Attaques en faute et sécurisation de code

Karine Heydemann

Colloque Architecture : Hier, Aujourd'hui, Demain

Journée en l'honneur de M. Auguin, D. Etiemble et B. Goossens.

LIP6 cnrs

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

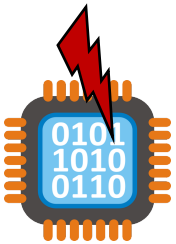
Contexte

Systèmes embarqués, objets connectés

Menaces variées exploitant

Failles humaines

Caractéristiques matérielles



Vulnérabilités logicielles

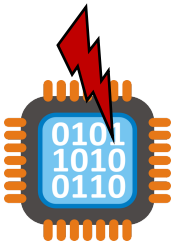
Contexte

Systèmes embarqués, objets connectés

Menaces variées exploitant

Caractéristiques matérielles

Attaques physiques

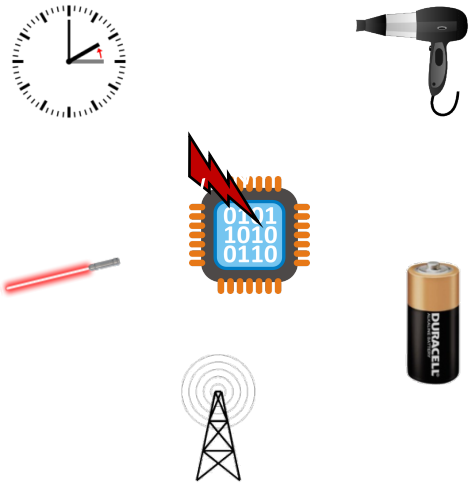


par perturbation ou « en faute »

Attaque en faute

Moyens d'injection

- Depuis 20 ans moyens physiques
  - faisceau laser, impulsion électromagnétique, glitch d'horloge ou de tension [El Bar et al., 2006]
- Récemment moyens purement logiciels
  - *row hammer* [Kim et al., 2014], *clock skrew* [Tang et al., 2017]



Exploitation

- Récupération de clé cryptographique [Dehbaoui et al., 2013] [Kumar et al., FDTC 2017]
- Contournement de boot sécurisé [Timmers et al., FDTC 2016]
- Prise de contrôle d'un système [Timmers et al., FDTC 2017]
- Escalade de privilèges [Vasselle et al. FDTC 2017]

Protections contre les attaques en faute

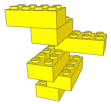
Contremesures matérielles [El Bar et al., 2006]

- Capteur de lumière, détecteur de glitch d'horloge ou de tension [Zussa et al., 2014]
- Redondance matérielle [Karaklajic et al, 2013]
- Code correcteur d'erreur (registres, mémoire)

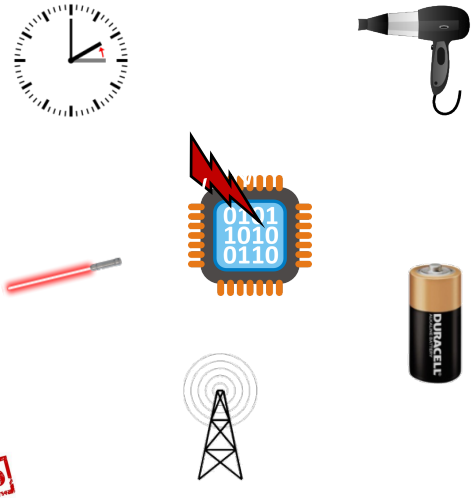
Trop coûteux pour certains systèmes et pas de garantie complète

Contremesures logicielles [Verbauhede, 2011] [Rauzy et al., 2015]

- Redondance au niveau des fonctions
- Protections dédiées à des algorithmes (e.g. RSA)
- Protections ad-hoc conçues par des experts



En pratique combinaison des deux types (*secure elements*)



Problématique générale

Sécurisation de code

- **Encore très manuelle**
  - Haute expertise requise
  - Fastidieux et source d'erreur
  - Peu de réutilisation possible, coût global élevé
- **Automatisation et capitalisation des protections recherchées**
  - Réduction des coûts, disponible à des non-experts
  - Adaptable aux spécificités d'un produit
  - Compromis sécurité performance
- **Besoin de protections génériques**
  - Non dédiées à une classe d'algorithme (crypto)

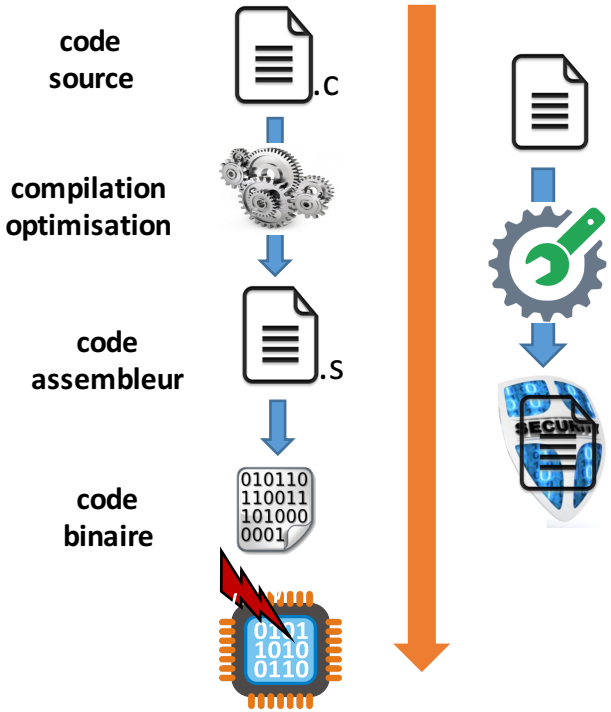
Problématique générale

Sécurisation de code

- Automatisation et capitalisation des protections recherchées
- Besoin de protections génériques
- **À quel niveau de représentation du code ?**

Protection logicielle contre les attaques en faute

Sécurisation de code : à quel niveau de représentation du code ?

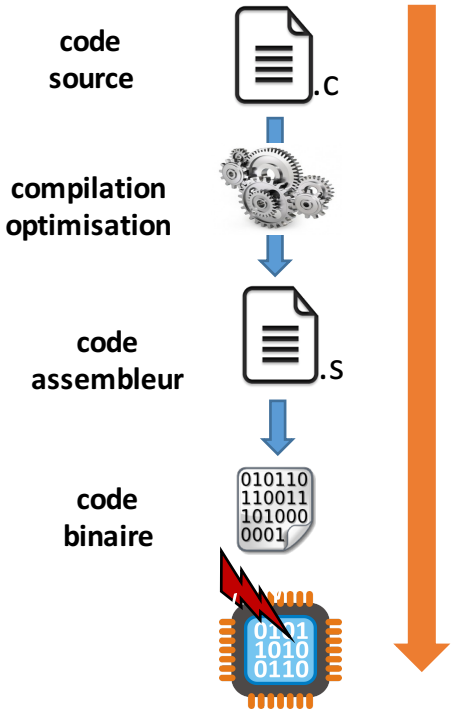


- **Source**
  - Revue de code aisée, portabilité, indépendance aux outils
  - Compilation et optimisation
- **Compilation**
  - Adaptabilité et contrôle sur les optimisations possibles
  - Compilateur propriétaire ou certifié
- **Assembleur**
  - Codes écrits directement ainsi, information bas niveau disponible
  - Lié à un ISA, potentiel manque d'information de haut niveau
- **Binaire**
  - Code "attaqué", vue globale, code des bibliothèques
  - Manque d'information sémantique, code à reconstruire

