Désobfuscation de DRM par attaques auxiliaires

Montre moi ta trace et je te dirais qui tu es

SSTIC 2014

QUARKSLAB
INNOVATIVE SECURITY

Auteurs

Camille Mougey

- @Quarkslab au moment des faits
- @CEA-DAM actuellement
- Intéressé par l'obscurcissement, le R-E, les réseaux, l'algorithmie, le Water-Poney, ...

Francis Gabriel

- @Quarkslab
- Intéressé par le R-E, la crypto, les DRM, les maths (ou pas), le patin à roulettes

De quoi va-t-on parler ?

Ingénierie inverse

- Découverte d'une DRM (R&D)
- Méthodologie d'attaque

De quoi va-t-on parler ?

Ingénierie inverse

- Découverte d'une DRM (R&D)
- Méthodologie d'attaque

Trace d'exécution

- Collecte de l'évolution du contexte au cours de l'exécution
- Gestion et analyse des données collectées

De quoi va-t-on parler ?

Ingénierie inverse

- Découverte d'une DRM (R&D)
- Méthodologie d'attaque

Trace d'exécution

- Collecte de l'évolution du contexte au cours de l'exécution
- Gestion et analyse des données collectées

Obscurcissement de code

- Description des protections rencontrées
- Attaques auxiliaires (basées sur la trace d'exécution)

Rappel sur l'obscurcissement (ou *obfuscation*)

Buts

- Protéger du code (tout ou partie)
- Rendre l'analyse plus longue et plus complexe
- Augmenter les coûts en RE

Rappel sur l'obscurcissement (ou *obfuscation*)

Buts

- Protéger du code (tout ou partie)
- Rendre l'analyse plus longue et plus complexe
- Augmenter les coûts en RE

Procédés répandus

- Aplatissement du flot de contrôle
- Protection du flot de données
- Instructions inutiles
- ...

L'obscurcissement d'un binaire, c'est comme un oignon . . .

Constats initiaux

Intéractions réseau

- Observation du contenu des paquets
- Données à forte entropie

$\Rightarrow$ Présence de compression ou de crypto' ?

Constats initiaux

Intéractions réseau

- Observation du contenu des paquets
- Données à forte entropie

$\Rightarrow$ Présence de compression ou de crypto' ?

Analyse du binaire (statique et dynamique)

- Présence d'aplatissement du graphe du flot de contrôle (*code-flattening*)
- Obscurcissement des opérations à l'intérieur des *basic blocks*

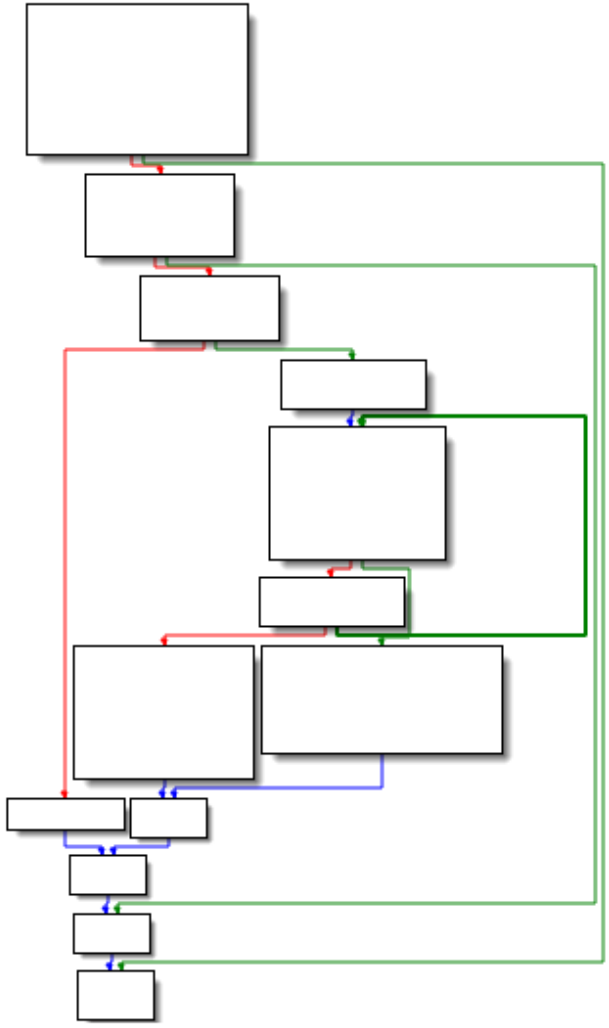
Plan

1. Première couche : Code flattening
   - Présentation
   - Méthodologie
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

