From CVEs to proof :
Make your USB device stack great again

SSTIC 2021

IRSN
INSTITUT
DE RADIOPROTECTION
ET DE SÛRETÉ NUCLÉAIRE

Ryad Benadjila, Cyril Deberge
Patricia Mouy, Philippe Thierry

ANSSI, IRSN

<prenom.nom@ssi.gouv.fr>
<cyril.deberge@irsn.fr>

Juin 2021

Protocole USB

- Protocole largement répandu, présent dans de très nombreux appareils
- Protocole complexe à implémenter :
  - champs de longueur variable
  - requêtes génériques
  - changements d'états provoqués par des évènements asynchrones

Protocole USB

- Protocole largement répandu, présent dans de très nombreux appareils
- Protocole complexe à implémenter :
  - champs de longueur variable
  - requêtes génériques
  - changements d'états provoqués par des évènements asynchrones
- **Une mine à CVEs !**

Quelques exemples :

- *Checkm8* : *jailbreak* de certains iPhones

Pourquoi s'intéresser à l'USB ?

Quelques exemples :

- *Checkm8* : *jailbreak* de certains iPhones

- *Fusée gelée* : *hack* des Nintendo Switchs

Pourquoi s'intéresser à l'USB ?

Quelques exemples :

- *Checkm8* : *jailbreak* de certains iPhones

- *Fusée gelée* : *hack* des Nintendo Switchs

- Vulnérabilité dans le micro-logiciel *STM32CubeMX*, présent dans de nombreux IoTs

POURQUOI S'INTÉRESSER À L'USB ?

Quelques exemples :

Points communs

Issues de RunTime Errors (RTEs)

Conduisant à une exécution de code arbitraire (RCE)

Difficiles ou impossibles à corriger (BootROM, IoT, etc.)

Critiques d'un point de vue sécurité

Notre objectif :

- Améliorer la sécurité d'une pile USB open-source

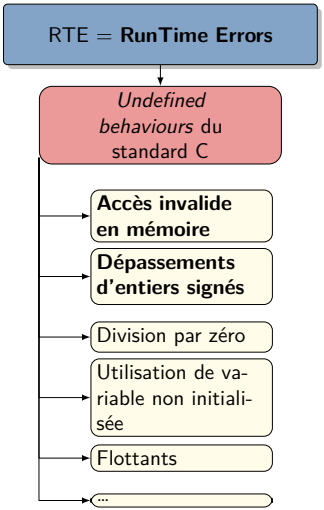
Notre objectif :

- Améliorer la sécurité d'une pile USB open-source
  - Pile USB implémentée dans WOOKEY

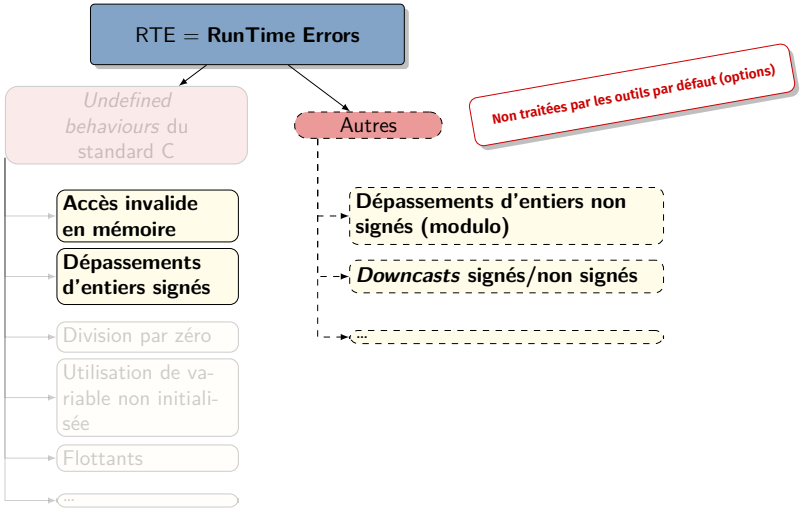
Notre objectif :