➜ **Besoins variés**

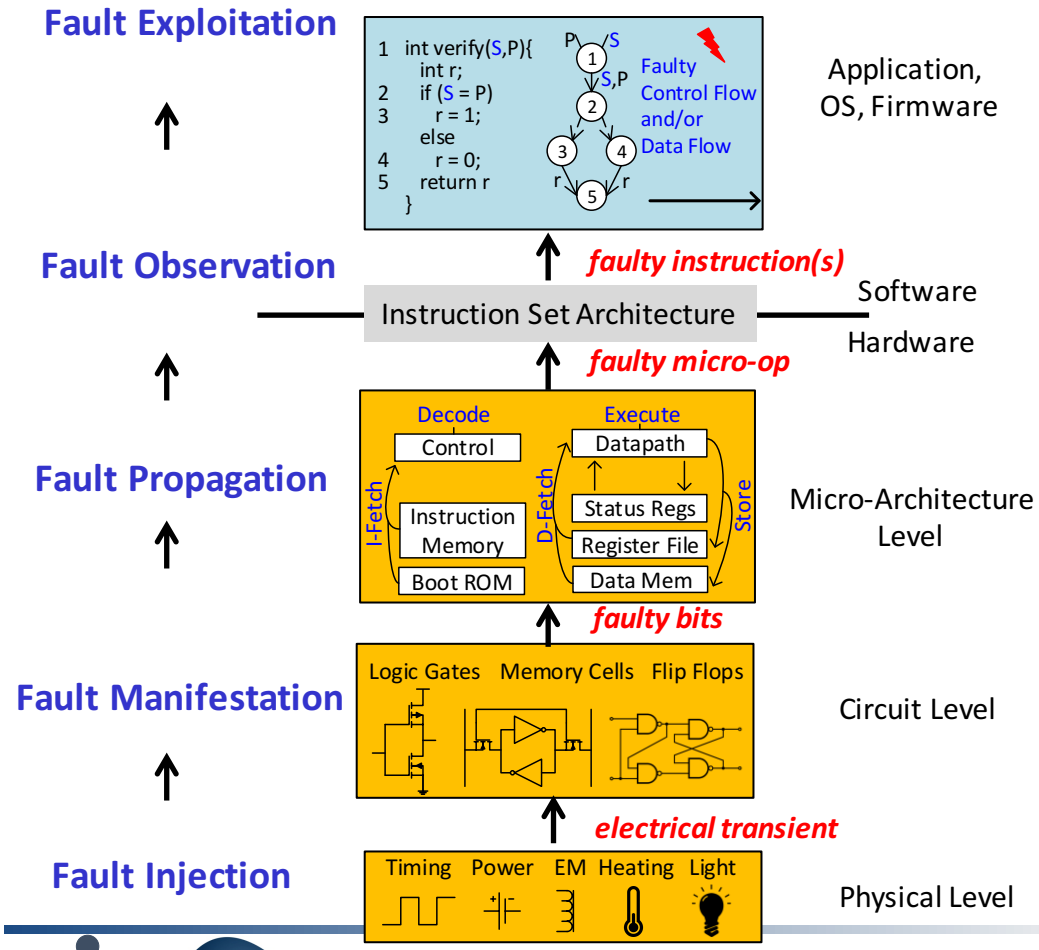
Problématique générale

Sécurisation de code



- Automatisation et capitalisation des protections recherchées
- Besoin de protections génériques
- Besoin à differents niveaux de représentation du code
- **Se protéger contre quel effet exactement ?**

Effets des injections de fautes



[Yuce et al., JHSS 2018]

- **Exploitation d'une faute**
  - Vue macroscopique d'une attaque en faute
  - Point de vue de l'attaquant
- **Compréhension des effets des injections de faute**
  - Caractérisation des effets observables possibles
  - Point de vue du designer
  - Nécessaire pour la sécurisation

Caractérisation des fautes au niveau logiciel

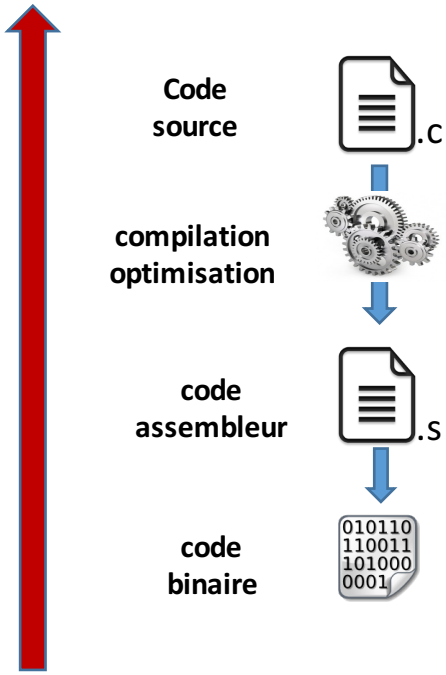
Modélisation des fautes

Caractérisation des fautes possibles **au niveau logiciel**

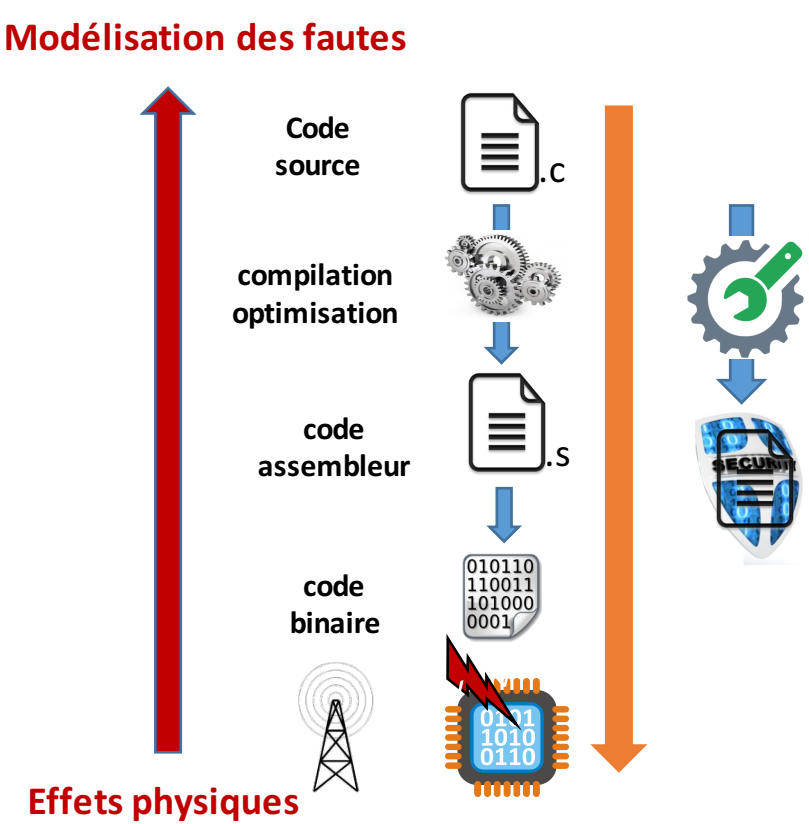
- Nécessaire pour concevoir et déployer des protections

Modèles de faute

- **Représentation simplifiée et abstraite** des effets d'une faute physique au niveau logiciel
- Représentation **à un niveau de code / au niveau de sécurisation** : binaire, assembleur, IR, code source



Problématique générale



Sécurisation de code

- Besoin d'automatisation et capitalisation des protections
- Besoin de protections génériques
- Besoin à differents niveaux de représentation du code
- **Besoin de modèles de faute, adaptés au niveau de sécurisation choisi**

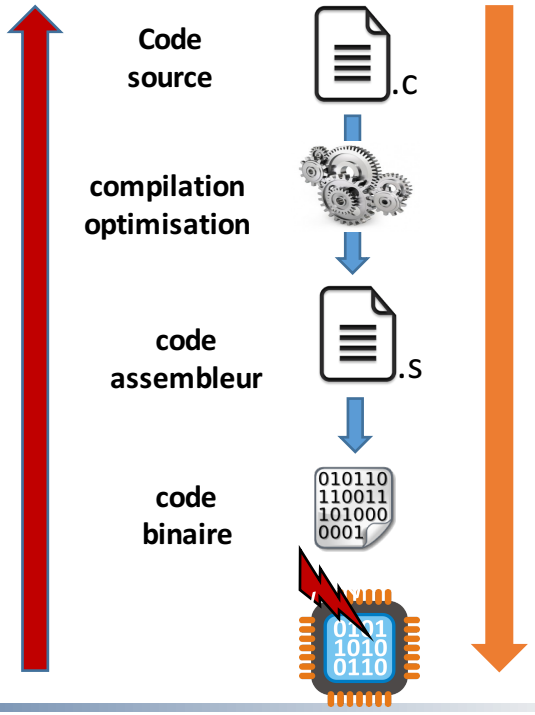
Panorama général des travaux réalisés

Modélisation des fautes

À partir de modèles de faute assembleur

À partir d'expérimentations

Effets physiques



Sécurisation de code

Protection du flot de contrôle

Protection des boucles

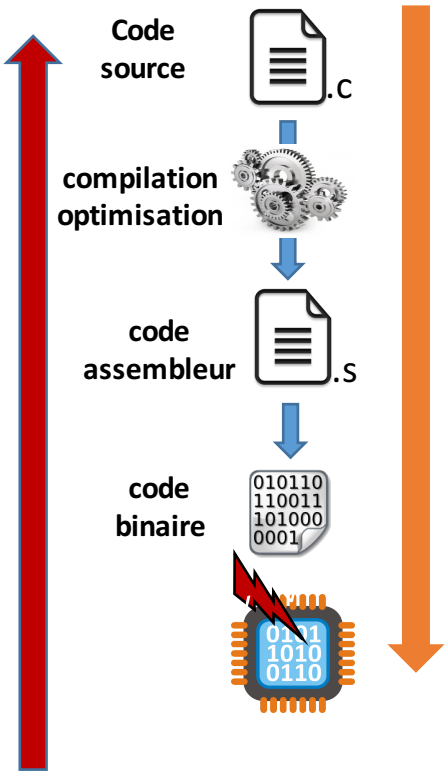
Protection contre un saut d'instruction

Travaux existants

Modélisation des fautes

[Kelly et al., 2017]
[Yuce et al., 2017]
[Timmers et al., 2016]
[Dureuil et al., 2015]
[Rivière et al., 2015]
[Balash et al., 2011]
[Verbauwhede et al., 2011]
[El Bar et al., 2006]

Effets physiques



Sécurisation de code : principe et/ou automatisation

- Source : [Goloubeva et al., 2005] [Akkar et al., 2003]
- Représentation bas-niveau : [Reis et al., 2005] [Barry et al., 2016]
- Assembleur : [Barenghi et al., 2010]
- Binaire : [De Keulenaer et al., 2016]

