Analyse de programmes par traçage

Wadie Guizani, Jean-Yves Marion, Daniel Reynaud

Nancy Université - Loria
{guizaniw—jean-yves.marion—reynaudd}@loria.fr
http://lhs.loria.fr

June 7, 2010

Plan

Pourquoi tracer ? (1/3)

Définition : l'analyse binaire, c'est

- de l'analyse de programme
- où le programme est inconnu

$\Longrightarrow$ on a juste un blob binaire

Raisons :

- sauts indirects
  $\Longrightarrow$ flot de contrôle indécidable
- lectures/écritures indirectes
  $\Longrightarrow$ flot de données indécidable
- code auto-modifiant
  $\Longrightarrow$ syntaxe indécidable

Pourquoi tracer ? (1/3)

Définition : l'analyse binaire, c'est

- de l'analyse de programme
- où le programme est inconnu

$\Longrightarrow$ on a juste un blob binaire

Raisons :

- sauts indirects
  $\Longrightarrow$ flot de contrôle indécidable
- lectures/écritures indirectes
  $\Longrightarrow$ flot de données indécidable
- code auto-modifiant
  $\Longrightarrow$ syntaxe indécidable

Pourquoi tracer ? (2/3)

Dans une trace, on a :

- un chemin parcouru
- les lectures/écritures effectives
- la syntaxe effective

... pour l'entrée x donnée

$\Longrightarrow$ on sait si une propriété P(p,x) est vraie

- on ne sait pas montrer des propriétés du type

$$\exists x \mid P(\mathrm{p}, x) \text{ ou } \forall x \; P(\mathrm{p}, x)$$

Pourquoi tracer ? (2/3)

Dans une trace, on a :

- un chemin parcouru
- les lectures/écritures effectives
- la syntaxe effective

... pour l'entrée x donnée

$\Longrightarrow$ on sait si une propriété P(p,x) est vraie

- on ne sait pas montrer des propriétés du type

$$\exists x \mid P(\mathrm{p}, x) \text{ ou } \forall x \; P(\mathrm{p}, x)$$

Pourquoi tracer ? (2/3)

Dans une trace, on a :

- un chemin parcouru
- les lectures/écritures effectives
- la syntaxe effective

... pour l'entrée x donnée

$\Longrightarrow$ on sait si une propriété P(p,x) est vraie

- on ne sait pas montrer des propriétés du type

$$\exists x \mid P(\mathrm{p}, x) \text{ ou } \forall x \; P(\mathrm{p}, x)$$

Pourquoi tracer ? (3/3)

- les traces sont des objets propres et bien définis
- elles permettent de séparer :
  - l'extraction de la trace (programmation système)
  - l'analyse (partie algorithmique)
- en particulier, il y a plusieurs alternatives FOSS pour la partie extraction de traces
- applications à :
  - l'analyse de programmes malveillants
  - la recherche de vulnérabilités

Pourquoi tracer ? (3/3)

- les traces sont des objets propres et bien définis
- elles permettent de séparer :
  - l'extraction de la trace (programmation système)
  - l'analyse (partie algorithmique)
- en particulier, il y a plusieurs alternatives FOSS pour la partie extraction de traces
- applications à :
  - l'analyse de programmes malveillants
  - la recherche de vulnérabilités

Pourquoi tracer ? (3/3)

- les traces sont des objets propres et bien définis
- elles permettent de séparer :
  - l'extraction de la trace (programmation système)
  - l'analyse (partie algorithmique)
- en particulier, il y a plusieurs alternatives FOSS pour la partie extraction de traces
- applications à :
  - l'analyse de programmes malveillants
  - la recherche de vulnérabilités

Exemple : BitBlaze

- projet d'analyse binaire de l'Université de Berkeley
- TEMU : outil d'analyse dynamique basée sur QEMU
  - extraction de trace
  - propagation de tainte
  - Windows/Linux
  - sous LGPL
- Vine : outil d'analyse statique
  - traduction en langage intermédiaire (façon REIL de Zynamics)
  - traduction en langage formel pour solveurs de contraintes (STP)

$\Longrightarrow$ Automatic Patch-Based Exploit Generation
$\Longrightarrow$ Renovo: A Hidden Code Extractor for Packed Executables
$\Longrightarrow$ Capturing System-wide Information Flow for Malware Detection

Exemple : BitBlaze

- projet d'analyse binaire de l'Université de Berkeley
- TEMU : outil d'analyse dynamique basée sur QEMU
  - extraction de trace
  - propagation de tainte
  - Windows/Linux
  - sous LGPL
- Vine : outil d'analyse statique
  - traduction en langage intermédiaire (façon REIL de Zynamics)
  - traduction en langage formel pour solveurs de contraintes (STP)