- Améliorer la sécurité d'une pile USB open-source
  - Pile USB implémentée dans WOOKEY

- Vérifier formellement l'absence de RTE dans cette pile USB
  - Étape nécessaire pour prouver l'absence de RCE



Classiquement traitées par les outils


RTE = RunTime Errors
Undefined behaviours du standard C
Autres
Non traitées par les outils par défaut (options)
Accès invalide en mémoire
Dépassements d'entiers signés
Division par zéro
Utilisation de variable non initialisée
Flottants
...
Dépassements d'entiers non signés (modulo)
Downcasts signés/non signés
...

Définition de la vérification formelle

- **Raisonnement mathématique** sur des programmes informatiques
- Démonstration de leur validité par rapport à une certaine propriété :
  - Propriétés fonctionnelles
  - Propriétés de sécurité (dont absence de RTE)

Définition de la vérification formelle

- **Raisonnement mathématique** sur des programmes informatiques
- Démonstration de leur validité par rapport à une certaine propriété :
  - Propriétés fonctionnelles
  - Propriétés de sécurité (dont absence de RTE)

Notions importantes

- **Complétude** : absence de faux positif (sous-approximation)
- **Correction** : absence de faux négatif (sur-approximation)

Plate-forme Frama-C (Framework for modular analysis of C programs)

- Développée par le CEA-LIST et l'Inria
- Open-source, modulaire et collaborative dédiée à l'analyse du langage C
- Réalise des analyses **statiques** et dynamiques

Plate-forme Frama-C (Framework for modular analysis of C programs)

- Développée par le CEA-LIST et l'Inria
- Open-source, modulaire et collaborative dédiée à l'analyse du langage C
- Réalise des analyses **statiques** et dynamiques

EVA

- **Preuve d'absence de RTE**
- Explore tous les chemins d'exécution dans le code analysé
- Outil **correct** : toutes les RTEs sont détectées, mais avec des **faux-positifs**

Plate-forme Frama-C (Framework for modular analysis of C programs)

- Développée par le CEA-LIST et l'Inria
- Open-source, modulaire et collaborative dédiée à l'analyse du langage C
- Réalise des analyses **statiques** et dynamiques

EVA

- **Preuve d'absence de RTE**
- Explore tous les chemins d'exécution dans le code analysé
- Outil **correct** : toutes les RTEs sont détectées, mais avec des **faux-positifs**

WP

- Permet de prouver des propriétés plus complexes
- Preuves fonctionnelles
- Outil correct, modulaire
- **Utilisé pour vérifier les faux positifs levés avec EVA** et la bonne terminaison des fonctions analysées

ACSL

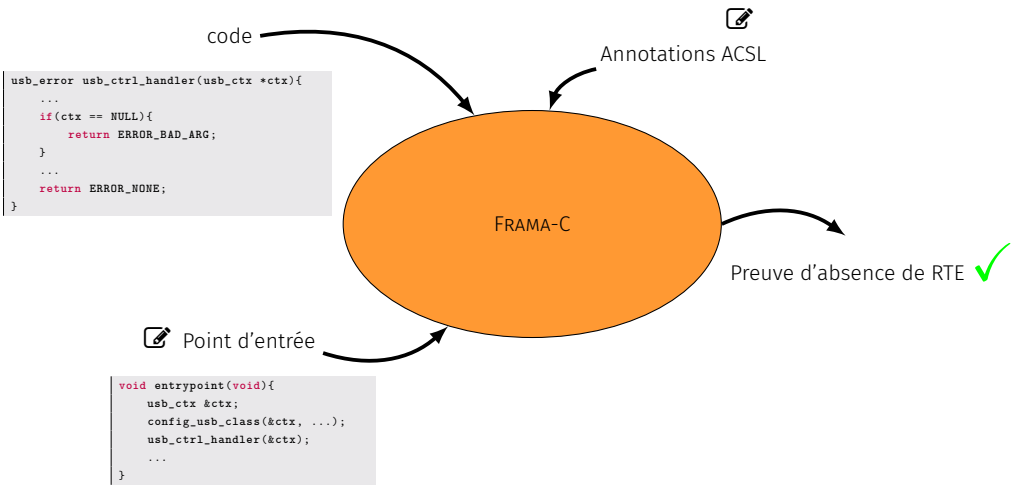
- Langage basé sur le C, avec des prédicats supplémentaires
- Permet de faire communiquer les différents plug-ins de Frama-C
- Annotations dans le code : automatiques ou manuelles (interaction utilisateur)

