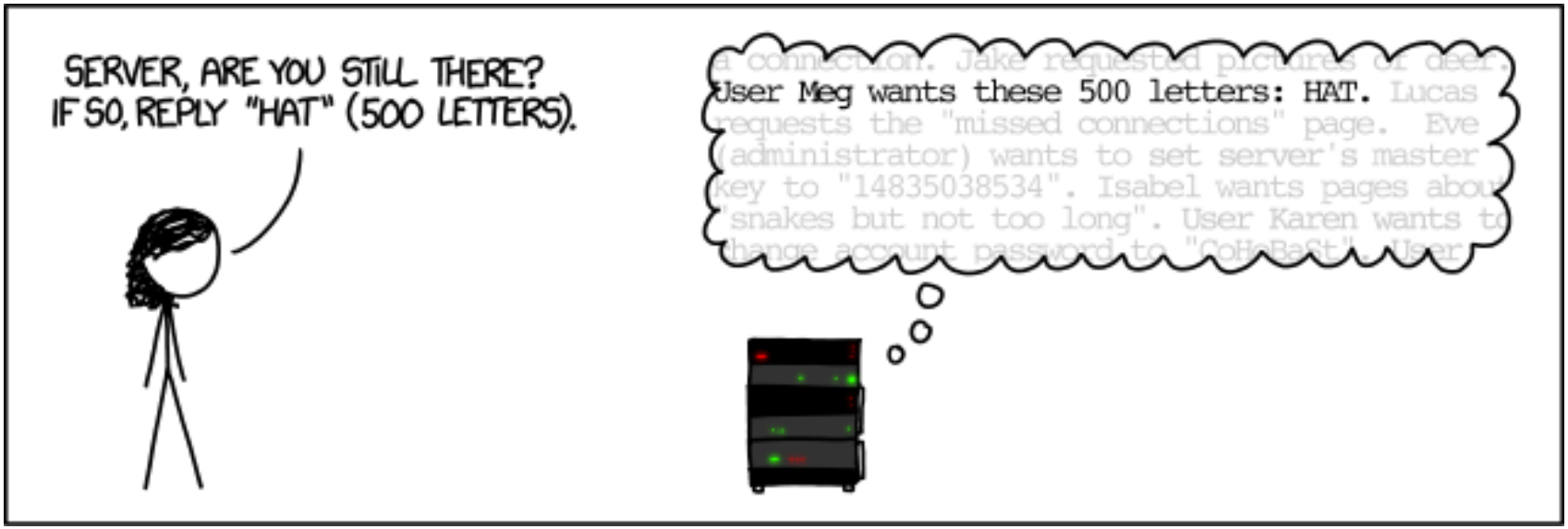
https://heartbleed.com/

Heartbleed



OpenSSL
Cryptography and SSL/TLS Toolkit

https://www.