$\Longrightarrow$ Automatic Patch-Based Exploit Generation

$\Longrightarrow$ Renovo: A Hidden Code Extractor for Packed Executables

$\Longrightarrow$ Capturing System-wide Information Flow for Malware Detection

Exemple : BitBlaze

- projet d'analyse binaire de l'Université de Berkeley
- TEMU : outil d'analyse dynamique basée sur QEMU
  - extraction de trace
  - propagation de tainte
  - Windows/Linux
  - sous LGPL
- Vine : outil d'analyse statique
  - traduction en langage intermédiaire (façon REIL de Zynamics)
  - traduction en langage formel pour solveurs de contraintes (STP)

$\Longrightarrow$ Automatic Patch-Based Exploit Generation

$\Longrightarrow$ Renovo: A Hidden Code Extractor for Packed Executables

$\Longrightarrow$ Capturing System-wide Information Flow for Malware Detection

Plan

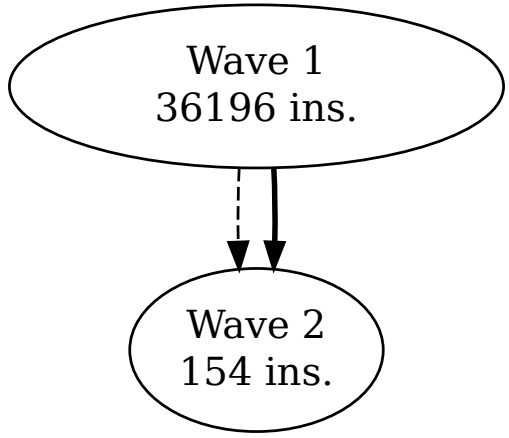
Plan

TraceSurfer

- TraceSurfer est un outil d'analyse du comportement des programmes en mémoire
  - auto-modifications
  - vérification d'intégrité
  - écrasement de code...
- permet de visualiser le comportement des programmes auto-modifiants sous forme de graphe
  - tous les détails sont dans le papier
- permet d'importer des traces dans IDA
- utilise Pin (`http://www.pintool.org`) pour extraire les traces

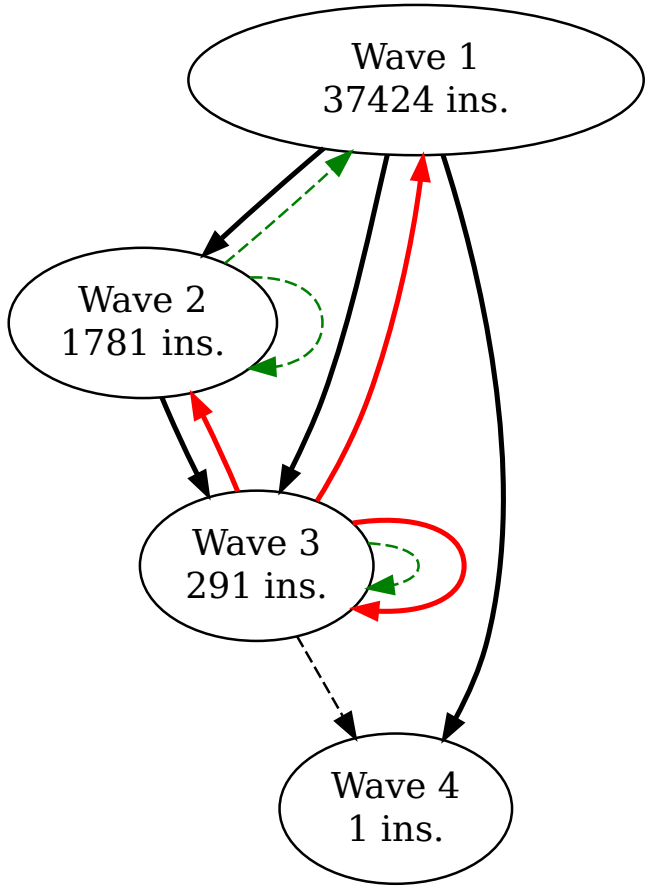
Exemple (1/5)

- hostname packé avec UPX



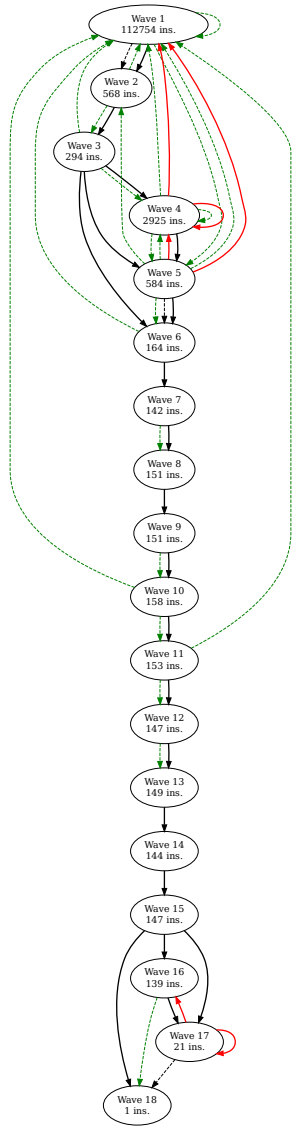
Exemple (2/5)