ACSL

- Langage basé sur le C, avec des prédicats supplémentaires
- Permet de faire communiquer les différents plug-ins de Frama-C
- Annotations dans le code : automatiques ou manuelles (interaction utilisateur)

```
/*@ requires \valid_read(pkt);
    ensures \result ≡ \old(pkt)->wValue >> 8;
    assigns \nothing;
  */
usbctrl_req_descriptor_type_t usbctrl_std_req_get_descriptor_type(usbctrl_setup_pkt_t const * const pkt)
{
  /*@ assert rte: mem_access: \valid_read(&pkt->wValue); */
  /*@ assert rte: shift: 0 ≤ (int)pkt->wValue; */
  usbctrl_req_descriptor_type_t val =
    (enum __anonenum_usbctrl_req_descriptor_type_t_95)((int)pkt->wValue >> 8);
  return val;
}
```

Méthodologie d'analyse mise en œuvre



Méthodologie d'analyse mise en œuvre

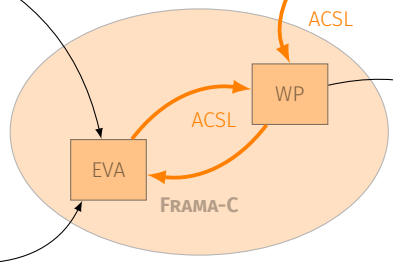
```
int i;
//@ loop invariant 0 <= i <= 10;
//@ loop assigns i;
//@ loop variant (10 - i);
for (i = 0; i < 10; i++) {
  ++i;
}
```

```
/* @ ...
@ behavior invalid_ctx:
@    assumes ctx == NULL;
@    ensures \result == ERROR_BAD_ARG;
@ ...
@ behavior ok:
@    assumes ctx != NULL;
@    ...
@    ensures \result == ERROR_NONE;
@ complete behaviors ;
@ disjoint behaviors ;
*/
```

code

```
usb_error usb_ctrl_handler(usb_ctx *ctx){
    ...
    if(ctx == NULL){
        return ERROR_BAD_ARG;
    }
    ...
    return ERROR_NONE;
}
```

Contrats de boucles et de fonctions



Preuve d'absence de RTE ✓

Point d'entrée

```
void entrypoint(void){
    usb_ctx &ctx;
    config_usb_class(&ctx, ...);
    usb_ctrl_handler(&ctx);
    ...
}
```

Travail réalisé sur le parser X.509

- Vérification formelle avec FRAMA-C de l'absence de RTE dans un parser de certificats X.509
- Présentation lors du SSTIC 2019
- `https://www.sstic.org/2019/presentation/journey-to-a-rte-free-x509-parser/`

Travail réalisé sur le parser X.509

- Vérification formelle avec FRAMA-C de l'absence de RTE dans un parser de certificats X.509
- Présentation lors du SSTIC 2019
- `https://www.sstic.org/2019/presentation/journey-to-a-rte-free-x509-parser/`

Un Retex pour l'utilisation de FRAMA-C

- Stratégie d'utilisation
- Paramétrages des plug-ins
- Annotation de code pour la preuve de l'absence de RTE

Travail réalisé sur le parser X.509

- Vérification formelle avec FRAMA-C de l'absence de RTE dans un parser de certificats X.509
- Présentation lors du SSTIC 2019
- `https://www.sstic.org/2019/presentation/journey-to-a-rte-free-x509-parser/`