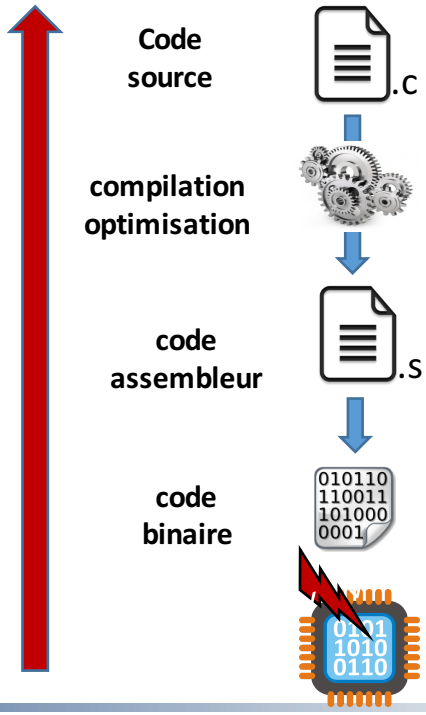
Panorama de la suite de l'exposé

Modélisation des fautes

Sécurisation de code

2- À partir de modèles de faute assembleur

1 - À partir d'expérimentations



Effets physiques

3 - Protection du flot de contrôle

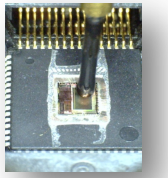
4 - Protection des boucles

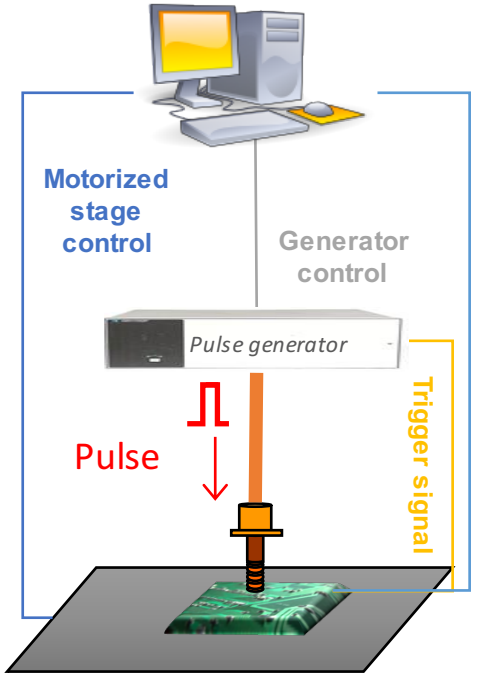
Protection contre un saut d'instruction

Modélisation de fautes au niveau assembleur

Étude des perturbations par injection d'impulsion électromagnétique

- Circuits standards non protégés, technique moins étudiée [AAPS CC 2013]
- Difficultés
  - **Pas de méthodologie** pour caractériser les fautes possibles (attaque en boite grise)
  - **Espace gigantesque des paramètres :** code et processeur cibles, paramètres du moyen d'injection





Approche classique de caractérisation en 3 étapes

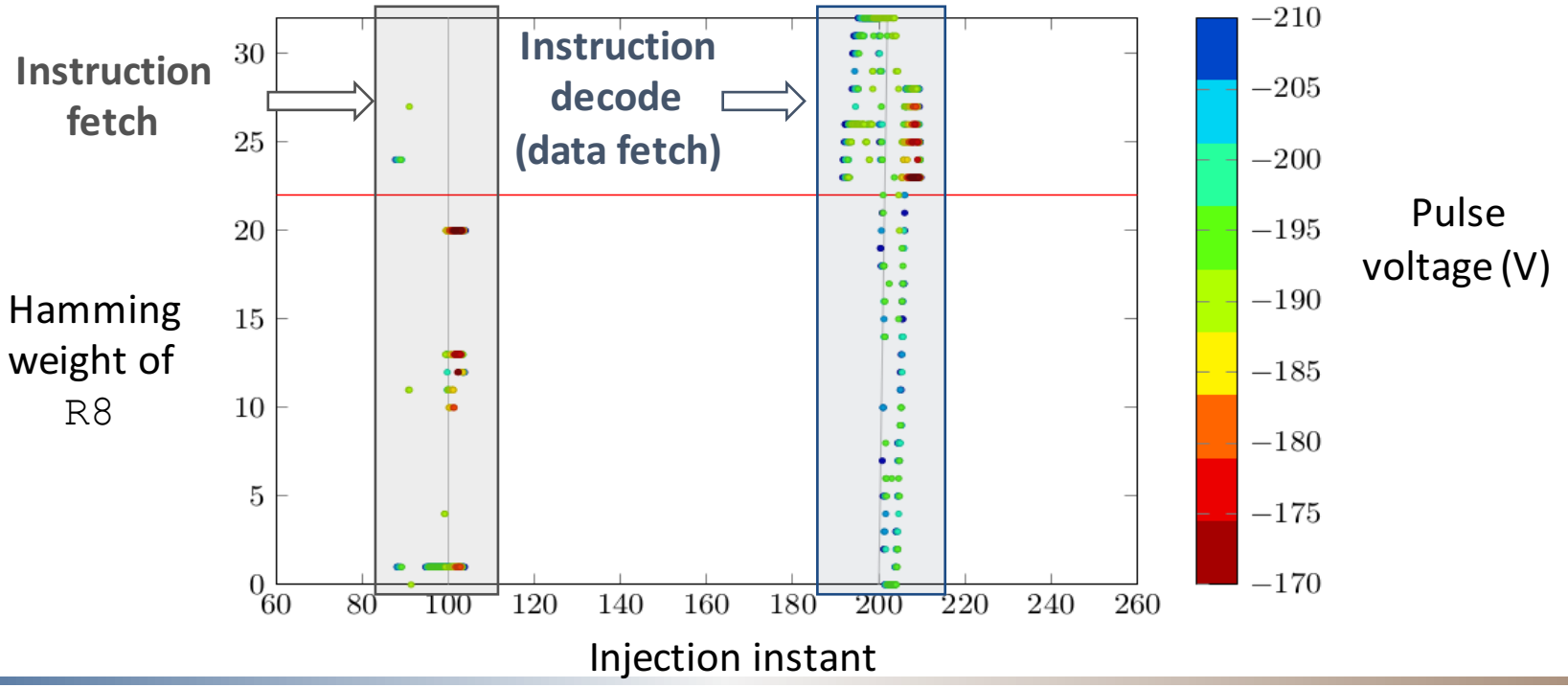
1. **Scan du composant** : recherche de zone avec observation de sorties fautées
2. **Sélection d'une configuration** avec forte probabilité d'observation de sortie fautée
3. **Élaboration d'un modèle de faute** pour la zone sélectionnée

Ecole Nationale Supérieure des Mines SAINT-ETIENNE

cea

Elaboration d'un modèle de faute

- **Composant :** carte STM32 avec un processeur ARM Cortex-M3
- **Instruction visée** : chargement depuis la mémoire (LOAD) de la valeur 0x12345678 dans R8
- **Cartographie temporelle et variation de l'intensité de l'impulsion**



Elaboration de modèle de fautes au niveau assembleur

Confrontation expériences physiques avec simulation

- Codes cibles : suite d'instructions NOP, chargement mémoire isolée, somme d'éléments tab[i] valant 2^i
- Simulation d'injection de faute
  - Faute simulée : **remplacement d'instruction**
  - Lorsqu'aucun remplacement explique une sortie → **corruption du chargement / flot de données**

Deux modèles de faute mis en évidence

- Faute transitoire sur les instructions
  - **Remplacement d'instruction**
  - **Saut d'instruction** (skip)
- Faute transitoire sur les données
  - **Corruption de données lues** depuis la mémoire Flash

Modèle de faute

inst → new_inst

inst → NOP (occurrence 25%)