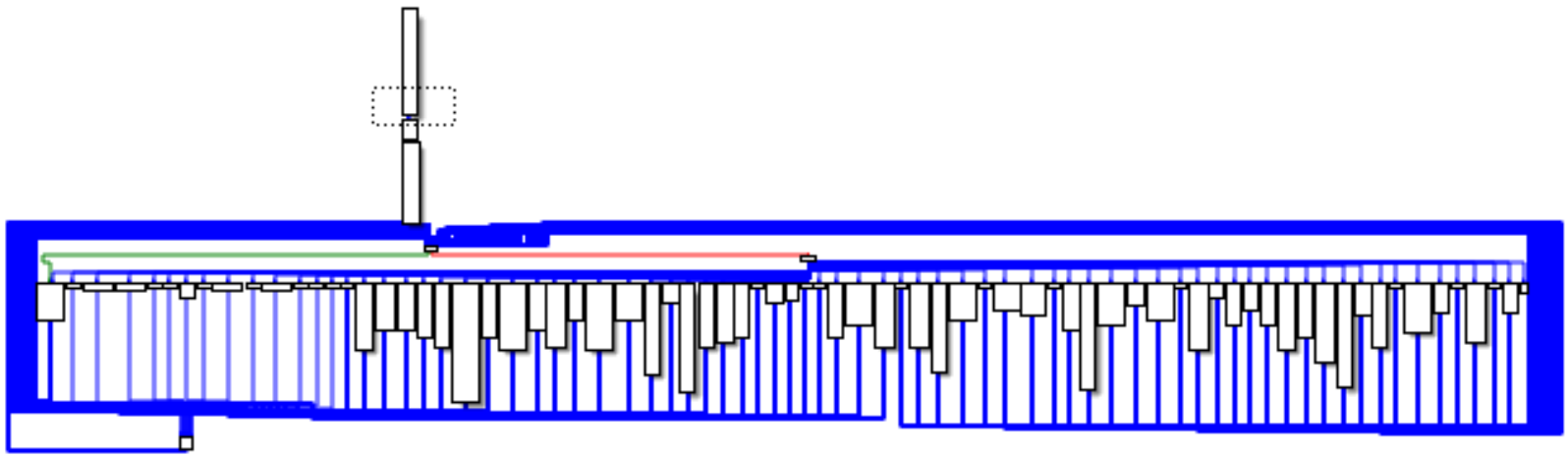
Plan

1. Première couche : Code flattening
   - Présentation
   - Méthodologie
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

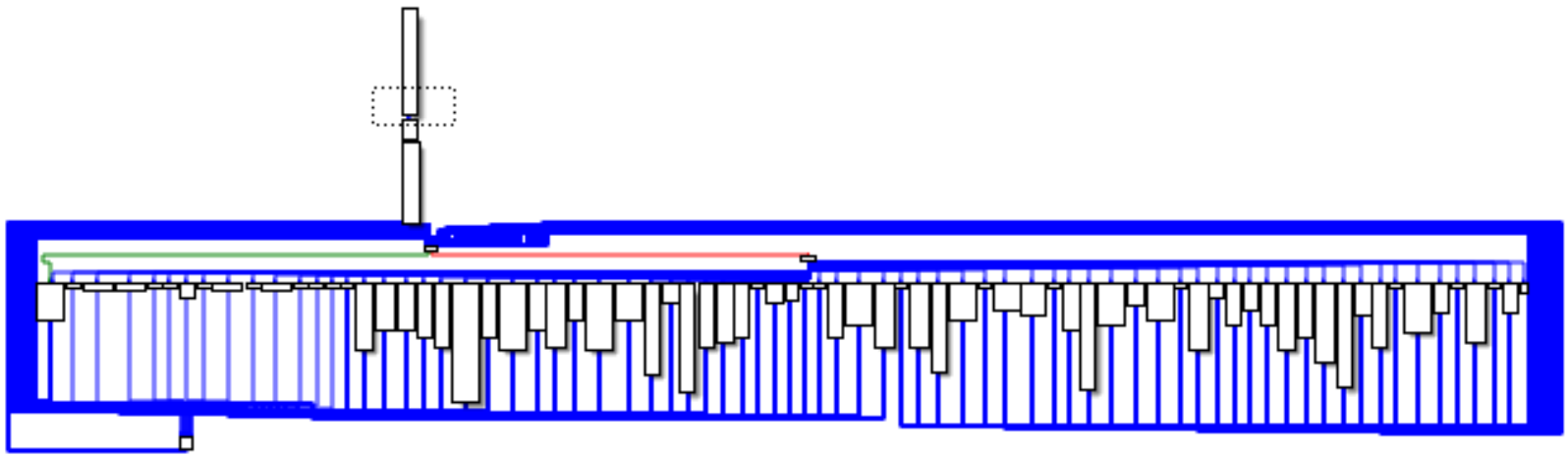
Graphe de contrôle normal



Graphe de contrôle + code flattening



Graphe de contrôle + code flattening



Quelle méthode utiliser ?

Plan

1. Première couche : Code flattening
   - Présentation
   - Méthodologie
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Deux approches possibles

Étudier la protection

- Présentée à SSTIC l'an dernier
- Exécution symbolique/concolique du code cible
- Avantage : Capitalisation des recherches

Si la protection est complexe :

- Beaucoup de ressources nécessaires
- Explosion combinatoire
- Travaux en cours...