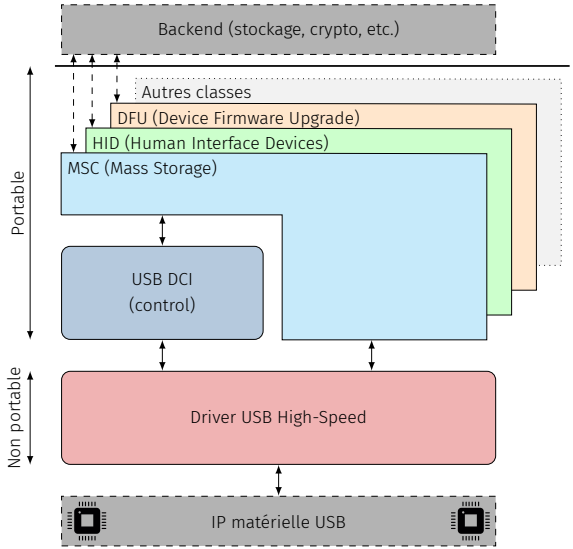
Un Retex pour l'utilisation de FRAMA-C

- Stratégie d'utilisation
- Paramétrages des plug-ins
- Annotation de code pour la preuve de l'absence de RTE

Néanmoins, challenge plus élevé pour une pile USB

- 12000 sloc pour la pile USB, contre 5000 pour le parser X.509
- Architecture plus complexe de la pile USB
- Évènements asynchrones dans la pile USB, contre un code purement séquentiel pour le parser X.509
- Adhérence avec du matériel pour la pile USB