ldr r4, [@flash] → r4 ← bad_value

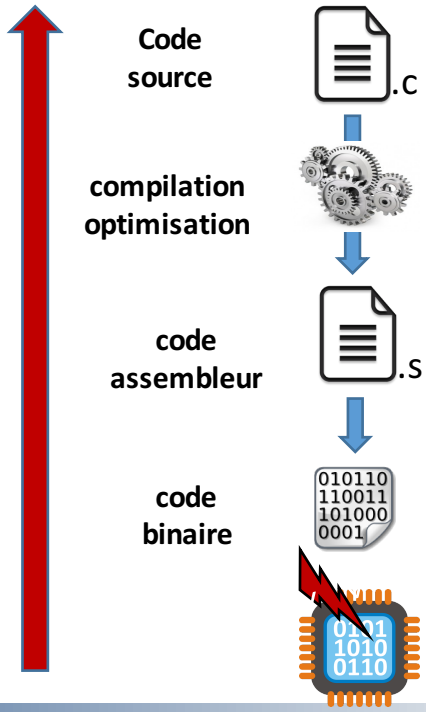
Panorama général de la suite

Modélisation des fautes

Sécurisation de code

2- À partir de modèles de faute assembleur

1 - À partir d'expérimentations



3 - Protection du flot de contrôle

4 - Protection des boucles

Protection contre un saut d'instruction

Effets physiques

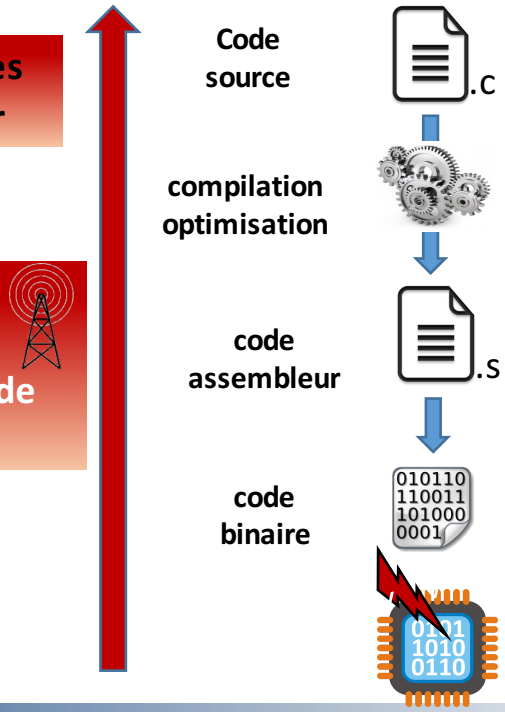
Panorama général de la suite

Modélisation des fautes

Sécurisation de code

2- À partir de modèles de faute assembleur

Saut d'instructions, remplacement d'instruction, lecture de données fautées



Effets physiques

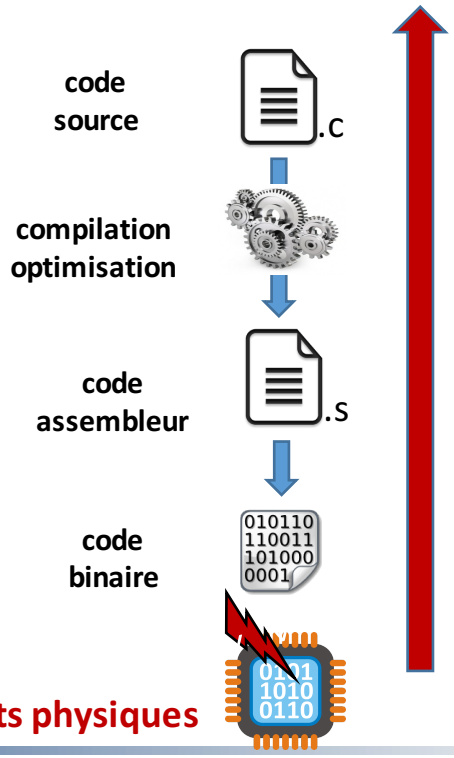
3 - Protection du flot de contrôle

4 - Protection des boucles

Protection contre un saut d'instruction

Modélisation de fautes au niveau source

Modélisation des fautes



Effets physiques

- Niveau code source

?

- Niveau assembleur
  - Saut d'instruction(s) skip
  - Remplacement(s) d'instruction
  - Corruption d'une donnée chargée
  - Corruption(s) de registre(s)

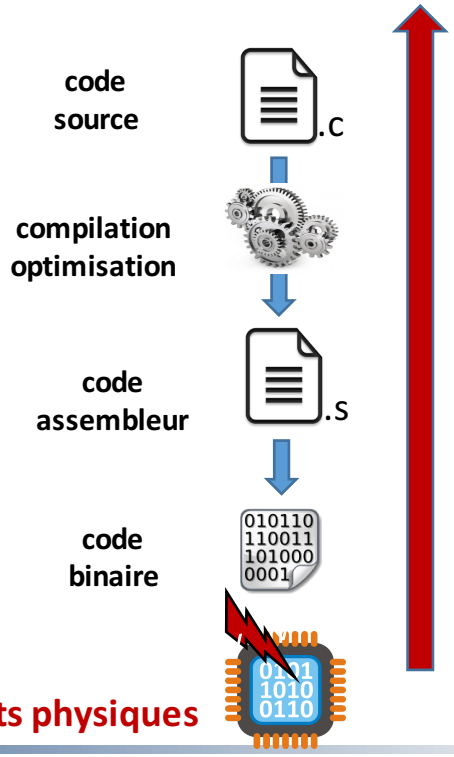
[Kelly et al., 2017]
[Yuce et al., 2017]
[Timmers et al., 2016]
[Dureuil et al., 2015]
[Rivière et al., 2015]
[Moro et al, 2013]
[Balash et al., 2011]
[Verbauwhede et al., 2011]
[El Bar et al., 2006]

Modélisation de fautes au niveau source

Modélisation des fautes

[Berthomé et al, 2010]
[Berthomé et al, 2012]

[Kelly et al., 2017]
[Yuce et al., 2017]
[Timmers et al., 2016]
[Dureuil et al., 2015]
[Rivière et al., 2015]
[Moro et al, 2013]
[Balash et al., 2011]
[Verbauwhede et al., 2011]
[El Bar et al., 2006]



Effets physiques

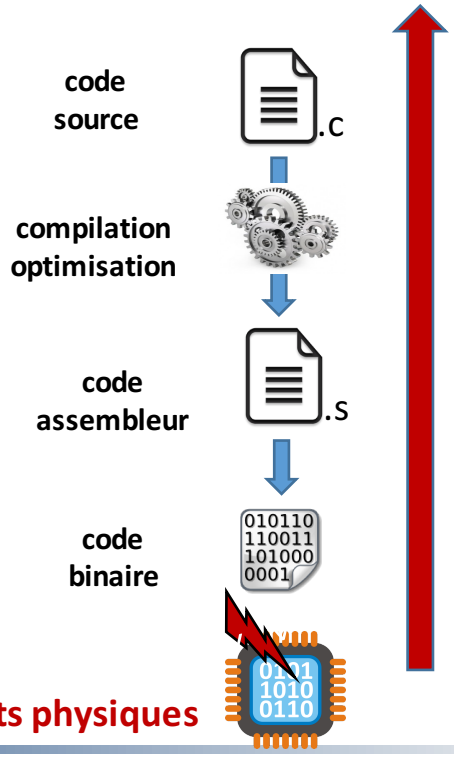
- Niveau code source
  - Corruption du flot de contrôle
  - Corruption de variables
  - Combinaison
- Niveau assembleur
  - Saut d'instruction(s) skip
  - Remplacement(s) d'instruction
  - Corruption d'une donnée chargée
  - Corruption(s) de registre(s)

Modélisation de fautes au niveau source

Modélisation des fautes

[Berthomé et al, 2010]
[Berthomé et al, 2012]

[Kelly et al., 2017]
[Yuce et al., 2017]
[Timmers et al., 2016]
[Dureuil et al., 2015]
[Rivière et al., 2015]
[Moro et al, 2013]
[Balash et al., 2011]
[Verbauwhede et al., 2011]
[El Bar et al., 2006]



Effets physiques

- Niveau code source
  - Corruption du flot de contrôle
    - Attaque par saut : **`goto label;`**
  - Corruption de variables
    - Attaque de variable : **`var = attack();`**
  - Combinaison
- Niveau assembleur
  - Saut d'instruction(s) skip
  - Remplacement(s) d'instruction
  - Corruption d'une donnée chargée
  - Corruption(s) de registre(s)

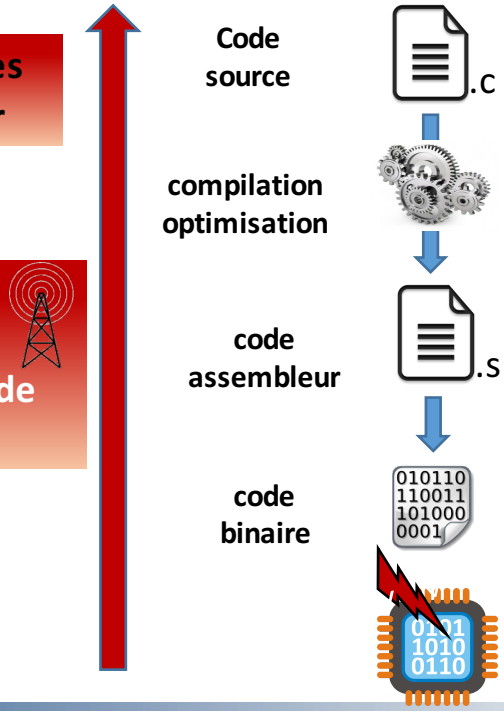
Panorama général de la suite

Modélisation des fautes

Sécurisation de code

2- À partir de modèles de faute assembleur

Saut d'instructions, remplacement d'instruction, lecture de données fautées



Effets physiques

3 - Protection du flot de contrôle

4 - Protection des boucles

Protection contre un saut d'instruction

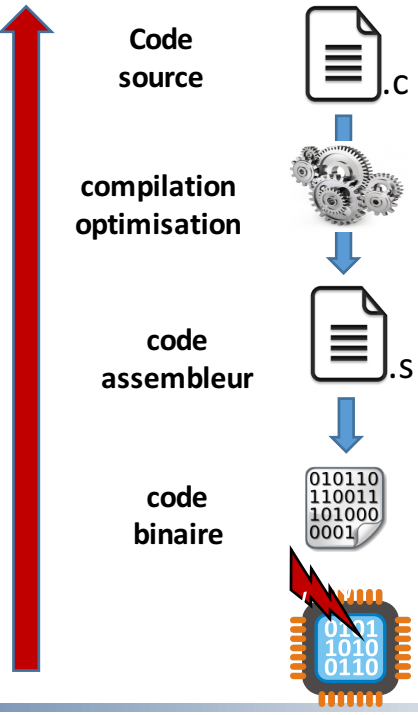
Panorama général de la suite

Modélisation des fautes

Attaque par saut, attaque de variable ou combinaison

Saut d'instructions, remplacement d'instruction, lecture de données fautées

Effets physiques



Sécurisation de code

3 - Protection du flot de contrôle

4 - Protection des boucles

Protection contre un saut d'instruction

Schéma de protection du code source contre la corruption du flot de contrôle

Principe

- Insertion de code de protection pour vérifier que
  - **toutes les lignes** ont été exécutées,
  - dans le **bon ordre,** et
  - en **accord avec le contexte d'exécution**