Deux approches possibles

Étudier la protection

- Présentée à SSTIC l'an dernier
- Exécution symbolique/concolique du code cible
- Avantage : Capitalisation des recherches

Si la protection est complexe :

- Beaucoup de ressources nécessaires
- Explosion combinatoire
- Travaux en cours...

Cibler une exécution donnée

- Tracer l'exécution du code
- Perte de l'information du CFG
- Un (seul) chemin à analyser
- Avantage : Compréhension plus rapide du code cible

Méthode retenue

Approche par trace d'exécution

1. Enregistrement de l'évolution du contexte
   - état des registres
   - instructions exécutées
   - accès mémoire
2. Développement d'une plateforme pour traiter la trace
3. Développement de modules pour en extraire des informations

Méthode retenue

Approche par trace d'exécution

1. Enregistrement de l'évolution du contexte
   - état des registres
   - instructions exécutées
   - accès mémoire
2. Développement d'une plateforme pour traiter la trace
3. Développement de modules pour en extraire des informations

Concepts à aborder

- Instrumentation : collecte des informations en exécution
- Base de données : stockage intelligent de la trace
- Traitement : accès pertinent aux données

Méthode retenue

Approche par trace d'exécution

1. Enregistrement de l'évolution du contexte
   - état des registres
   - instructions exécutées
   - accès mémoire
2. Développement d'une plateforme pour traiter la trace
3. Développement de modules pour en extraire des informations

Concepts à aborder

- Instrumentation : collecte des informations en exécution
- Base de données : stockage intelligent de la trace
- Traitement : accès pertinent aux données

That's why we made: pTra

Plan

1. Première couche : Code flattening
2. pTra
   - Quoi que c'est ?
   - Quelques mots sur l'implémentation
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Plan

1. Première couche : Code flattening
2. pTra
   - Quoi que c'est ?
   - Quelques mots sur l'implémentation
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

pTra - Ce que l'on souhaite

Python TRace Analyser

- Plate-forme de traitement de trace d'exécution
- But : fournir une API de manipulation de trace
- Complètement modulaire, *scalable*

Contraintes

- Indépendant de l'architecture (ré-utilisabilité)
- Temps de réponse raisonnable (utilisabilité)

pTra - Ce que l'on souhaite

Python TRace Analyser

- Plate-forme de traitement de trace d'exécution
- But : fournir une API de manipulation de trace
- Complètement modulaire, *scalable*

Contraintes

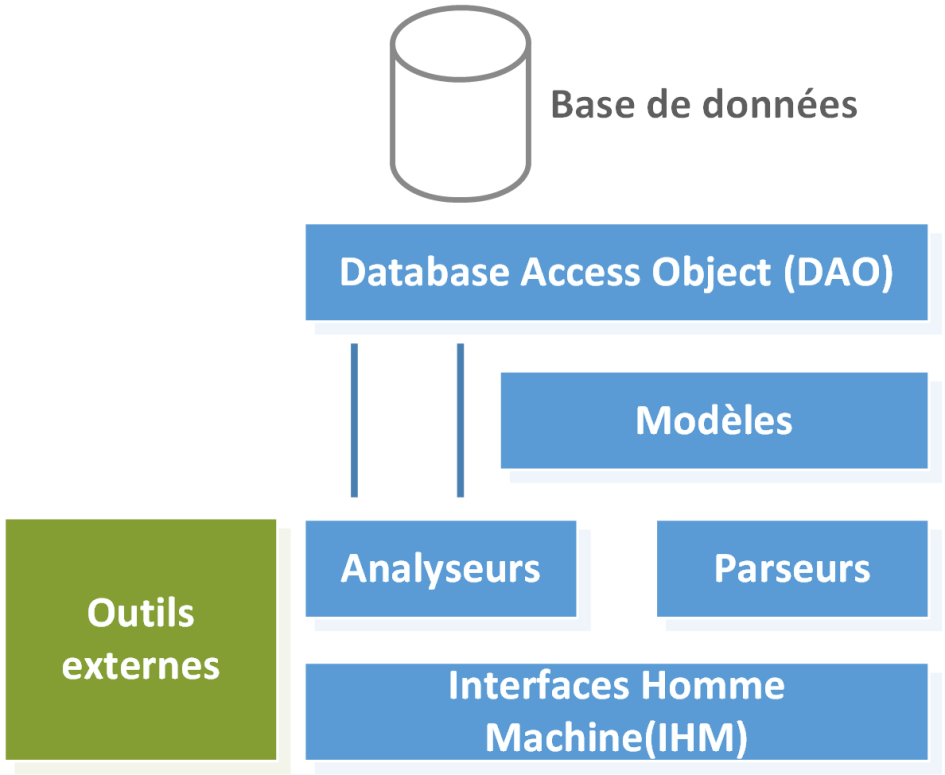
- Indépendant de l'architecture (ré-utilisabilité)
- Temps de réponse raisonnable (utilisabilité)