- hostname packé avec Yoda Protector



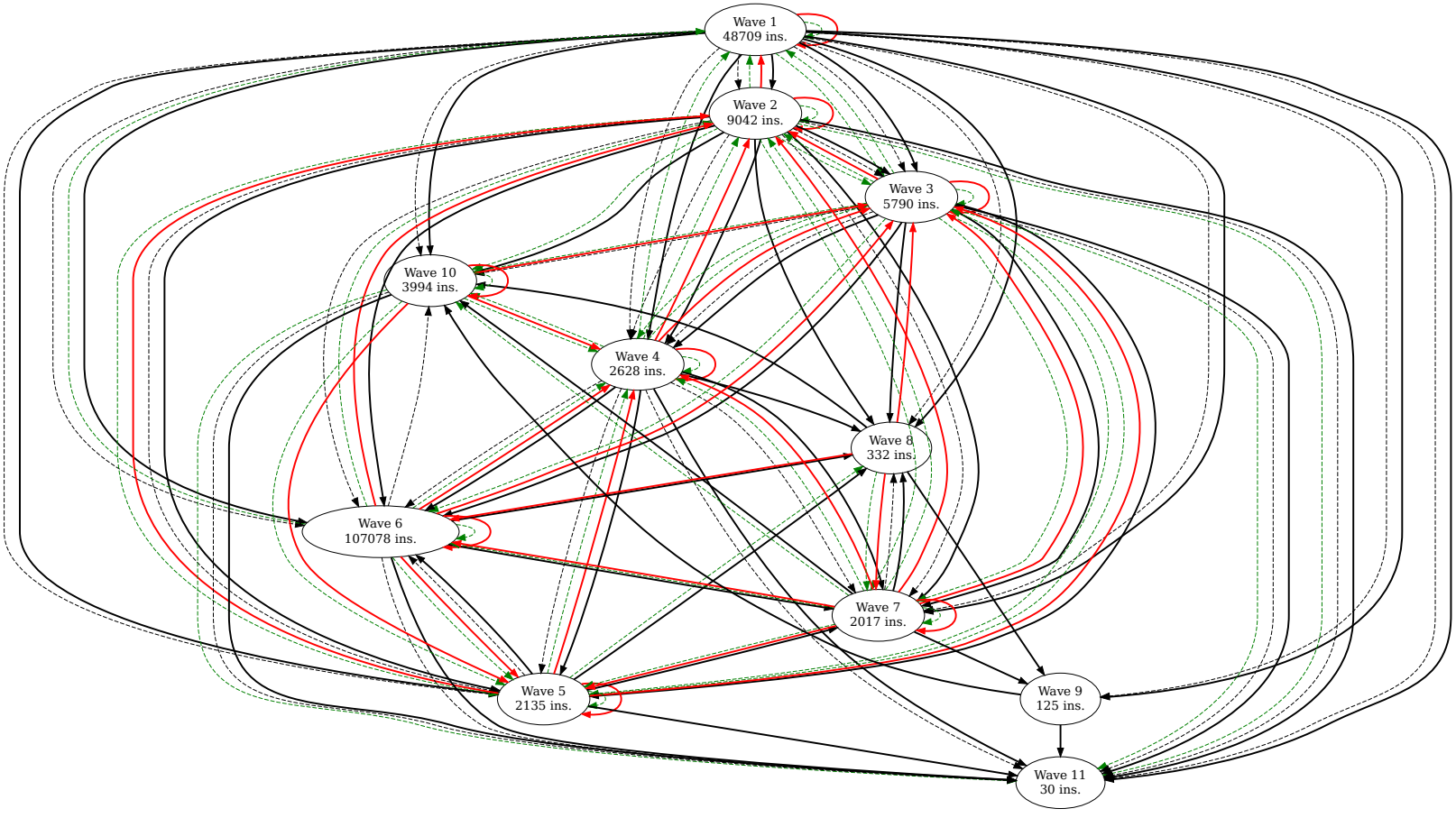
Exemple (3/5)

