Génération de Code au Run Time pour la Sécurisation de Composants

Workshop mi-parcours

Saclay, 3 Décembre 2015

Genèse

- Les attaques sur le matériel soit par canaux auxiliaires soit par faute sont un enjeu considérable,
- Les industriels de l'industrie de la carte à puce sont au fait:
  - Composants tamper resistant,
  - Programmation sécurisée tolérant les fautes,
  - De nombreuses recettes de cuisine *ad hoc*.
- Le monde de l'IoT et de l'embarqué:
  - Composant *off the shelf*
  - Programmation orientée fonctionnalités
  - Pas de savoir faire dédié.

Genèse

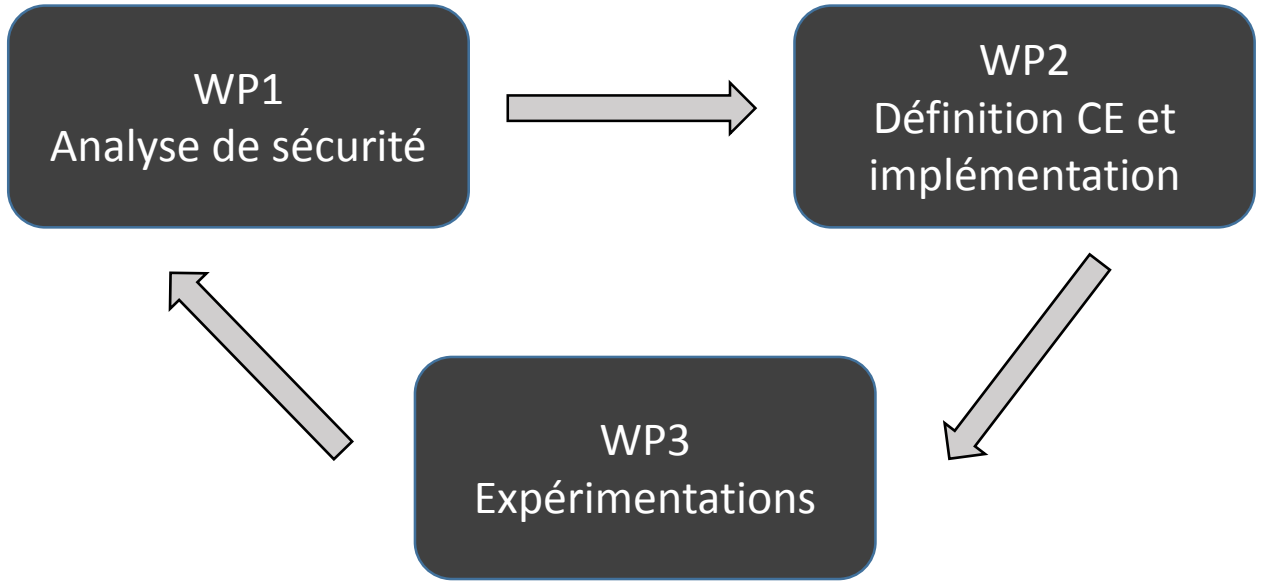
- Les contre mesures contre les attaques peuvent être automatisées,
- Généralement à la compilation,
  - Vision statique du fonctionnement dynamique,
  - Certifiable au sens CC.
- Le projet Cogito
  - Solution purement logicielle pour la portabilité et facilité de généralisation
  - Combinaison des leviers de polymorphisme présentés dans l'état de l'art
  - Surcoût modeste (temps d'exécution et empreinte mémoire)
  - Savoir faire embarqué dans le code

Le projet ANR

- Domaine d'application : sécurité des composants, logiciel embarqué.
- Objectif du projet : Exploration de l'intérêt de la génération de code au run time, et preuve de concept
- deGoal (CEA-LIST) : outil pour la conception de générateurs de code au run time
  - Grande portabilité ;
  - Légèreté : preuve de concept sur micro-contrôleur dans 512 octets de RAM