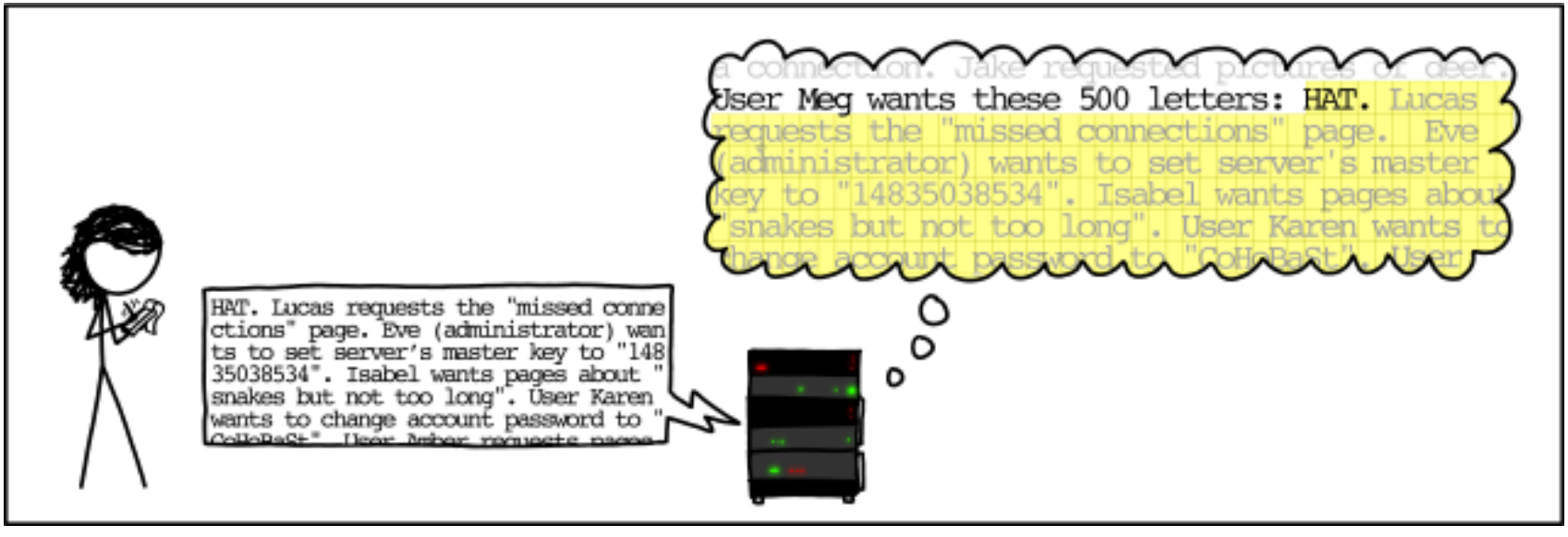
https://xkcd.com/1354/

https://heartbleed.com/

Heartbleed

OpenSSL
Cryptography and SSL/TLS Toolkit

https://www.



https://xkcd.com/1354/

https://heartbleed.com/

SHA-3

Bug dans l'implémentation officielle de SHA-3, découvert après 10 ans !

https://mouha.be/sha-3-buffer-overflow/

SHA-3

Bug dans l'implémentation officielle de SHA-3, découvert après 10 ans !

```
import hashlib
h = hashlib.sha3_224()
h.update(b"\x00" * 1)
h.update(b"\x00" * 4294967295)
print(h.hexdigest())
```

Segmentation fault
dans Python !

https://mouha.be/sha-3-buffer-overflow/

SHA-3

Bug dans l'implémentation officielle de SHA-3, découvert après 10 ans !

```python
import hashlib
h = hashlib.sha3_224()
h.update(b"\x00" * 1)
h.update(b"\x00" * 4294967295)
print(h.hexdigest())
```

Segmentation fault dans Python !

```c
const u32 BUF_SIZE = ...;
u8 buf[BUF_SIZE];
// ...
u32 x, y;
// ...
if(x+y < BUF_SIZE) {
    memcpy(&buf[x], ..., y);
}
```

Les bornes sont testées, tout va bien ?

https://mouha.be/sha-3-buffer-overflow/

SHA-3

Bug dans l'implémentation officielle de SHA-3, découvert après 10 ans !

```python
import hashlib
h = hashlib.sha3_224()
h.update(b"\x00" * 1)
h.update(b"\x00" * 4294967295)
print(h.hexdigest())
```

Segmentation fault
dans Python !

```
const u32 BUF_SIZE = ...;
u8 buf[BUF_SIZE];
// ...
u32 x = 1, y = 4294967295;
// ...
if(x+y < BUF_SIZE) {
    memcpy(&buf[x], ..., y);
}
```

Les bornes sont testées,
tout va bien ?

Raté : x+y = 0 par overflow.
La copie est effectuée :
buffer overflow.

https://mouha.be/sha-3-buffer-overflow/

Matrix

Problème dans la spécification du protocole
$\rightarrow$ toutes les implémentations sont affectées.

`https://nebuchadnezzar-megolm.github.io/`

Paypal

Paypal

Méthode pour doubler l'argent d'un compte
Utilise trois comptes, avec des transferts et demande de remboursement.

https://thehackernews.com/2014/06/loophole-in-paypal-terms-allows-anyone.html

Paypal

Méthode pour doubler l'argent d'un compte
Utilise trois comptes, avec des transferts et demande de remboursement.