⇒ De manière générale, être capable d'implémenter rapidement une idée

Plan

1. Première couche : Code flattening
2. pTra
   - Quoi que c'est ?
   - Quelques mots sur l'implémentation
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Architecture logicielle en couches



Choix logiciels

Base de données

- *MongoDB*
  - *Scalable*
  - Non relationnelle, idéale pour le prototypage
- Une base par trace
  - Pas de *lock* inter-trace
  - Permet des hypothèses sur les éléments de la base

Choix logiciels

Base de données

- *MongoDB*
  - *Scalable*
  - Non relationnelle, idéale pour le prototypage
- Une base par trace
  - Pas de *lock* inter-trace
  - Permet des hypothèses sur les éléments de la base

Obtention de trace d'exécution

- Intel PIN
- Sandbox Miasm
- Trace IDA, ollydbg

Choix logiciels

Base de données

- *MongoDB*
  - *Scalable*
  - Non relationnelle, idéale pour le prototypage
- Une base par trace
  - Pas de *lock* inter-trace
  - Permet des hypothèses sur les éléments de la base

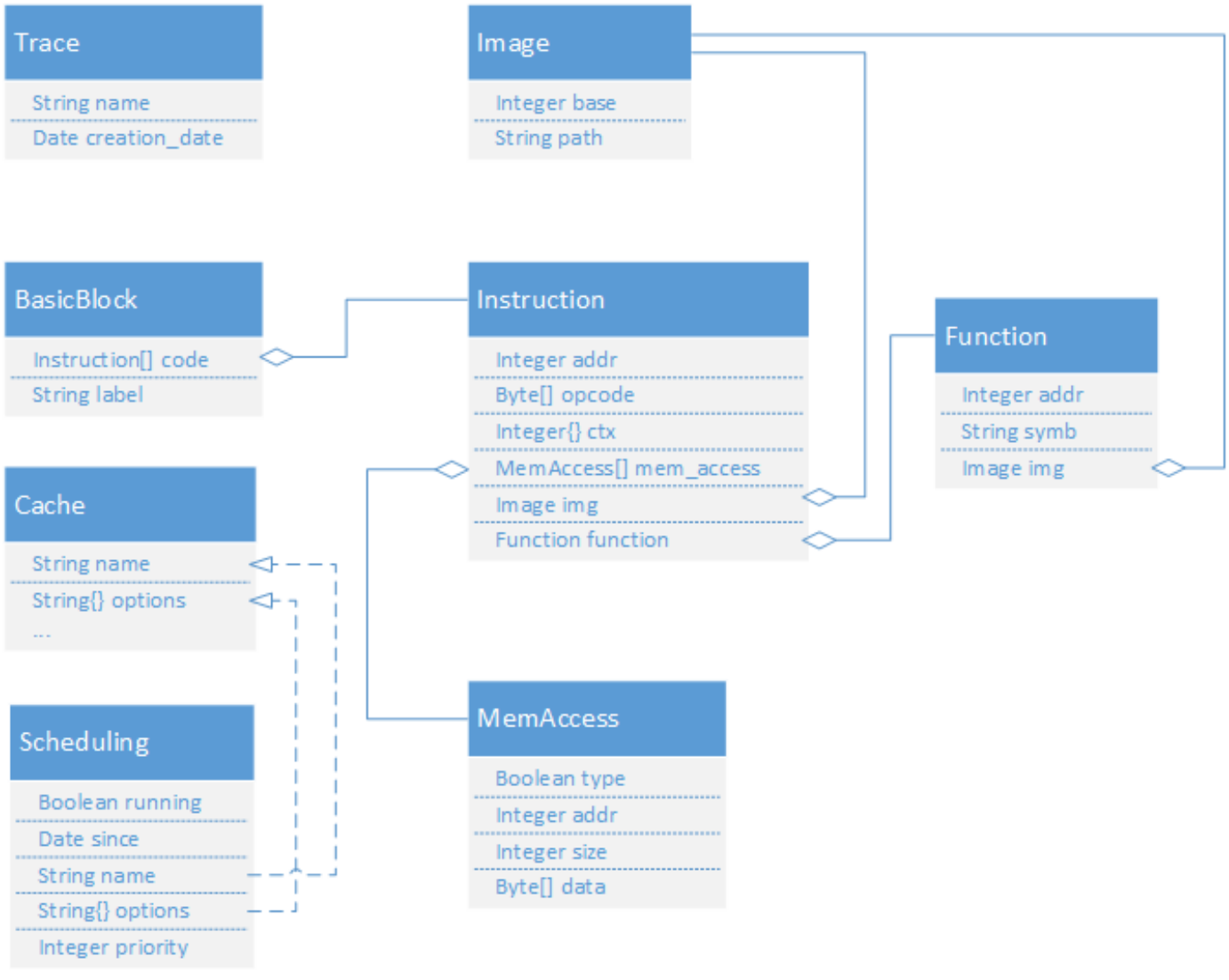
Obtention de trace d'exécution

- Intel PIN
- Sandbox Miasm
- Trace IDA, ollydbg

Désassembleur

- *DiStorm*
- Puis Miasm, pour être architecture indépendant... et avoir une IR

Modèle mémoire



Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
   - Introduction
   - Détection de constantes
   - Obscurcissement des données
   - Data slicing et reconstruction de fonction
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
   - Introduction
   - Détection de constantes
   - Obscurcissement des données
   - Data slicing et reconstruction de fonction
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Reconstruction d'un algorithme - Introduction

Que voulons-nous savoir ?