- hostname packé avec ACProtect



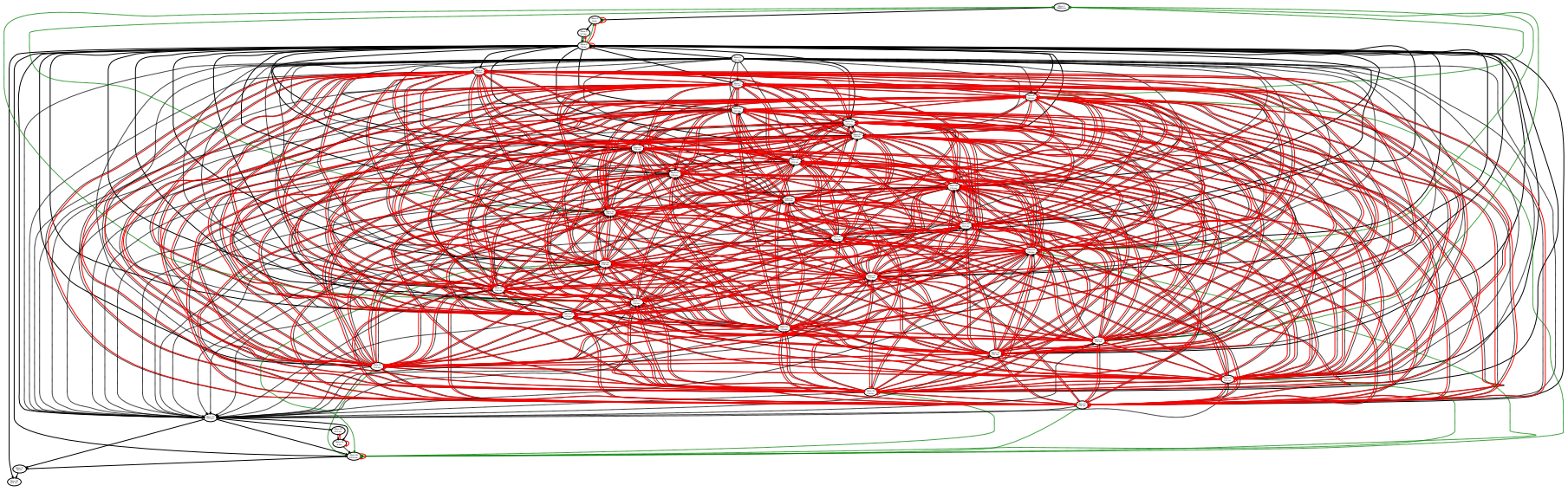
Exemple (4/5)

- hostname packé avec Themida



Exemple (5/5)