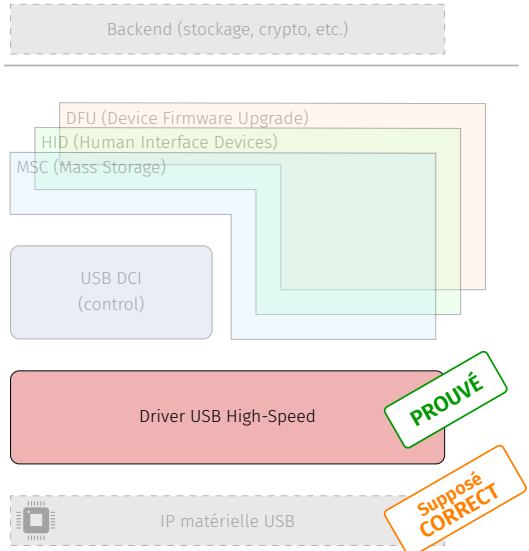
Architecture de la pile USB analysée



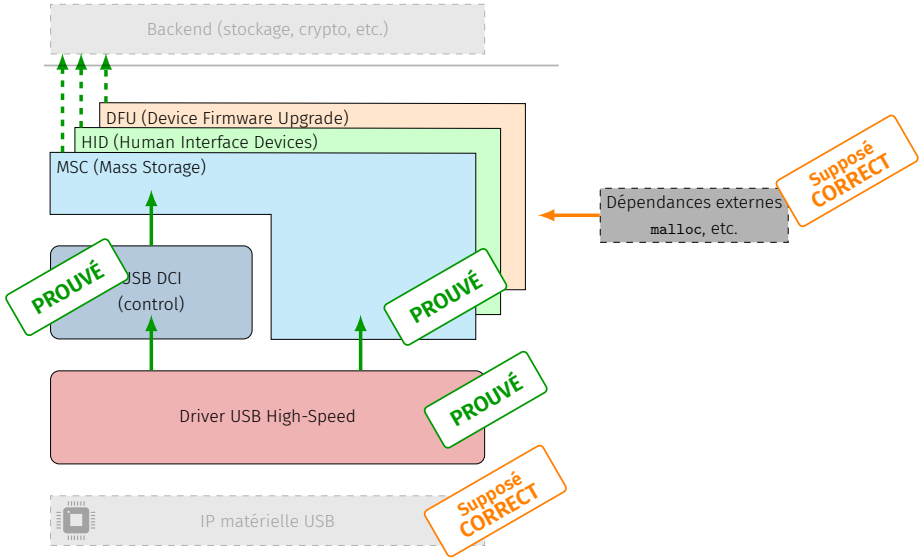
Caractéristiques

- Pile USB développée en langage C (12000 sloc)
- Code embarqué :
  - pas de code POSIX
  - pas de libc standard
  - API propriétaire
- Architecture portable sur différentes plateformes comprenant :
  - un driver USB (High-Speed ou Full-Speed)
  - une bibliothèque implémentant le plan contrôle USB 2.0
  - plusieurs classes USB (MSC, DFU, HID, CDC, etc.)

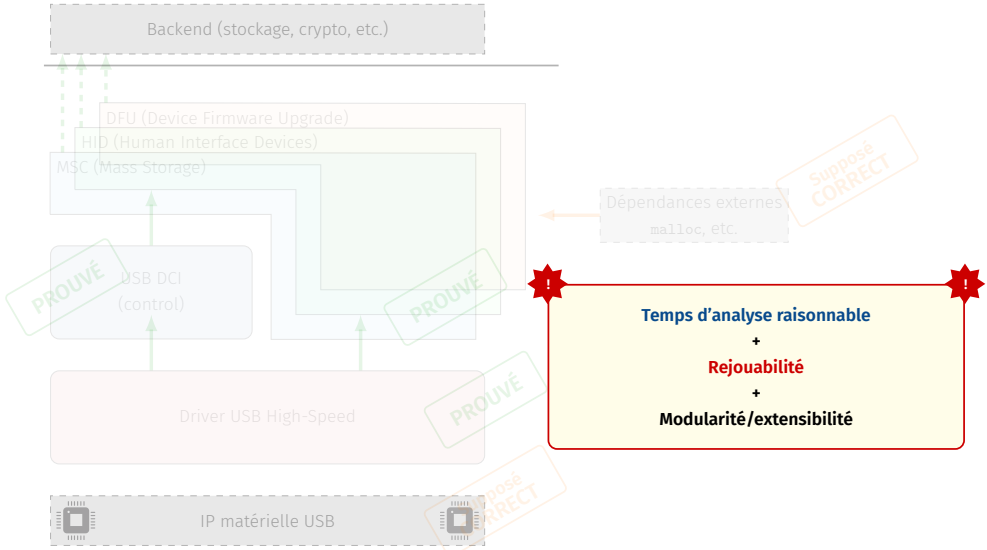
Approche par preuves modulaires



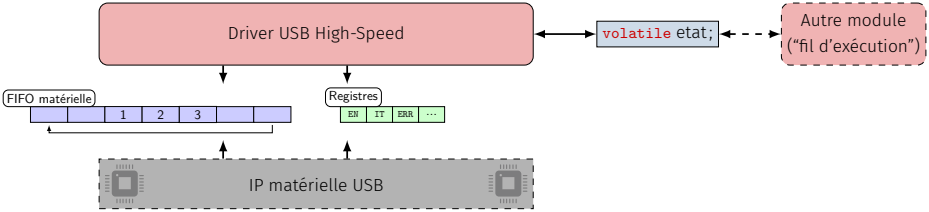
Approche par preuves modulaires



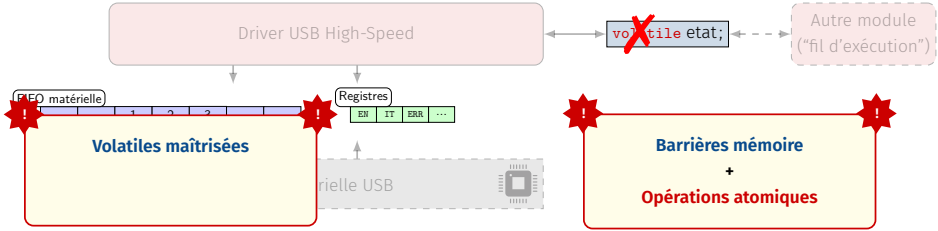
Approche par preuves modulaires



GESTION DES "VOLATILE"