- Comprendre un algorithme dans sa globalité
- De quelles briques est-il composé (chiffrement, dérivation, . . .)

⇒ La BDD de pTra contient tout ce qu'il nous faut

Reconstruction d'un algorithme - Introduction

Que voulons-nous savoir ?

- Comprendre un algorithme dans sa globalité
- De quelles briques est-il composé (chiffrement, dérivation, . . .)

⇒ La BDD de pTra contient tout ce qu'il nous faut

Comment procéder ?

1. Identifier les briques impliquées (fonctions, crypto)
2. Trouver les entrées et sorties de ces briques
3. Comprendre comment elles sont reliées

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
   - Introduction
   - Détection de constantes
   - Obscurcissement des données
   - Data slicing et reconstruction de fonction
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Détection de constantes - Théorie

Constats

- Un algorithme cryptographique peut comporter certaines constantes "magiques"
- Les fonctions de hachage en sont un bon exemple
- Si un algorithme est présent, ses constantes aussi

Détection de constantes - Théorie

Constats

- Un algorithme cryptographique peut comporter certaines constantes "magiques"
- Les fonctions de hachage en sont un bon exemple
- Si un algorithme est présent, ses constantes aussi

Où les trouver ?

Elles peuvent apparaître à plusieurs endroits :

- Les instructions (cas général de l'analyse statique)
- Les registres processeur
- Les accès mémoire

$\Rightarrow$ pTra fourni un accès direct à ces éléments

Détection de constantes - Pratique

Méthode

- Programmation d'un module pour pTra
- Recherche exhaustive dans la base de données de constantes connues
- Suppression des faux positifs
  - Faible probabilité
  - Regroupement par zone d'apparition
- Simple, rapide et efficace

Détection de constantes - Pratique

Méthode

- Programmation d'un module pour pTra
- Recherche exhaustive dans la base de données de constantes connues
- Suppression des faux positifs
  - Faible probabilité
  - Regroupement par zone d'apparition
- Simple, rapide et efficace

Résultat

- Identification d'un Mersenne Twister (`0x6c078965`)
- Identification claire des 5 constantes de SHA-1 (`0x67452301`, `0xefcdab89`, `0x98badcfe`, `0x10325476`, `0xc3d2e1f0`)

$\Rightarrow$ Identification des fonctions SHA-1 dans le *call graph* (`init`, `update`, `final`)

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
   - Introduction
   - Détection de constantes
   - Obscurcissement des données
   - Data slicing et reconstruction de fonction
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Identification des entrées/sorties - Théorie

Quels sont les buts ?

- Fonctions non identifiées :
  - Comprendre, voire identifier les fonctions
- Fonctions déjà identifiées :
  - Trouver la source des arguments
  - Faire les liens avec les autres algorithmes

$\Rightarrow$ Récupération des entrées et sorties des fonctions

Identification des entrées/sorties - Théorie

Quels sont les buts ?

- Fonctions non identifiées :
  - Comprendre, voire identifier les fonctions
- Fonctions déjà identifiées :
  - Trouver la source des arguments
  - Faire les liens avec les autres algorithmes

⇒ Récupération des entrées et sorties des fonctions

Constats

Étude des accès mémoire pour une fonction donnée

- Si une donnée est traitée, elle sera lue
- Les résultats seront écrits

⇒ pTra fourni un accès direct à ces éléments

Identification des entrées/sorties - Pratique

Méthode

- Pour identifier les sorties :
  - On effectue un différentiel mémoire
  - (état après) - (état avant)
  - On peut éliminer les données écrites mais lues avant la fin (données temporaires)
- Pour identifier les entrées :
  - Données accédées pour la première fois, en lecture
- Ajout de quelques heuristiques (détections de pointeurs, regroupement par bloc, entropie, . . .)

pTra - Différentiel Mémoire - memcpy()

#Addr	#Data	#ASCII	#Size	#Compression Rate
0x02030134	7e 03 00 00 e2 00 00 00 b3 01 00 00 dc 01 00 00	~.....	32	00%
0x02030144	41 02 00 00 9c 02 00 00 42 00 00 00 70 03 00 00	A.....B...p...		
0xbffe21b4	18 22 fe bf	."..	4	00%
0xbffe21c4	58 22 fe bf	X"..	4	00%
0xbffe21cc	6a 3c 50 58	j<PX	4	00%
0xbffe2218	73 01 09 07 75 07 86 71 86 84 01 09 b0 15 f2 75	s...u..q.....u	64	00%
0xbffe2228	86 84 80 03 80 f4 1d 71 03 75 01 1d 03 73 03 88	.....q.u...s..		
0xbffe2238	1d 71 05 f4 07 1d 86 84 f2 82 07 75 03 05 86 80	.q.....u....		
0xbffe2248	f6 86 b0 25 7d 51 53 1d 07 84 82 09 01 82 75 f6	...%}QS.....u.		