Modèle de faute

- Attaque par saut

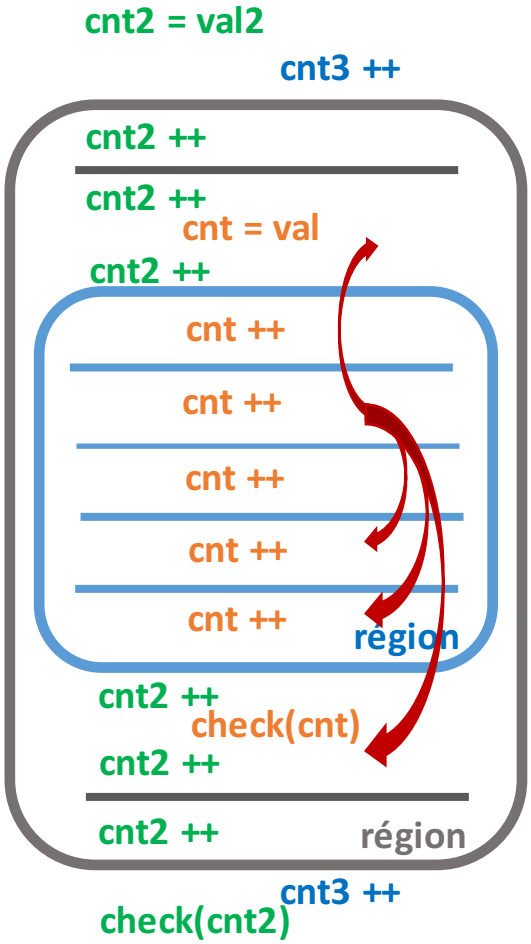
Schéma de protection de code source contre la corruption du flot de contrôle

Réalisation

- **Protection par régions** : séquentielles, constructions if-then-else, switch, boucles avec et sans rupture, fonctions et appels de fonction
- **Code de protection** reposant sur des **compteurs et des variables** pour capturer les conditions influençant le flot de contrôle
- **Imbrication et recouvrement** des protections des régions pour garantir la détection de faute

Vérification formelle du principe de sécurisation

- Aide à la conception et validation du principe



Évaluation du schéma de protection de code C

Simulation exhaustive des sauts intra-procéduraux

- **Au niveau C : 100% des attaques** par saut d'au moins 2 lignes détectées
  - Par construction, les sauts d'une ligne non détectés (même si nombre réduit)
  - Besoin de coupler l'approche avec de la duplication de ligne
- **Au niveau assembleur** : seulement **50 à 75% des attaques** par saut sont **détectées**
  - Sauts intra-lignes non détectés
  - Pas de correspondance directe entre les sauts haut et bas niveau

Impact en taille et performance de code important : jusqu'à un facteur 3

- Compromis entre le contenu d'une ligne et l'impact

Évaluation de la robustesse : étude de cas

Vérification de PIN [Benchmarks FISSC, Dureuil et al 2016]

- 2 fonctions étudiées
  - Revue de code
  - Analyse de vulnérabilité par attaque par saut simulée au niveau C

Réécriture du code source

- Une fonction ne comporte plus de vulnérabilité
- Aucune attaque dangereuse (attaque par saut au niveau assembleur)

Duplication des lignes sensibles de la deuxième fonction

- **100% des attaques par sauts** détectées (niveau assembleur)

Sécurisation partielle

- performance + 60 % (versus + 317%)
- taille de code + 40,5 % (versus + 118%)

Bilan sécurisation du code source

Avantages

- Régions de code et variables sensibles connues, identification automatique possible
- Revue de code aisée (mais longue)
- Indépendante de l'architecture cible

Inconvénients

- **Pas de correspondance** entre les **modèles de faute** haut et bas niveau
- **Compilation post-sécurisation :**
  - **Sans optimisation** (niveau zéro)
  - Essai-erreur de technique de codage spécifique possible :

    long et expertise nécessaire, dépendant du compilateur

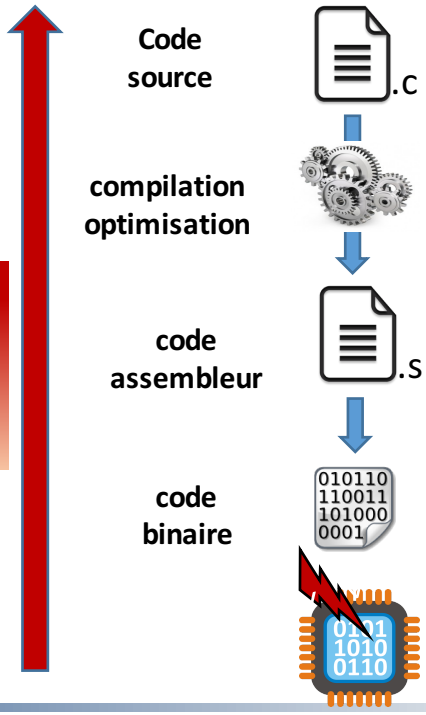
Panorama général de la suite

Modélisation des fautes

Attaques par saut, attaque de variable ou combinaison

Saut d'instructions, remplacement d'instruction, lecture de données fautées

Effets physiques



Sécurisation de code

Protection du flot de contrôle

4 - Protection des boucles

Protection contre un saut d'instruction

Schéma de protection des boucles à la compilation

J. Proy

Thèse en cours

Motivation

- Plusieurs attaques reposent sur la **corruption du nombre d'itérations**
  - Via dépassement de tampon (*buffer overflow*) [Nashimoto et al. 2017]
  - Cryptanalyse possible [Dehbaoui et al. 2013, Espitau et al. 2016]
  - Code d'authentification [Dureuil et al. 2016]

Modèles de faute compatibles « certification »

- Un « saut d'instruction » ou une « corruption de registre » lors l'exécution d'une boucle
- Raison : double fautes à deux moments bien différents de l'exécution pour arriver à un « gain »

```
void aes_addRoundKey_cpy(
      uint8_t *buf, uint8_t *
      key, uint8_t *cpk)
{
  register uint8_t i = 16 ;

  while (i--)
  {
    buf[i] ^= key[i] ;
    cpk[i] = key[i] ;
    cpk[16+i] = key[16+i] ;
  }

}
```

Schéma de protection des boucles à la compilation

Objectif

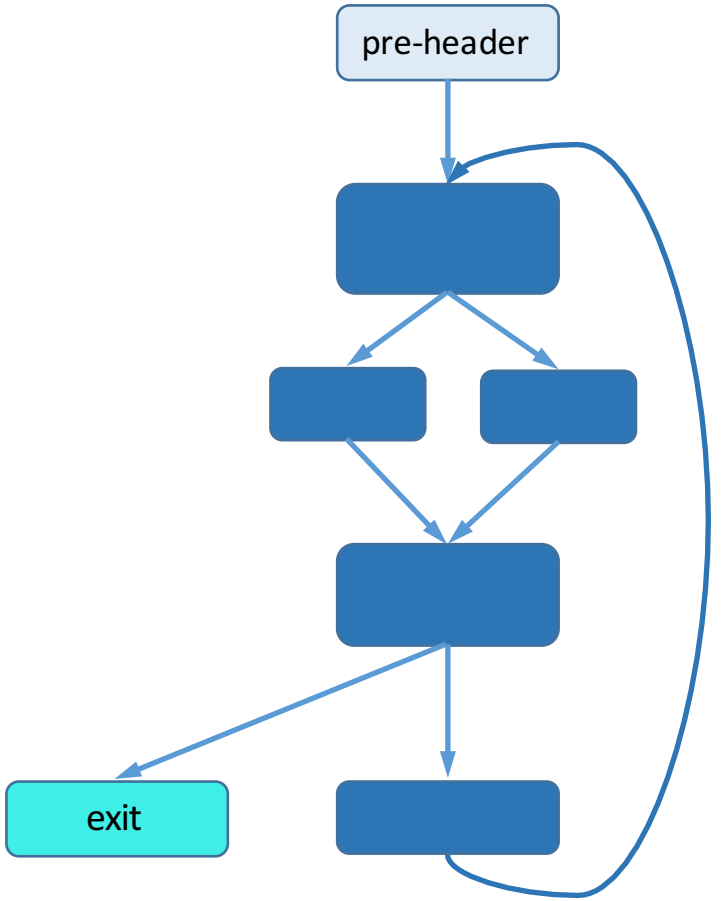
- Garantir le bon nombre d'itérations et la bonne sortie, ou détecter une attaque