- hostname packé avec PESpin



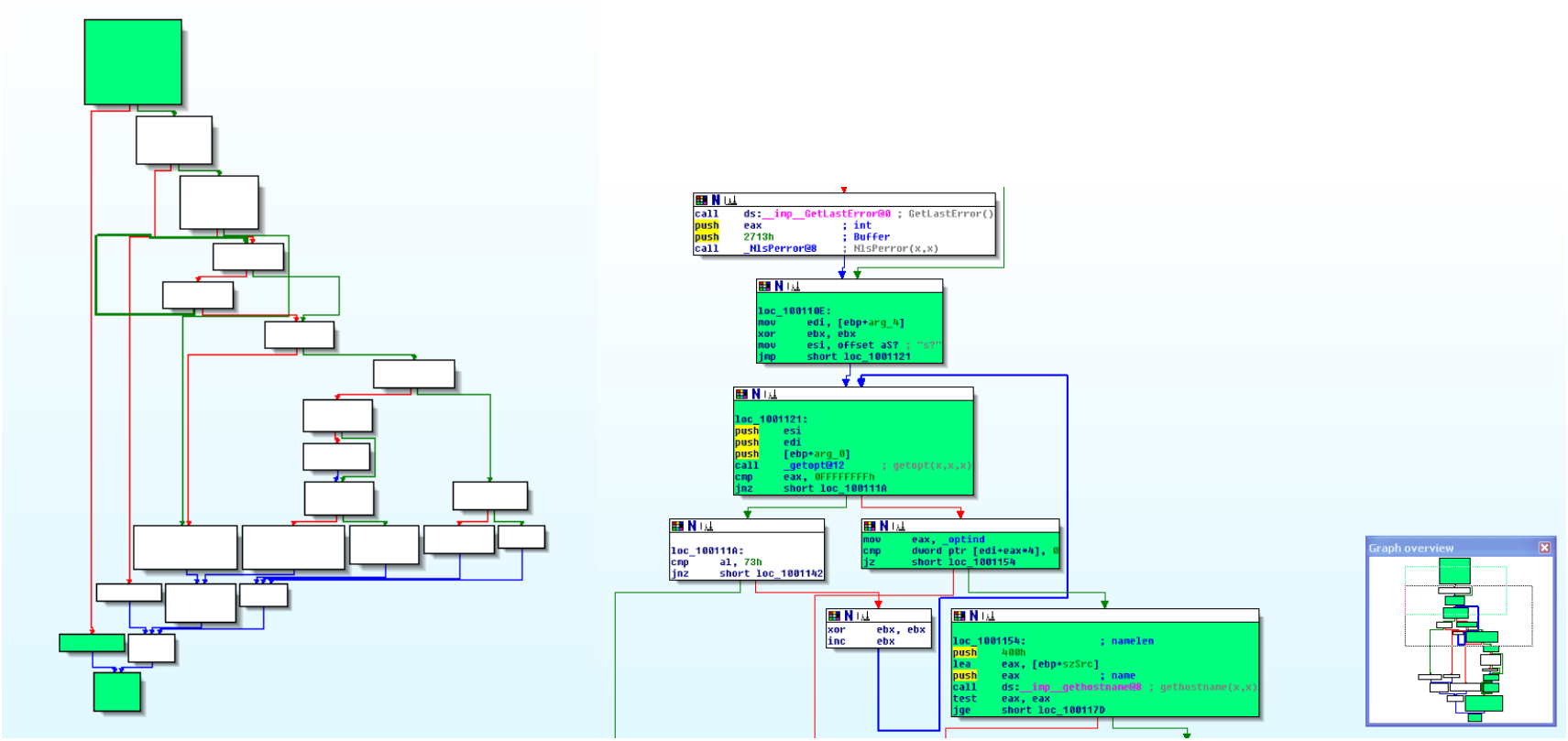
Statistiques

- testé sur hostname packé avec 29 packers différents
- taux de réussite de 86,2%
- en moyenne :
  - 39,9s pour extraire la trace
  - une trace de 123,9mo
  - 14 640 462 instructions par trace
  - 188,3s pour analyser la trace
  - 3,3 vagues de code
  - 19 178 octets générés dynamiquement et exécutés

