Analyse statique au niveau binaire

Adel DJOUDI

Encadré par Sébastien BARDIN
CEA LIST
Début de thèse : Mars 2013

EJCP Mai 2013

Analyse de code binaire : pourquoi ?

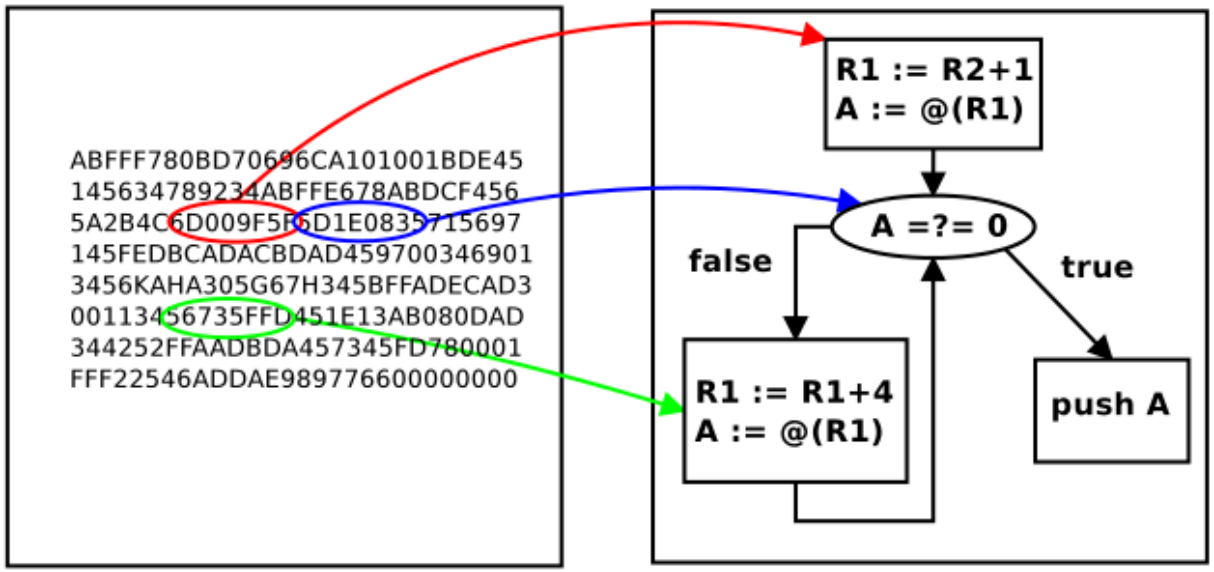
- Avantages sur l'analyse du code source :
  - Exécutable toujours disponible
  - Pas d'intervention de compilateur
- Domaines d'applications :
  - Composants sur étagère
  - Codes mobiles (Malwares, ...)

Difficultés

- Sémantique des données de bas niveau
  - Arithmétique machine, opérations sur des vecteurs de bits
  - Usage d'une mémoire non typée (Tableau d'octets)
- Sémantique de contrôle de bas niveau
  - Pas de distinction claire entre données et instructions
  - Sauts dynamiques (GOTO x)

Reconstruction du CFG : le problème

- Entrées
  - Code exécutable
  - Adresse initiale
  - Décodeur basique : fichier $\times$ adresse $\rightarrow$ instruction $\times$ taille
- Sortie
  - (Sur-approximation) du CFG