Principe de protection

- Pour chaque sortie de boucle possible, vérification de la direction prise

Réalisation

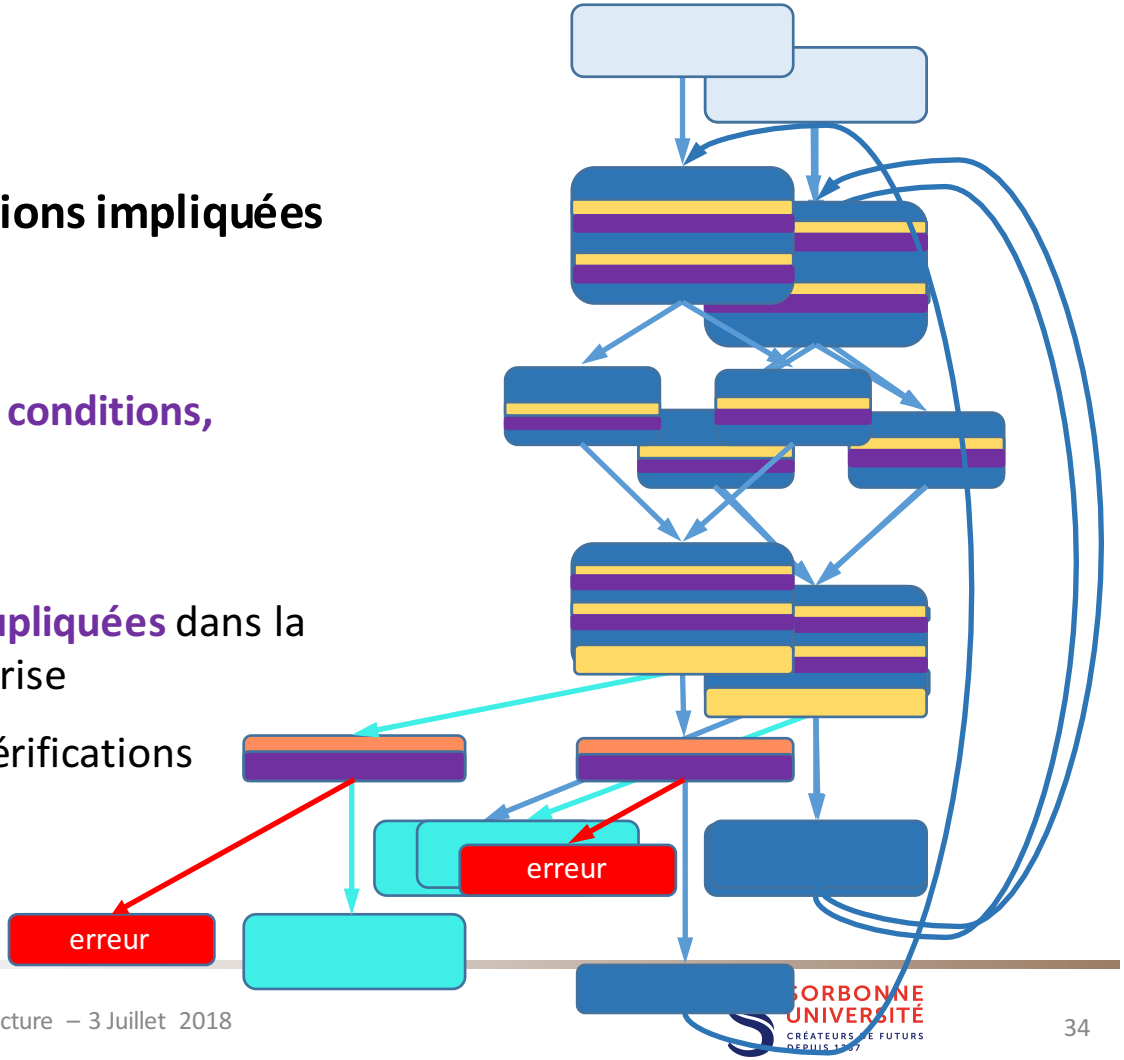
- Duplication des instructions impliquées dans le calcul des conditions des tests de sortie
- Ajout d'une vérification de la direction prise après les tests de sortie



Algorithme pour la protection d'une boucle

Pour une boucle et chacune de ses sorties

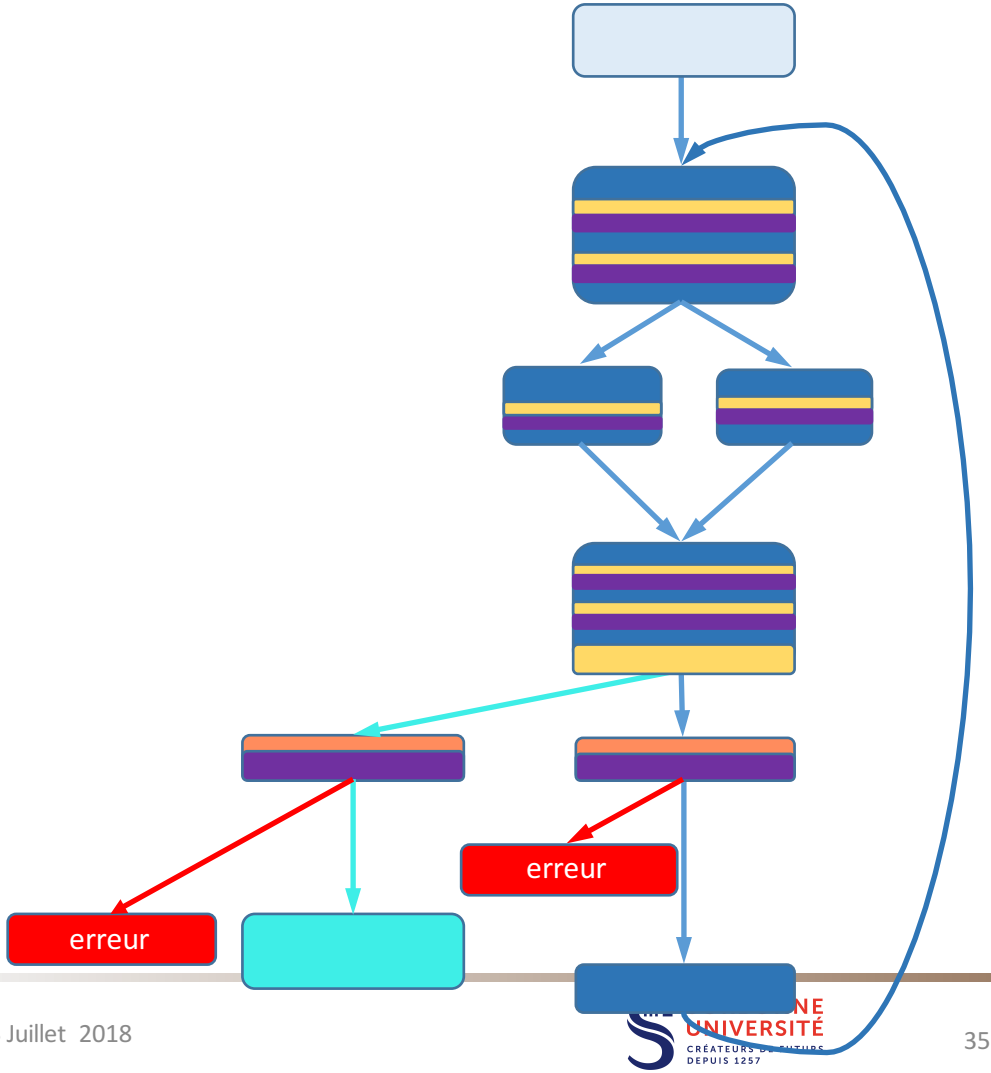
- Détermination par analyse arrière des **instructions impliquées** dans une **condition de sortie**
- Duplication d'instructions
  - **Création d'une deuxième chaîne de calcul des conditions, indépendante de l'originale**
- Ajout de blocs
  - **de vérification avec les conditions de sortie dupliquées** dans la boucle et à sa sortie pour vérifier la direction prise
  - **de gestion d'erreur** en cas d'échec d'une des vérifications



Algorithme pour la protection d'une boucle

Est-ce suffisant ?