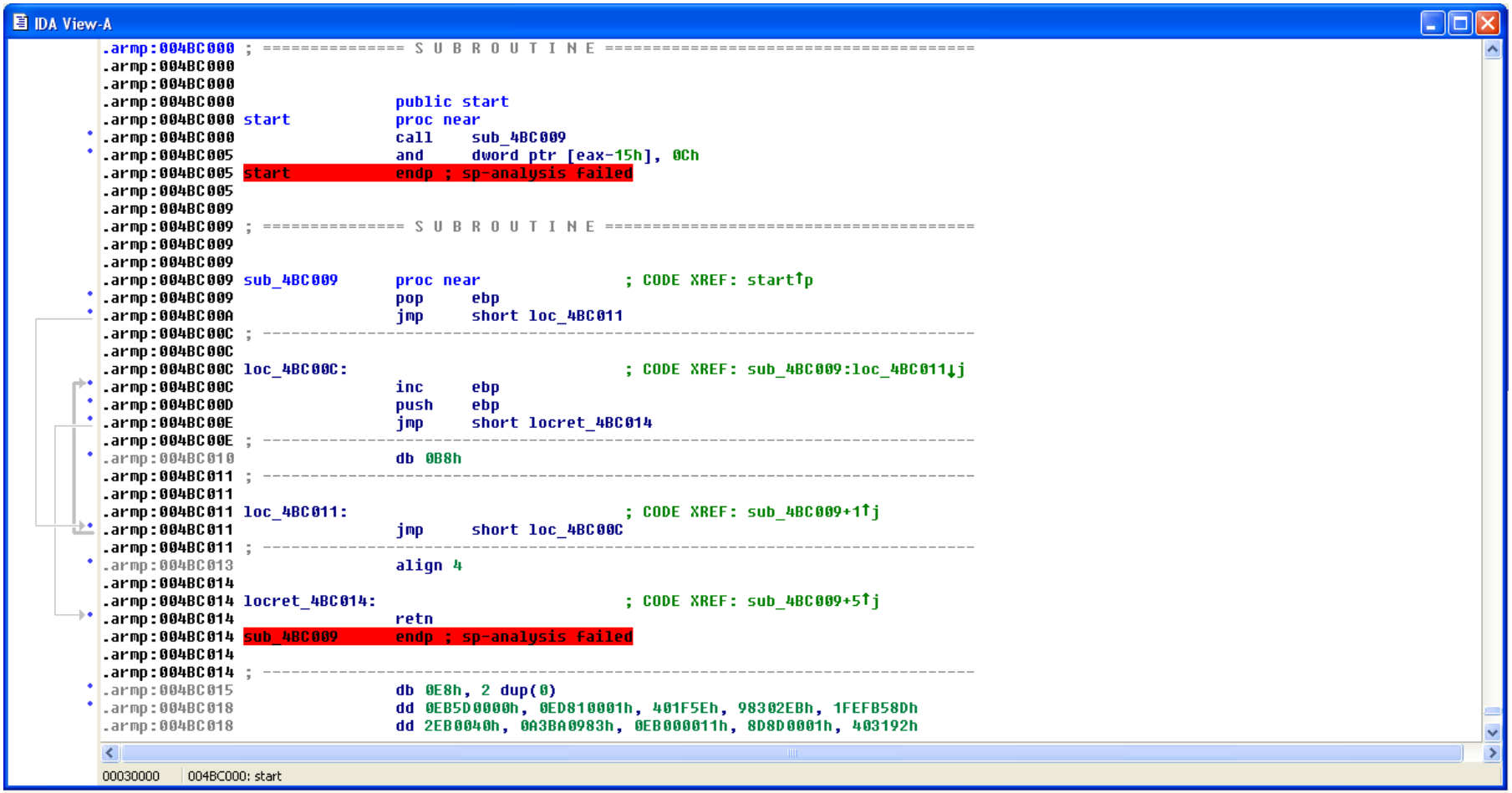
Plan

Intégration à IDA

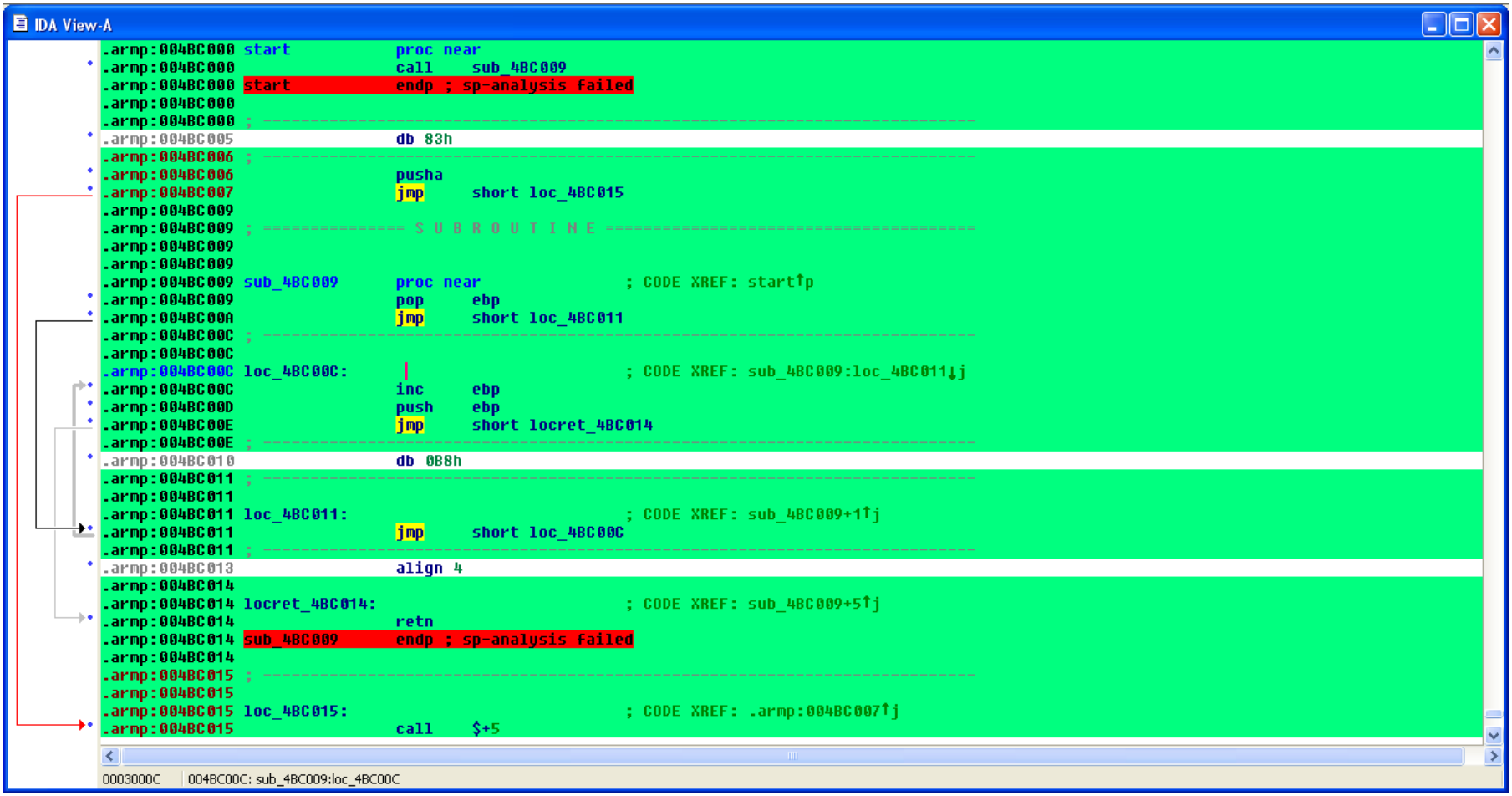
- importer une trace permet de voir le chemin parcouru et corriger le désassemblage