Identification des entrées/sorties - Résultats

Dans les faits

- Méthode très efficace pour lier des parties de code entre elles
- Prise de conscience d'une protection : mémoire transformée
  - Les données en mémoire ne sont *jamais* en clair
  - Fonction de dérivation par zone mémoire
  - Pas de *pattern* dans le code
  - Mais indépendant de l'adresse mémoire

Identification des entrées/sorties - Résultats

Dans les faits

- Méthode très efficace pour lier des parties de code entre elles
- Prise de conscience d'une protection : mémoire transformée
  - Les données en mémoire ne sont *jamais* en clair
  - Fonction de dérivation par zone mémoire
  - Pas de *pattern* dans le code
  - Mais indépendant de l'adresse mémoire

Algorithmes identifiés

- Entrées/Sorties validées sur le SHA-1 précédemment identifié
- Entrées du SHA-1 : Certificats $\Rightarrow$ Validation de *cert-chain*
- Présence de l'algorithme RSA dans la trace ($\pm$50 millions d'instructions)

$\Rightarrow$ Identification de la fonction RSA dans la DRM

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
   - Introduction
   - Détection de constantes
   - Obscurcissement des données
   - Data slicing et reconstruction de fonction
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Data slicing et reconstruction de fonction

Définitions

- **Data tainting** : trouver l'ensemble des éléments *influencés* par un élément donné;
- **Data slicing** : trouver l'ensemble des éléments *ayant influencé* un élément donné;

Intuitivement, le tainting avancera dans le temps, le slicing le remontera.

Data slicing et reconstruction de fonction

Définitions

- **Data tainting** : trouver l'ensemble des éléments *influencés* par un élément donné;
- **Data slicing** : trouver l'ensemble des éléments *ayant influencé* un élément donné;

Intuitivement, le tainting avancera dans le temps, le slicing le remontera.

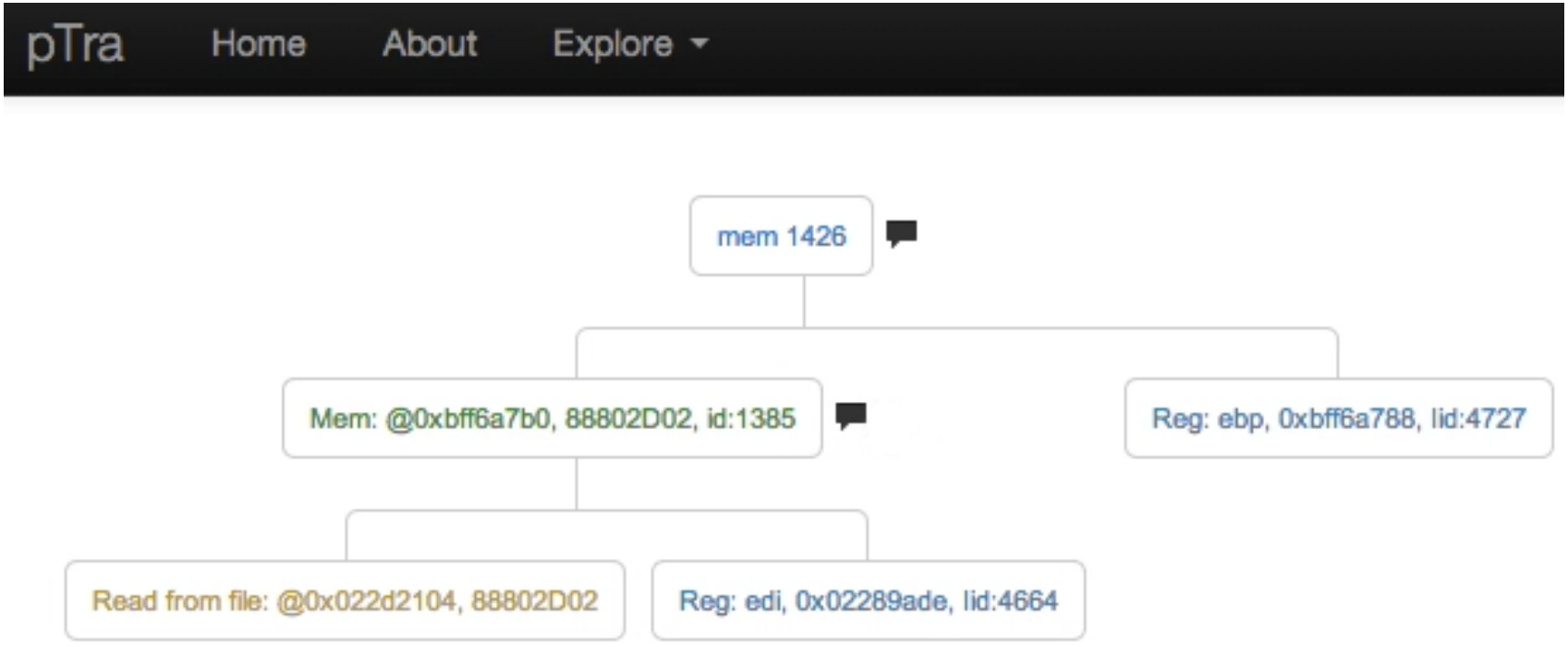
Implémentation : Data slicing

En utilisant l'IR de Miasm :