  → Applicable aux éléments sécurisés (e.g. SmartCards) et petits systèmes embarqués
- Partenaires :
  - CEA : Génération de code au runtime, compilation, apport de la techno deGoal.
  - INRIA Rennes : Sécurité logiciel, attaques logiques et physiques, Smart Cards
  - ENSMSE : Sécurité des composants, attaques physiques et contre-mesures

Le projet ANR



- Actuellement à mi-parcours,
  - Premiers résultats disponibles,
  - Plusieurs publications scientifiques,
  - Un CE achevé, un en cours d'expérimentation

Principe de l’attaque matérielle

L’attaquant procède en deux temps :

- Premier temps : analyse globale et recherche de faiblesses
- Second temps : attaque ciblée sur un point de faiblesse

Procédés mis en œuvre :

- Cryptanalyse :

  En dehors du périmètre scientifique du projet
- Rétro-conception / reverse engineering

  Inspection matérielle : décapsulation, abrasion, etc.

  Inspection logicielle : debug, memory dumps, analyse de code, etc.
- Attaques passives **:** attaques par canaux cachés

  Observations électromagnétiques, électriques, acoustiques, temps d’exécution, etc.
- Attaques actives **:** attaques en fautes

  Sur/sous-alimentation, laser, illumination, corruption horloge, etc.

Etat de l’art

Solutions matérielles

- Non-deterministic processor [May 2011b]
- Random register renaming [May 2011a]
- Exécution d’instructions factices (dummy instructions, ≠NOP) [Ambrose 2007]
- Instruction shuffler [Bayrak 2012]

Solutions logicielles

- Insertion de NOPs [Coron 2009, Coron 2010]
- Génération de code polymorphique avec un compilateur statique [Amarilli 2011]
- Code auto-modifiant sur un langage interprété [Amarilli 2011]
- Code morphing : instanciation de fragments polymorphiques de code au runtime [Agosta 2012]
- Sélection aléatoire parmi plusieurs instances compilées statiquement [Morpho 2013, Agosta 2014]
- Randomisation du flot de contrôle d’un programme [Crane 2015]

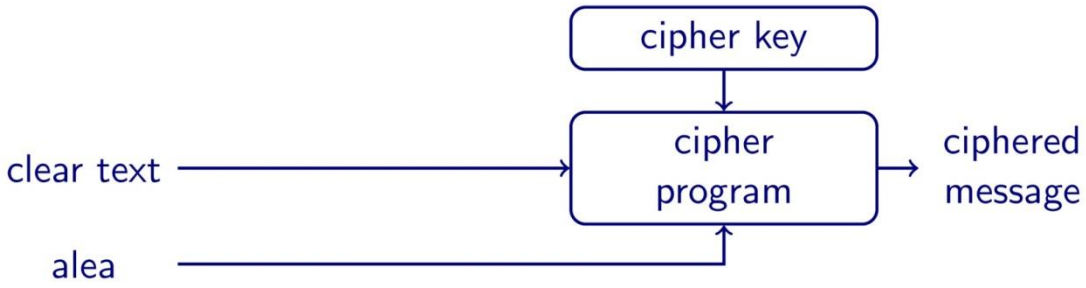
Polymorphisme de Code

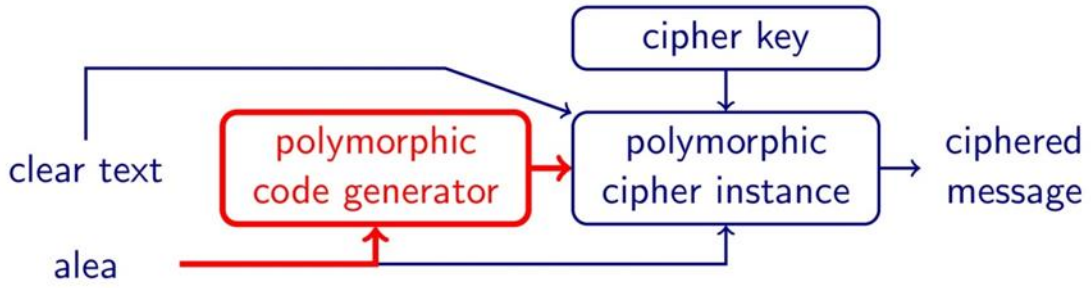
- Modifier régulièrement le comportement observable du composant sécurisé, *lors de son exécution*, sans changer ses propriétés fonctionnelles
- Accroître la difficulté de la rétro-conception du logiciel
  - Le code sécurisé change de forme
  - Le code sécurisé reste inaccessible tant que la cible ne fonctionne pas
- Accroître la difficulté des attaques physiques
  - Le polymorphisme affecte les propriétés spatiales et temporelles du code : **impact sur l'efficacité des attaques par canaux cachés et par fautes**
  - Compatible avec l'état de l'art des protections matérielles et logicielles (masking, redondance, etc.)