Correction du désassemblage (1/2)



Correction du désassemblage (2/2)



(penser à faire la démo)

Plan

Le Laboratoire de Haute Sécurité du Loria (Nancy)

Cluster utilisé pour l'expérience :

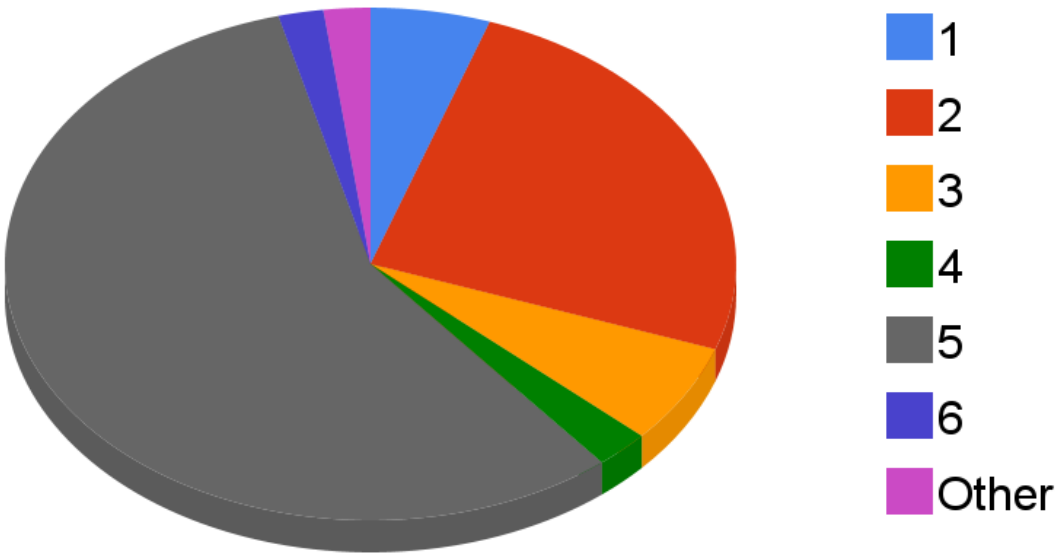
- 1 noeud maître avec la base de malwares
- 12 noeuds esclaves
- 8 coeurs et 16Go de RAM par noeud
- 2 Windows XP virtuels par noeud
- isolé du réseau

Statistiques

- 95 613 binaires
- ils proviennent :
  - du honeypot du LHS
  - d'échanges avec d'autres labos
    - we want you! (in our malware database)
- avec un timeout de 2 minutes par binaire, l'analyse prend 67h30
  - environ 1 400 binaires/h
- taux de succès : 80,53%

Résultats expérimentaux 1/3

- Nombre de vagues de code détectées sur l'ensemble des binaires
  - max : 56 vagues