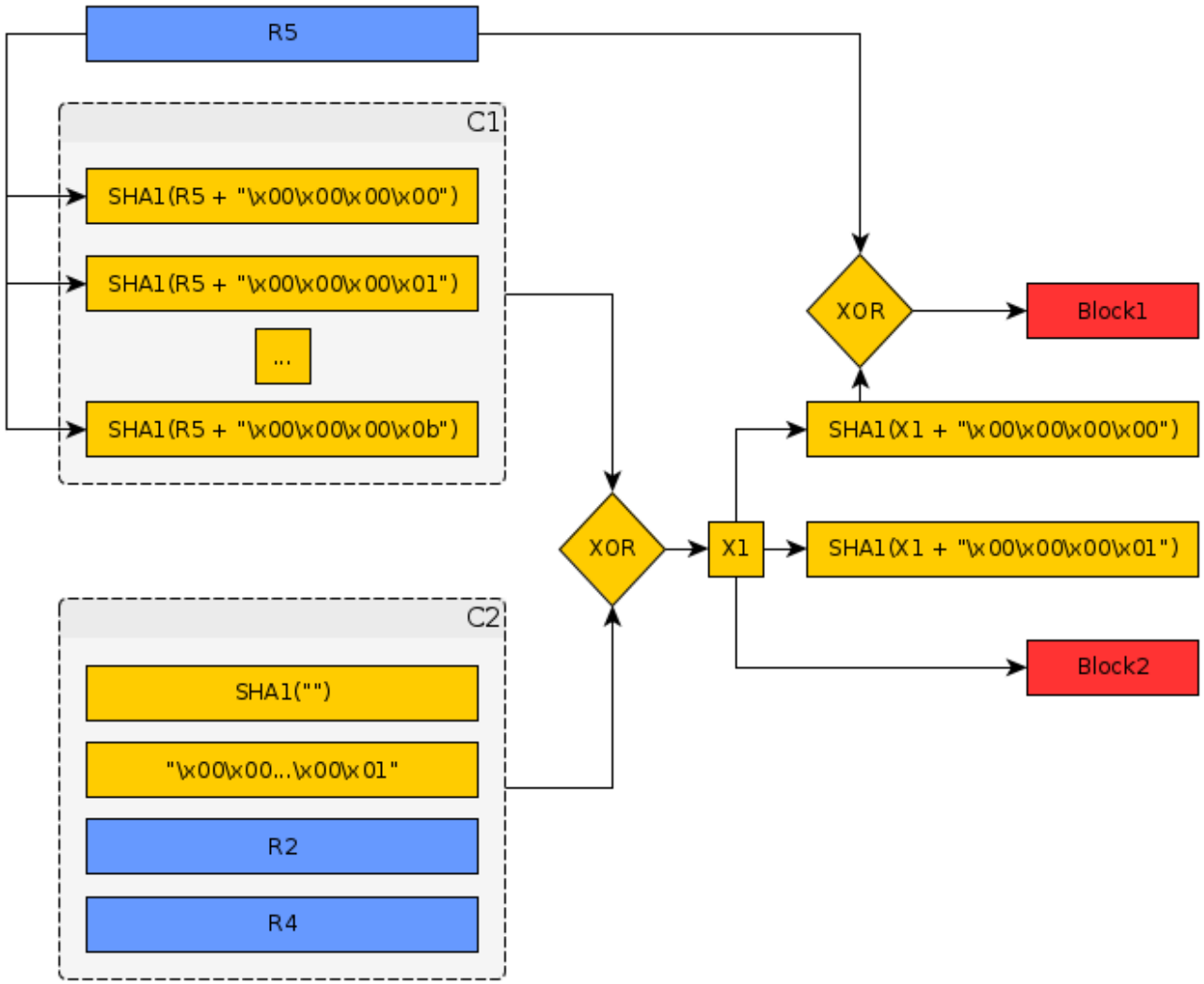
1. Exécution symbolique du basic block de l'élément ciblé
2. Obtention des dépendances de son équation
3. Recherche des dernières écritures de ces éléments
4. Et on remet ça !

Pour le data tainting, la même chose mais en recherchant les éléments dont les dépendances contiennent l'élément ciblé.