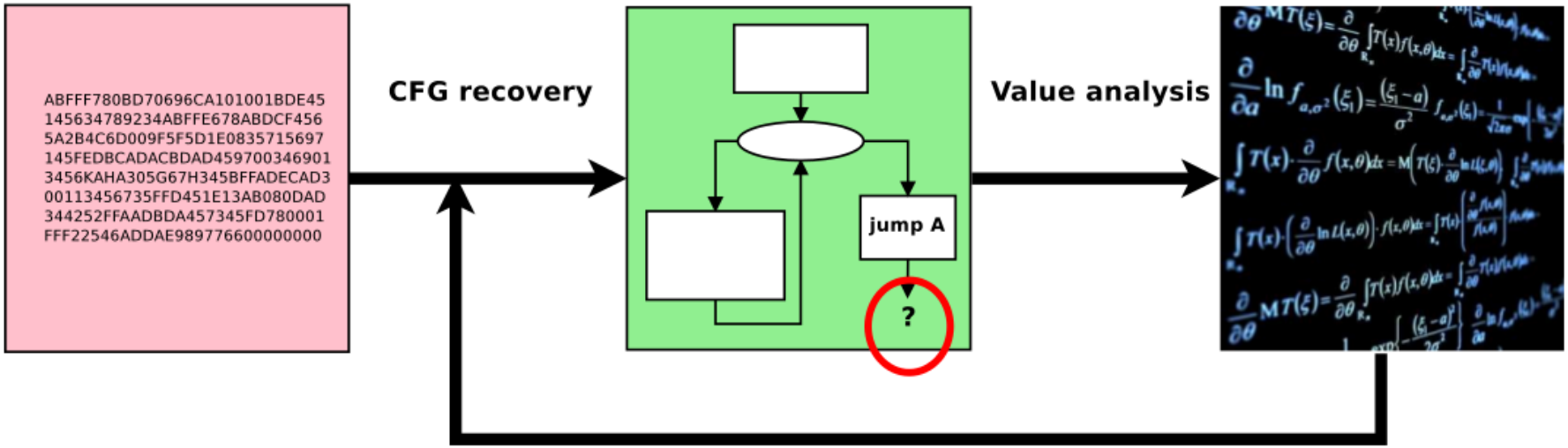


Reconstruction du CFG : le problème (Suite)

- Les successeurs des instructions déterminées de façon syntaxique
  - (adr : move a b) -> successeur à adr + taille + 1
  - (adr : goto 100) -> successeur à 100
  - (adr : ble 100) -> successeur à 100 ou adr + taille + 1
- (adr : goto x) -> successeur à ?
- Outils de désassemblage syntaxique non suffisants

Reconstruction du CFG : le problème (Suite)

- Besoin de combiner reconstruction syntaxique du CFG avec analyse de valeurs (VA)

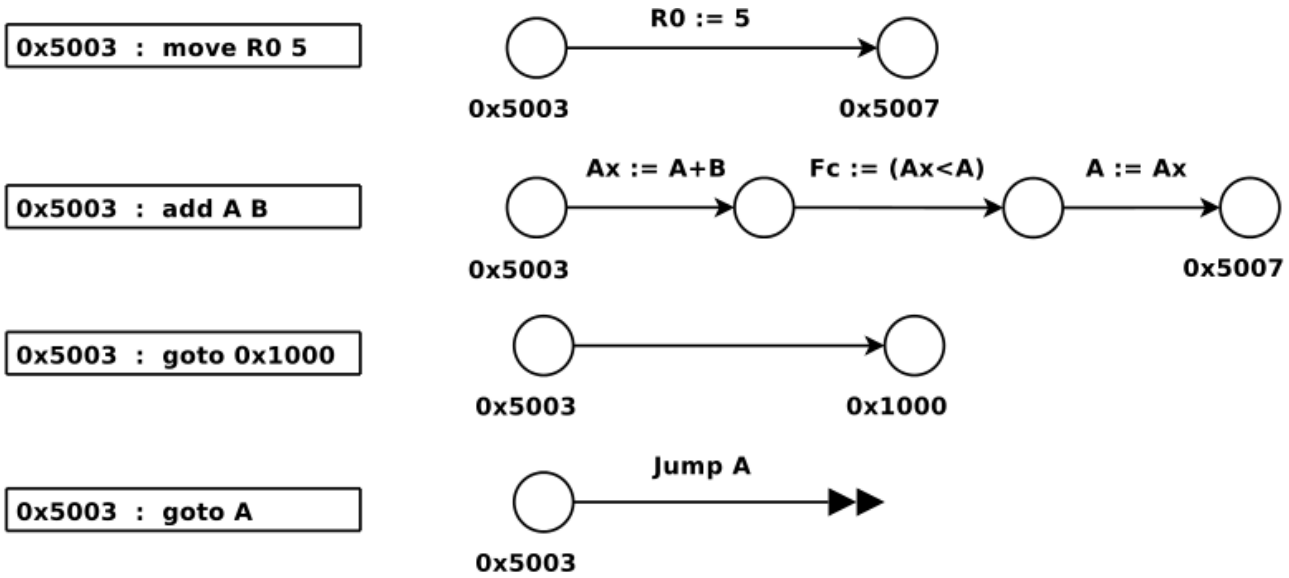


- **VA** imprécise sur $(GOTO\ x) \rightarrow$ Plus d'instructions $\rightarrow$ Plus de propagation $\rightarrow$ **VA** imprécise sur $(GOTO\ x)$ ...

Travaux réalisés

[Sébastion BARDIN and Franck VEDRINE]

- Application sur des systèmes critiques
  - Sans allocation dynamique de mémoire
  - Taille limité des programmes analysés
  - Programmes issus du code C
- Modèle formel : DBA (Dynamic Bitvector Automata)



- Modèle mémoire en un seul espace contigu

Travaux en cours

- Extension du modèle des DBAs avec différentes régions (Contstant, Stack, Malloc(id))
  - Valeurs : (region, offset)
  - Opérations plus restreintes
    - (Constant, v) op (Contant, v') = (Constant, v op v')
    - (R, v) - (R, v') = (Contstant, v - v')
    - (R, v) + (Constant, v') = (R, v + v')
- Buts :
  - Étendre l'analyseur à des systèmes plus généraux (pas critiques)
  - Pouvoir analyser des exécutables issus du C++, libC, malloc/free ...