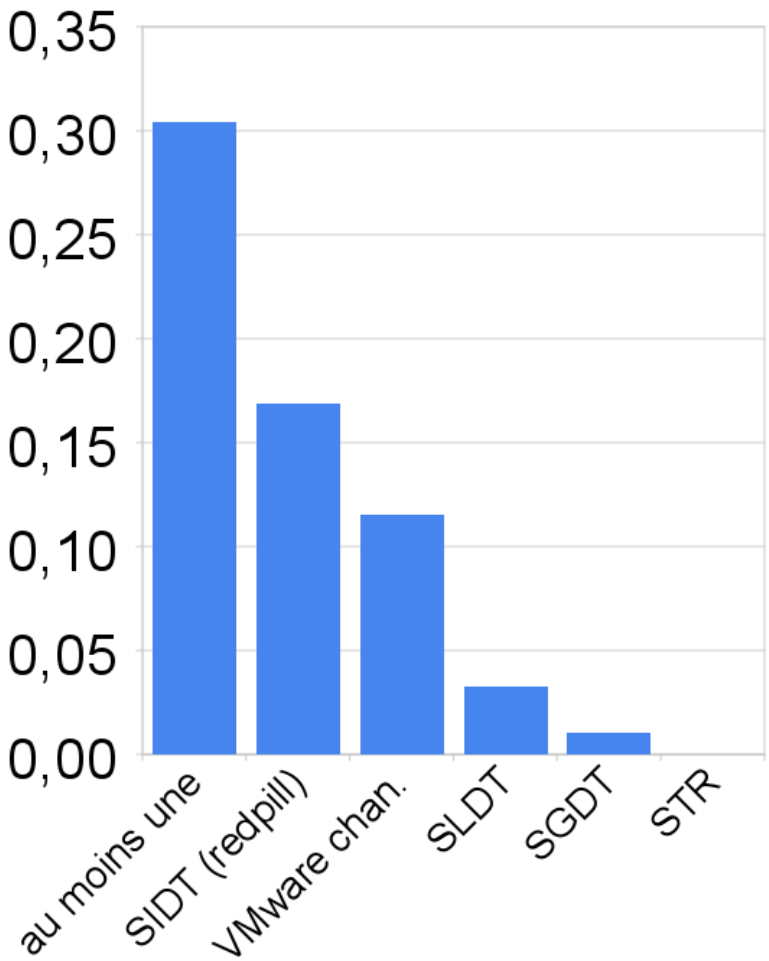


Résultats expérimentaux 2/3

Protection utilisée	Nb de binaires	Proportion
Déchiffrement	63 186	82,07%
Auto-modif. simple	56 520	73,41%
Vérification d'intégrité	45 569	59,19%
Écrasement de code	3 332	4,33%

Résultats expérimentaux 3/3

- Pourcentage d'utilisation de techniques anti-virtualisation

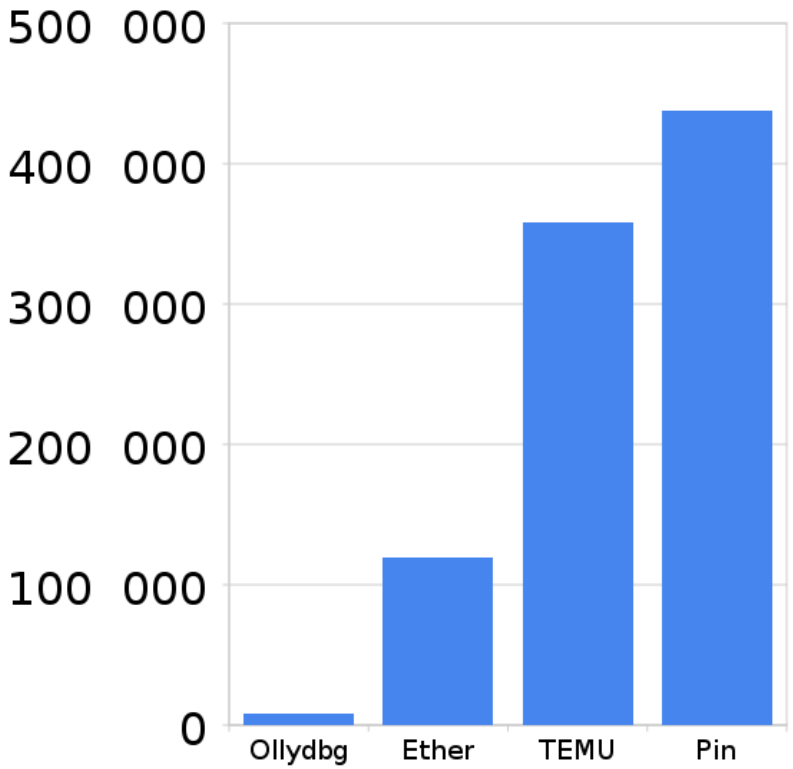


Plan

Plan

Performance

Vitesse (en instructions par seconde) pour différents traceurs :



disclaimer : résultats non contractuels obtenus dans des conditions douteuses

Comparaison

	Vitesse	Furtivité	Simplicité
Instrumentation			
Debugging	—	—	
Emulation			
Virtualisation			—

disclaimer : petites étoiles attribuées de manière arbitraire

Plan

Conclusion de la conclusion

- TraceSurfer, c'est :
  - 160 lignes de C++
  - 750 lignes de Python
- basé sur Pin, mais on pourrait utiliser des traces de TEMU, DynamoRio, Ether, metasm...
- disponible sous GPLv2 : `http://code.google.com/p/tartetatintools/`

Merci public !

more stuff at `http://indefinitestudies.org`

Conclusion de la conclusion

- TraceSurfer, c'est :
  - 160 lignes de C++
  - 750 lignes de Python
- basé sur Pin, mais on pourrait utiliser des traces de TEMU, DynamoRio, Ether, metasm...
- disponible sous GPLv2 : `http://code.google.com/p/tartetatintools/`

Merci public !

more stuff at `http://indefinitestudies.org`

Conclusion de la conclusion

- TraceSurfer, c'est :
  - 160 lignes de C++
  - 750 lignes de Python
- basé sur Pin, mais on pourrait utiliser des traces de TEMU, DynamoRio, Ether, metasm...
- disponible sous GPLv2 :
  `http://code.google.com/p/tartetatintools/`

Merci public !

more stuff at `http://indefinitestudies.org`

Conclusion de la conclusion

- TraceSurfer, c'est :
  - 160 lignes de C++
  - 750 lignes de Python
- basé sur Pin, mais on pourrait utiliser des traces de TEMU, DynamoRio, Ether, metasm...
- disponible sous GPLv2 :
  `http://code.google.com/p/tartetatintools/`

Merci public !

more stuff at `http://indefinitestudies.org`