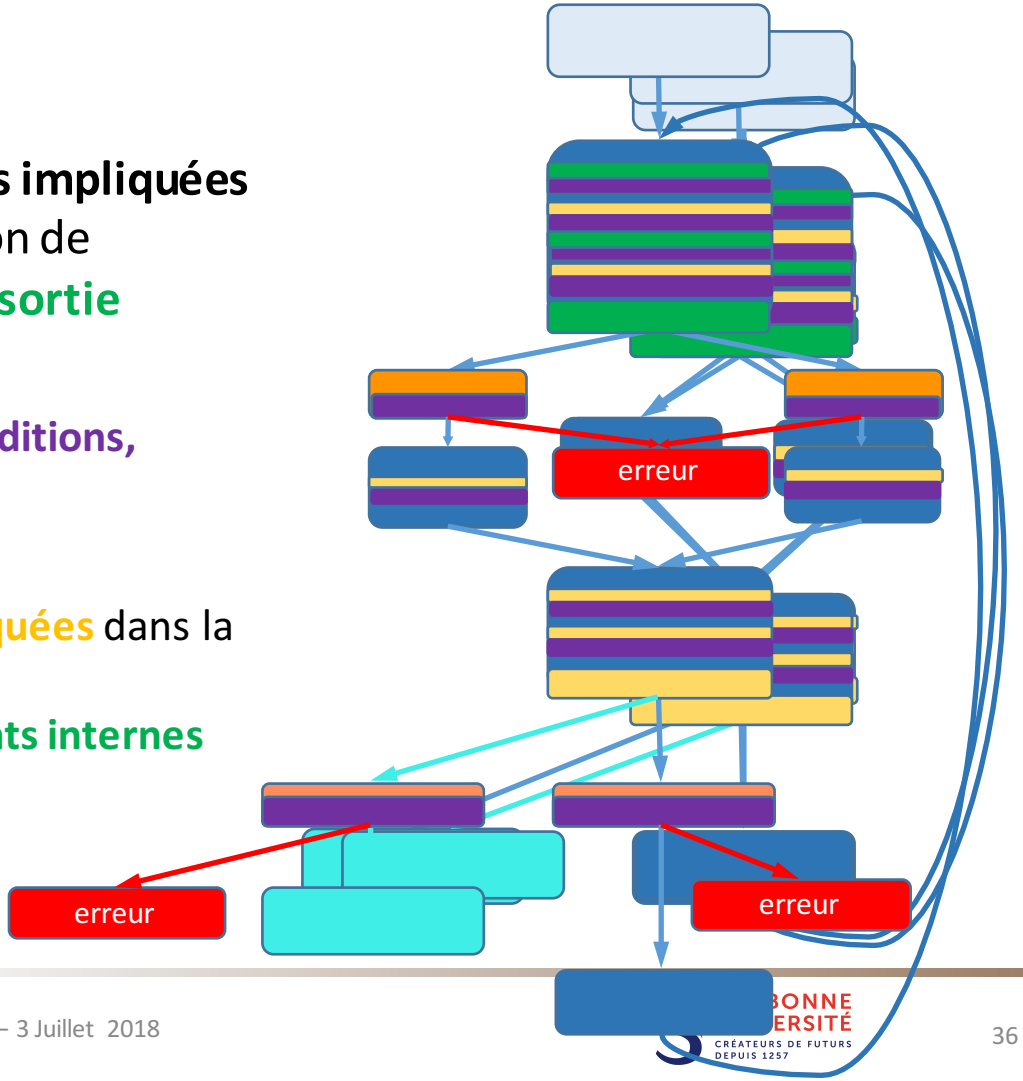
- Il est nécessaire de protéger le flot de contrôle interne de la boucle qui peut influencer la valeur d'une condition de sortie



Algorithme pour la protection d'une boucle

Pour une boucle et chacune de ses sorties

- Détermination par analyse arrière des **instructions impliquées** dans une **condition de sortie** ou dans une condition de **branchement qui peut influencer la condition de sortie**
- Duplication d'instructions
  - **création d'une deuxième chaîne de calcul des conditions, indépendante de l'originale**
- Ajout de blocs
  - **de vérification avec les conditions de sortie dupliquées** dans la boucle et à sa sortie pour vérifier la direction prise
  - **de vérification avec les conditions de branchements internes dupliquées** sur chaque branche possible
  - **de gestion d'erreur**



Passe de protection et flot de compilation

Automatisation

Dans LLVM 3.9 au niveau représentation intermédiaire (LLVM IR) via annotation du code

- Représentation indépendante de l'architecture cible
- Solution « utilisable » pour différentes architectures

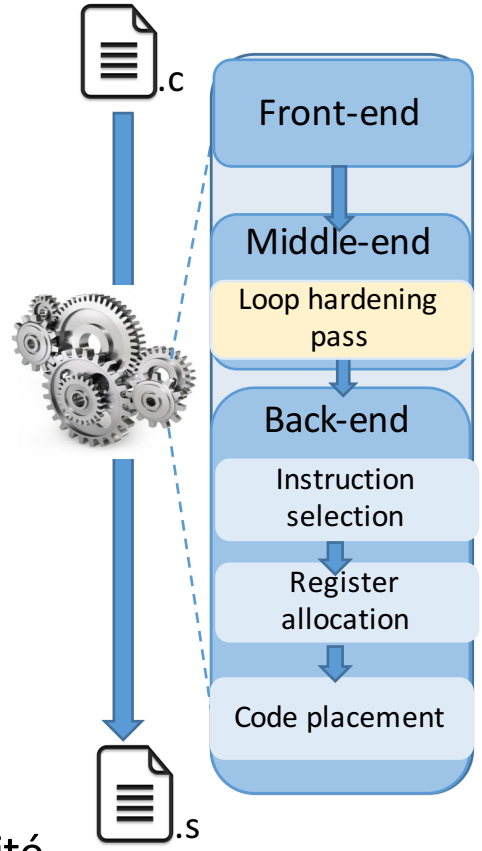
Positionnement dans le flot de compilation

- Après passes d'optimisation qui peuvent défaire la protection, ou limiter sa couverture (accès mémoire)
- Après les passes qui améliorent la qualité du code produit

Transformations et optimisations post-passe « dangereuses »

- Optimisation type « élimination de redondance »
- Sélection d'instructions
- Allocation de registres
- Placement du code et apparition de sauts inconditionnels

➜ Besoin de désactiver, adapter, ajouter certaines passes pour garantir l'efficacité



Évaluation de la robustesse des boucles protégées

Simulation de fautes directement sur carte (mode débogage)

- Fautes simulées : saut d'instruction et corruption de registre
- 2 architectures cibles : ARM Cortex A53 et ARM Cortex M3

- 3 applications cryptographiques (AES, SHA, ECC)
- Pour 2 niveaux d'optimisation différents

- **99% des fautes détectées**
- Taux de détection de **100% atteignable** en modifiant le back-end

- Total d'environ
  - 600 000 injections de faute sur les versions originales
  - 900 000 injections de faute sur les versions sécurisées

Évaluation de la protection des boucles

Mesures réalisées sur

- 18 applications (SPEC 2006, SQLlite, applications cryptographiques / sécurité)
- Tous les niveaux d'optimisation du compilateur considérés

Couverture de la sécurisation automatisée

- **70% de sorties** de boucle **sécurisées, plus de 50%** des boucles **entièrement sécurisées**
- Condition de sortie non sécurisée si besoin de dupliquer
  - Un appel de fonction – plus d'information nécessaire (fonction pure ou constante)
  - Une lecture « volatile » – pas de duplication possible sans support matériel

Impact sur la performance et la taille de code

- **Augmentation des performances** liée au pourcentage du temps d'exécution passé dans les boucles : de 0 à 50 % avec une **moyenne de 12,5 %** (ARM Cortex M3, ARM Cortex A56)
- **Accroissement de la taille de code** de 1 à 43 % avec une **moyenne de 14 %**

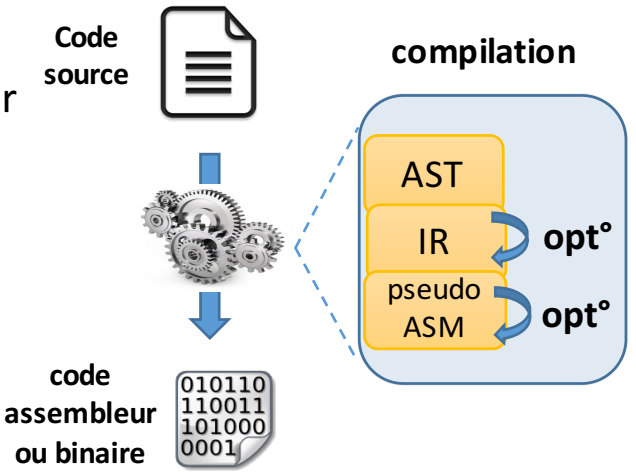
Sécurisation de code et compilation

Niveau IR ≈ code source

- Sémantique élevée, annotations du code disponibles
- Transformations et optimisations post-sécurisation dangereuses
- Problème de correspondance modèle de faute niveau RI et assembleur

Niveau back-end ≈ assembleur

- Modèle de faute plus proche des effets réels
- Manque d'information de plus haut niveau, analyses pré-sécurisation
- Optimisations post-sécurisation ou modifications du back-end



Points difficiles

- Manque de compréhension des liens entre modèles de faute et effet de la compilation
- Pas de cadre pour l'analyse et la préservation des propriétés de sécurité
- Pas de support le passage d'informations relatives à la sécurité / propriété visée