GESTION DES "VOLATILE"



Problèmes de concurrence et réentrance à gérer

DES ÉVÈNEMENTS ASYNCHRONES

Fil d'exécution 1

```
/* Attente de l'evenement */
while (!scsi_is_ready_for_data_send()) {
    request_data_membarrier();
    continue;
}
```

Fil d'exécution 2

```
...
/* Prevenir l'autre fil d'execution */
atomic_write(ready_for_data_send, True);
request_data_membarrier();
...
```

Fil d'exécution 1

```
/* Attente de l'evenement */
while (!scsi_is_ready_for_data_send()) {
    request_data_membarrier();
    continue;
}
```

Séquentialisation

```
/* Sequentialisation */
#ifdef __FRAMAC__
if (!scsi_is_ready_for_data_send()) {
    scsi_data_sent();
}
#else
while (!scsi_is_ready_for_data_send()) {
    request_data_membarrier();
    continue;
}
#endif
```

Divergence entre le code analysé et le code exécuté

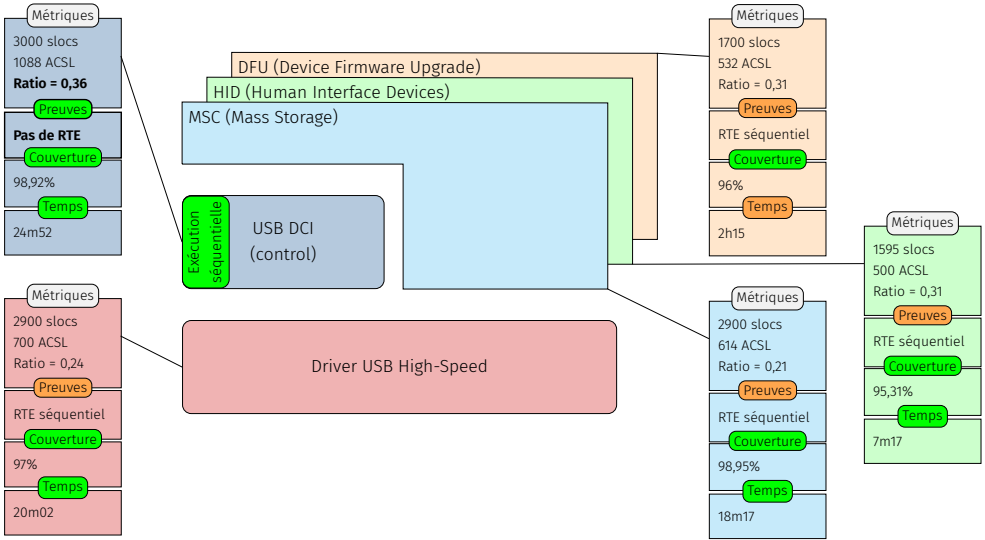
⇒

Parallélisme/concurrence/TOCTOU non couverts

Analysé par FRAMA-C ✓

Runtime : **non** analysé par FRAMA-C ✗

Résultats obtenus



Résultats obtenus

≈ 12000 sloc de C au total

≈ 1 ligne ACSL toutes les 4 lignes de C

≈ 1500 lignes d'entrypoints

GitHub

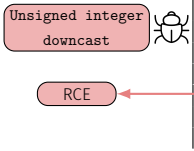
Intégration CI github Actions

Preuve d'absence de RTE :

- Dans un contexte séquentiel
  - RTEs asynchrones toujours possibles (concurrence, multi-threading)

Preuve d'absence de RTE :