Paypal refuse le bug bounty :
pas de bug dans l'application web, le problème vient des CGU.

https://thehackernews.com/2014/06/loophole-in-paypal-terms-allows-anyone.html

En résumé

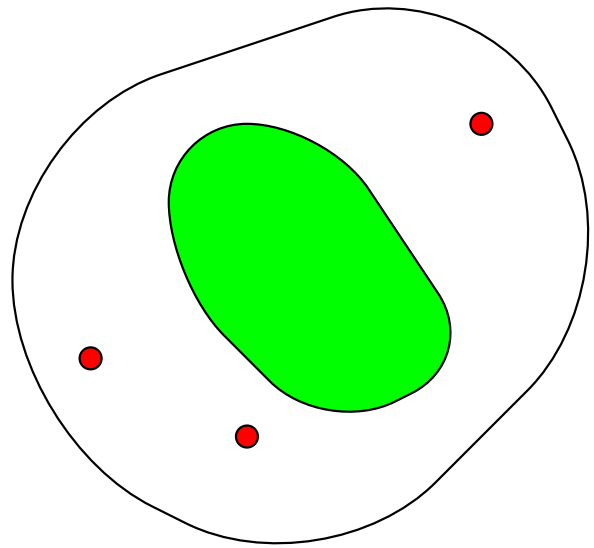
Implémentation : ce que le programme fait
Spécification : ce qu'on veut que le programme fasse

Implémentation	Spécification
Heartbleed	Matrix
SHA-3	Paypal

En résumé

Implémentation : ce que le programme fait
Spécification : ce qu'on veut que le programme fasse

Implémentation	Spécification
Heartbleed	Matrix
SHA-3	Paypal



Utilisabilité : aucun bug sur les entrées utilisateurs
Sécurité : aucun bug sur n'importe quelle entrée

Éliminer les bugs avec les méthodes formelles

Éliminer les bugs ?

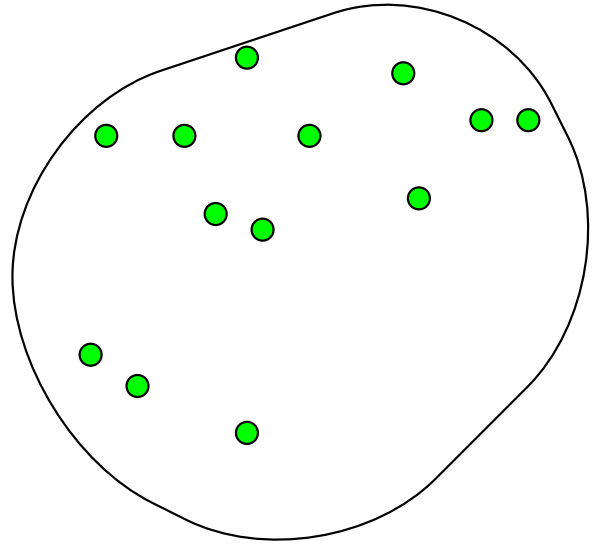
Éliminer les bugs ?

Éliminer **des classes de** bugs

- memory safety (pas de buffer overflow, segfault, . . .)
- sécurité cryptographique (pas d'attaque révélant des secrets)
- le programme n'arrive pas dans un état incohérent (e.g. deadlock)
- autre (e.g. Paypal ne perd pas d'argent)

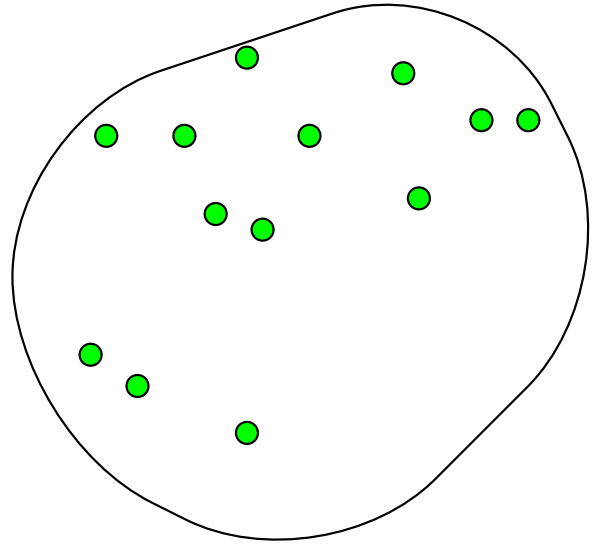
Le test, le fuzz : pas une solution

Le test garanti que le programme se comporte bien sur un nombre fini d'entrées.