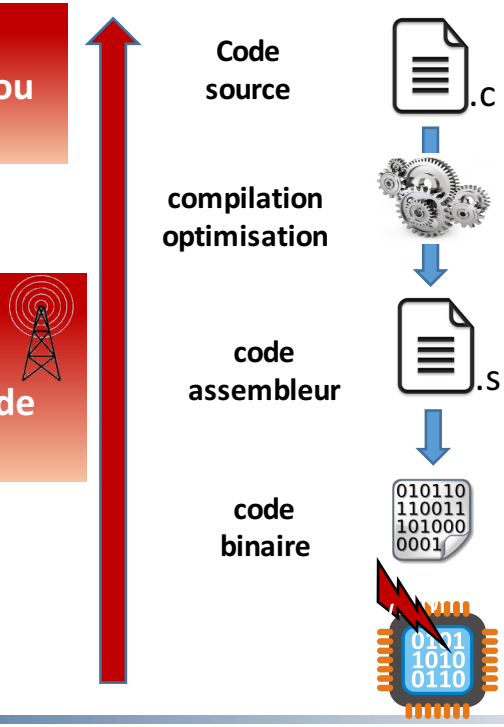
Panorama général

Modélisation des fautes

Attaques par saut, attaque de variable ou combinaison

Saut d'instructions, remplacement d'instruction, lecture de données fautées

Effets physiques



Sécurisation de code

Protection du flot de contrôle

Protection des boucles

Protection contre un saut d'instruction

Conclusion et perspectives

Modélisation des injections de fautes

- À différents niveaux de représentation du code
  - Nécessaire pour la sécurisation de code, l'analyse de vulnérabilités
- Problème de correspondance entre les modèles de différents niveaux
  - Effet de la compilation et de ses optimisations
- Type de fautes possibles et exploitables sur des architectures plus complexes ?

Conclusion et perspectives

Sécurisation de code

- Besoin d'**automatisation** et contre un **large éventail de fautes / attaques**
- **Combinaison** de protections
  - Interaction entre les protections ?
  - Empilement ou combinaison plus judicieuse ?

Compilation et sécurité

- Cadre pour l'**analyse** et la **préservation des propriétés de sécurité**
  - Dans le flot de compilation
  - Pour un analyse de robustesse post-compilation et post-sécurisation

Merci de votre attention

Bibliographie

[AAPS CC 2013] Common Criteria. (2013). Application of Attack Potential to Smartcards, Version 2.9.

[Balash et al. 2011] Balasch, J., Gierlichs, B., and Verbauwhede, I. (2011). An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011), pages 105–114. IEEE.

[Berthomé et al, 2010] Berthomé, P., Heydemann, K., Kauffmann-Tourkestansky, X., and Lalande, J.-F. (2010). Attack model for verification of interval security properties for smart card c codes. 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS'10)

[Berthomé et al, 2012] Berthomé, P., Heydemann, K., Kauffmann-Tourkestansky, X., and Lalande, J.-F. (2012). High level model of control flow attacks for smart card functional security. In Proceedings of the 7th International Conference on Availability, Reliability and Security (ARES'12),.

[Dehbaoui et al. 2013] Amine Dehbaoui, A., Mirbaha, A-P., Moro, N., Dutertre, J-M., Tria, A.: Electromagnetic Glitch on the AES Round Counter. COSADE 2013: 17-31

[Dureuil et al. 2015] Dureuil, L., Potet, M., de Choudens, P., Dumas, C., and Clédière, J. (2015). From code review to fault injection attacks: Filling the gap using fault model inference. In Proceedings of the 14th International Conference Smart Card Research and Advanced Applications (CARDIS), volume 9514 of LNCS, Springer.

[Dureuil et al, FISSC, 2016] Dureuil, L., Petiot, G., Potet, M.-L., Crohen, A., and Choudens, P. D. (2016). FISSC: a Fault Injection and Simulation Secure Collection. In Proceedings of International Conference on Computer Safety, Reliability and Security, volume 9922 of (SAFECOMP).

Bibliographie

[El Bar et al., 2006] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. (2006). The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE, 94(2):370–382.

[Espitau et al. 2016] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, Mehdi Tibouchi: Loop-Abort Faults on Lattice-Based Fiat-Shamir and Hash-and-Sign Signatures. SAC 2016: 140-158

[Kelly et al., 2017] M.S.Kelly, K.Mayes, J.F.Walker. Characterising a CPU fault attack model via run-time data analysis. In 2017 IEEE International Symposium on Hard- 1268 ware Oriented Security and Trust (HOST). 79–84.

[Kumar et al., 2017] S. V. Dilip Kumar; Sikhar Patranabis; Jakub Breier; Debdeep Mukhopadhyay; Shivam Bhasin; Anupam Chattopadhyay; Anubhab Baksi. A Practical Fault Attack on ARX-Like Ciphers with a Case Study on ChaCha20. 2017 Workshop Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC), 2017. IEEE.

[Karaklajic et al, 2013] Karaklajic, D., Schmidt, J-M., and Verbauwhede, I. Hardware Designer's Guide to Fault Attacks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2013.

[Moro et al, 2013] Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., Encrenaz, E. *Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller*. FDTC 2013.

[Nashimoto et al. 2017] Nashimoto, S. Homma, N., Hayashi, Y., Takahashi, J., Fuji, H., Taoki, T.: Buffer overflow attack with multiple fault injection and a proven countermeasure. J. Cryptographic Engineering 7(1): 35-46 (2017)

[Rivière et al. 2015] Rivière, L., Najm, Z., Rauzy, P., Danger, J.-L., Bringer, J., and Sauvage, L. (2015). High precision fault injections on the instruction cache of armv7-m architectures. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2015)

Bibliographie

[Timmers et al., 2016] Timmers, N., Spruyt, A., and Witteman, M. (2016). Controlling PC on ARM Using Fault Injection. 2016 Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC 2016), pages 25–35, Santa Barbara, CA, USA. IEEE.

[Timmers et al., 2017] Timmers, N., Mune, C. Escalating Privileges in Linux Using Voltage Fault Injection. Workshop Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC), 2017. IEEE.

[Vasselle et al., 2017] Vasselle, A., Thiebeauld, H., Maouhoub, Q., Morisset, A. and Ermeneux, S. Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot. 2017 Workshop Fault Diagnosis and Tolerance in Cryptography Workshop (FDTC), 2017. IEEE.

[Verbauwhede et al., 2011] Verbauwhede, I., Karaklajic, D., and Schmidt, J.-M. (2011). The Fault Attack Jungle - A Classification Model to Guide You. In Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011), IEEE.

[Yuce et al., 2017] Yuce, B., Ghalaty, N. F., Deshpande, D., Santapure, H., Conor, P., Nashandali, L., and Schaumont, P. (2017) Analyzing the Fault Injection Sensitivity of Secure Embedded Software. ACM Trans. Embedded Comput. Syst. 16(4): 95:1-95:25

[Yuce et al., JHSS 2018] Yuce, B., Schaumont, P. and Witteman, M. Fault. Attacks on Secure Embedded Software: Threats, Design, and Evaluation. Journal of Hardware and System Security, May 2018.

[Zussa et al., 2014] Zussa, L., Dehbaoui,A. , Tobich,K. , Dutertre, J-M., Maurine, P., Guillaume-Sage, L., Clediere, J., and Tria, A. Efficiency of a glitch detector against electromagnetic fault injection. In Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014.

Bibliographie

[Goloubeva et al., 2005] Goloubeva, O., Rebaudengo, M., Sonza Reorda, M., and Violante, M. (2005). Improved software-based processor control-flow errors detection technique. In Annual Reliability and Maintainability Symposium, pages 583–589, Alexandria, VA, USA. IEEE [Akkar et al., 2003] Akkar, M.-L., Goubin, L., and Ly, O. (2003). Automatic Integration of Counter- Measures Against Fault Injection Attacks. In Proceedings of E-Smart'2003, pages 1–13, Nice.

[Reis et al., 2005] Reis, G., Chang, J., Vachharajani, N., Rangan, R., and August, D. (2005). SWIFT: Software Implemented Fault Tolerance. In Proceedings of the 2005 International Symposium on Code Generation and Optimization (CGO'05), pages 243–254. IEEE.

[Barry et al., 2016] Barry, T., Couroussé, D., and Robisson, B. (2016). Compilation of a Countermeasure Against Instruction-Skip Fault Attacks. In Proceedings of the 3rd Workshop on Cryptography and Security in Computing Systems (CS2), pages 1–6, Prague, Czech Republic. ACM Press.

[Barenghi et al. 2010] Barenghi, A., Breveglieri, L., Koren, I., Pelosi, G., and Regazzoni, F. (2010). Countermeasures against fault attacks on software implemented AES. In Proceedings of the 5th Workshop on Embedded Systems Security (WESS'10), pages 1–10, New York, New York, USA. ACM Press.

[De Keulenaer et al., 2016] De Keulenaer, R., Maebe, J., De Bosschere, K., and De Sutter, B. (2016). Link-time smart card code hardening. International Journal of Information Security, 15(2):111–130.

[Moro et al] Moro, N., Heydemann, K., Encrenaz, E. and Robisson, R. *Formal verification of a software countermeasure against instruction skip attacks.* Journal of Cryptographic Engineering 2014

[Proy et al., 2016] Proy, J., Heydemann, K., Berzati, A., and Cohen, A. *Compiler-Assisted Loop Hardening Against Fault Attacks.* ACM Transaction on Architecture and Code Optimization. December 2017

[AAPS CC 2013] Common Criteria. (2013). Application of Attack Potential to Smartcards, Version 2.9.