- Dans un contexte séquentiel
  - RTEs asynchrones toujours possibles (concurrence, multi-threading)
- Nombreuses RTEs **corrigées**
  - **10** dans la bibliothèque USBctrl, **8** dans le driver par exemple (dont des RTEs exploitables)




```
mbed_error_t usbctrl_handle_class_requests(usbctrl_setup_pkt_t *pkt, usbctrl_context_t   *ctx)
{
  ...
  /* Get interface from the packet index */
  iface_idx = (((pkt->wIndex) & 0xff) - 1);
  ...
  /* Call our interface handler */
  usbctrl_ctx[ctxh].ifaces[iface_idx]();
  ...
}
```

Preuve d'absence de RTE :

- Dans un contexte séquentiel
  - RTEs asynchrones toujours possibles (concurrence, multi-threading)
- Nombreuses RTEs **corrigées**
  - **10** dans la bibliothèque USBctrl, **8** dans le driver par exemple (dont des RTEs exploitables)

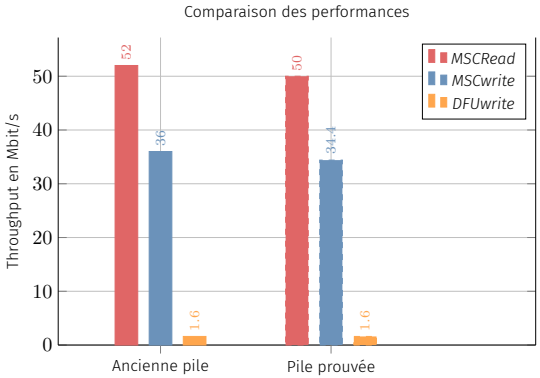
```
mbed_error_t usbctrl_handle_class_requests(usbctrl_setup_pkt_t *pkt, usbctrl_context_t   *ctx)
{
```

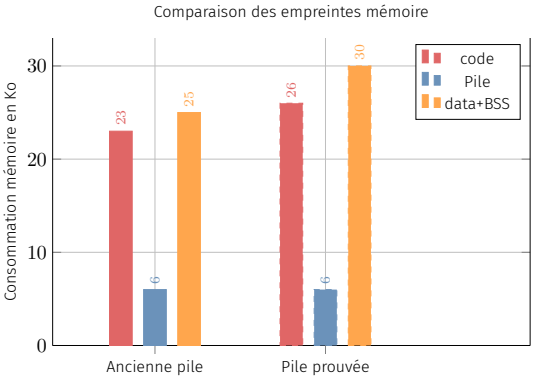
Gain de securité

Vulnérabilités *Checkm8*, *Fusée gelée* **et** *STMCubeMX* **attrapées**

Correction d'erreurs de programmation (`goto` invalides, code mort)

Au-delà de l'absence de RTE, des preuves fonctionnelles


Comparaison des performances
Throughput en Mbit/s
0
10
20
30
40
50
52
36
1.6
50
34.4
1.6
MSCRead
MSCwrite
DFUwrite
Ancienne pile
Pile prouvée


Comparaison des empreintes mémoire
Consommation mémoire en Ko
0
10
20
30
23
6
25
26
6
30
code
Pile
data+BSS
Ancienne pile
Pile prouvée

Un gain important pour la sécurité

- Preuve de l'absence de RTE dans un contexte séquentiel :
  - Nombreuses RTEs exploitables corrigées durant le développement de la pile USB
- Validation d'une méthodologie d'analyse par composition avec Frama-C et des plug-ins open-source
- Impact quasi-nul sur les performances de la pile USB
- Au-délà de l'absence de RTE :
  - Correction d'erreurs de programmation
  - Preuve fonctionnelle d'une partie de la pile USB

Work in progress

- Gérer les RTEs dans un contexte asynchrone
- Comparer les résultats obtenus avec d'autres outils d'analyses correctes
- Étendre la vérification des propriétés fonctionnelles

From CVEs to proof :
Make your USB device stack great again

SSTIC 2021

SSTIC SYMPOSIUM
SUR LA SÉCURITÉ DES TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

Question ?