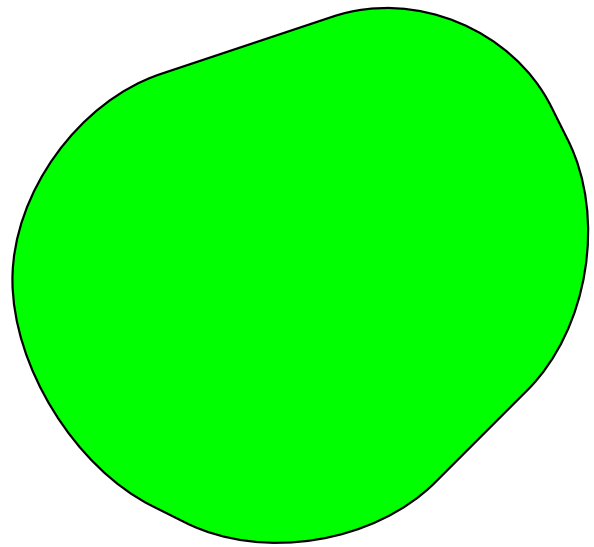


Le test, le fuzz : pas une solution

Le test garanti que le programme se comporte bien sur un nombre fini d'entrées.



On veut que le programme se comporte bien sur **toutes** les entrées (en nombre infini).



Memory safety

70% des bugs critiques de Google Chrome sont liés à la memory safety.
La moitié (35%) sont des use-after-free.

https://www.chromium.org/Home/chromium-security/memory-safety/

Memory safety

70% des bugs critiques de Google Chrome sont liés à la memory safety.
La moitié (35%) sont des use-after-free.

Solution 1 : utiliser des langages memory-safe (facile !)

- les langages avec garbage-collector
- Rust

https://www.chromium.org/Home/chromium-security/memory-safety/

Memory safety

70% des bugs critiques de Google Chrome sont liés à la memory safety.
La moitié (35%) sont des use-after-free.

Solution 1 : utiliser des langages memory-safe (facile !)

- les langages avec garbage-collector
- Rust

Solution 2 : utiliser des langages pas memory-safe (e.g. C, C++),
mais utiliser des outils vérifier que l'on ne fait pas de bêtises
(e.g. Astrée, Frama-C)

https://www.chromium.org/Home/chromium-security/memory-safety/
https://www.absint.com/astree/index.htm
https://www.frama-c.com/

Propriété de spécifications

Faisabilité : selon la complexité de la spécification, et des propriétés

Propriété de spécifications

Faisabilité : selon la complexité de la spécification, et des propriétés

Sécurité de protocoles cryptographiques : de nombreux outils.
Utilisés dans l'industrie : ProVerif et Tamarin.
Succès industriel : TLS 1.3 ! https://www.

Propriété de spécifications

Faisabilité : selon la complexité de la spécification, et des propriétés

Sécurité de protocoles cryptographiques : de nombreux outils.
Utilisés dans l'industrie : ProVerif et Tamarin.
Succès industriel : TLS 1.3 ! https://www.

Propriétés de systèmes distribués / concurrents : model checking.
Utilisé dans l'industrie : TLA+.

Et d'autres outils selon le cas d'usage !

Lien implémentation - spécification

Faisabilité : pas si simple (pour l'instant !).

Lien implémentation - spécification

Faisabilité : pas si simple (pour l'instant !).

Commence petit à petit à arriver dans l'industrie.
Entreprises spécialisées : Prove & Run, Galois, ...
Quelques projets de grandes entreprises : AWS, Microsoft, ...

Lien implémentation - spécification

Faisabilité : pas si simple (pour l'instant !).

Commence petit à petit à arriver dans l'industrie.
Entreprises spécialisées : Prove & Run, Galois, ...
Quelques projets de grandes entreprises : AWS, Microsoft, ...

Exemples de succès industriels :

- HACL* : implémentation de primitives cryptographiques vérifiées
- CompCert : compilateur C vérifié

Conclusion

Soyez humble,
— l'erreur est humaine.

Ne vous tirez pas une balle dans le pied,
— utilisez des langages memory-safe si possible.

Écrivez des spécifications,
— ce que vous voulez que le code fasse.

Vérifiez vos spécifications,
— utilisez des outils adaptés à votre cas d'usage.

Utilisez du code vérifié,
— vous vous éviterez bien des ennuis.

Feedback :