Polymorphisme de Code

- Principe: un générateur de code dédié à un fragment de code
  - Spécialisation du compilateur
  - Génération à la volée du code
- deGoal: outil pour la génération de code (machine) au runtime sur les systèmes embarqués
  - La génération de code est rapide
  - Les générateurs de code sont légers
  - Preuve de concept sur le MSP430 (Texas Instruments) avec seulement 512 octets de RAM
- Dans le cadre du projet COGITO
  - Portage pour ARM Cortex M3
  - Evaluation avec différents cas d'étude

Génération de Code





- **Diversification dynamique du code**
  - Allocation aléatoire de registres (*random register allocation*)
  - Sélection aléatoire d'instructions (*random instruction selection*)
  - Mélange des instructions (*instruction shuffling*)
  - Insertion d'instructions factices (*noise instructions*)

COGITO Cas d’étude

- Deux cas d'études initiaux (AES, PIN Code sécurisé), un troisième (Interpréteur Java Card)
  - Fréquence de rafraichissement du code différent
  - Nature des algorithmes (cryptographique, sécurité et machine virtuelle)
  - Type d'attaque (Reverse, corrélation, template,…)
- Les trois cas d'études sont portés sur la plateforme cible STM 32
- Ont été sécurisés par polymorphisme de code : AES, PIN Code (en cours)

Verrous à lever

- Qualité du générateur d'instance
  - Le code généré est il suffisamment robuste ?
  - Le générateur d'instance est il remarquable par SCA ? Adéquation des paramètres au type d'attaque…
- Une approche dynamique est elle certifiable ?
  - Extension de la note ANSSI de février 2015 ?
  - Chaque instance de la fonctionnalité doit elle être certifiée?
- Les uses cases sont ils représentatifs ?
  - D'autres protections seraient nécessaires à d'autres niveaux ?
  - Passage à l'échelle ?

Dissémination

- Publications
  - 2 Publications en conférences internationales avec comité de lecture (CEA)
    - Dont 1 réalisée conjointement par tous les partenaires
  - 1 Publication sur l'archive eprint.iacr (avec comité de revue) (CEA)
  - 2 soumissions en cours
- Séminaires, exposés, posters
  - 8e rencontres de la communauté française de compilation, Nice, Juillet 2014 (CEA)
  - Workshop Cybersécurité, Japon, mars 2015 (INRIA)
  - Rencontre industrie, Pôle d'Excellence Cybersécurité, 8 décembre 2015 Rennes
  - Séminaire invité : sécurité des systèmes électroniques embarqués, DGA-IRISA, Octobre 2015 (CEA)
  - TechDay du CEA-LIST (>200 participants), CEA NanoInnov, Palaiseau, Mai 2015
  - Conférence PHISIC (150 participants), Gardanne, Mai 2015
  - 2 posters à CHES 2015 (Saint-Malo, plus de 400 participants) (CEA)
  - FIC 2016…
- Site internet : http://www.cogito-anr.fr/

Conclusion

- Cogito est un projet académique,
  - Objectif: preuve de concept,
  - Forte volonté de le produire en adéquation avec les besoins de l'industrie,
  - Solution technique validée mais quelques verrous identifiés pour la fin du projet…
- Objectif de la journée:
  - Votre retour sur le concept,
  - Identification d'autres verrous ?

COGITO