Graphe de dépendance



RSA-OAEP



R2, R4, R5 : Valeurs aléatoires

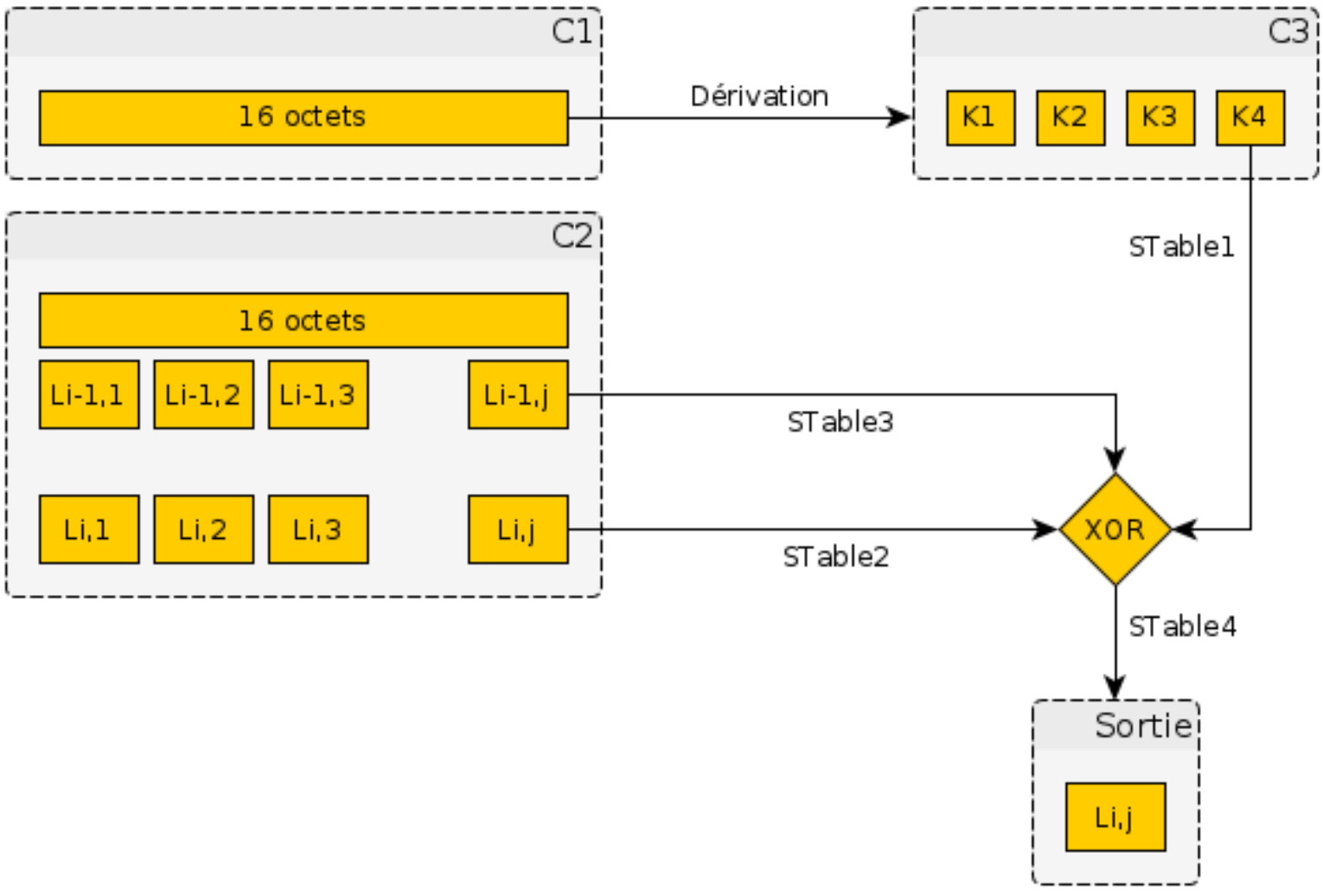
Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
   - Quelques indices
   - Identification d'une WhiteBox AES-CBC dynamique
   - Résultat
5. Le moment écolo : les instructions équivalentes
6. Bonus

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
   - Quelques indices
   - Identification d'une WhiteBox AES-CBC dynamique
   - Résultat
5. Le moment écolo : les instructions équivalentes
6. Bonus

Graphe de dépendance



Classe d'équivalence

Proposition de classe d'équivalence

Les données d1 et d2 sont équivalentes si et seulement si leur première lecture est faite par la même instruction. Deux instructions sont considérées comme étant les mêmes si et seulement si elles partagent la même adresse.

Classe d'équivalence

Proposition de classe d'équivalence

Les données d1 et d2 sont équivalentes si et seulement si leur première lecture est faite par la même instruction. Deux instructions sont considérées comme étant les mêmes si et seulement si elles partagent la même adresse.

Exemple

Classe:	01	02	03	04	01	02	03	04	01	02	03	04	05
Donnée:	63	66	F5	F3	76	DC	B1	C1	F6	BC	4D	21	7E

Classe d'équivalence

Proposition de classe d'équivalence

Les données d1 et d2 sont équivalentes si et seulement si leur première lecture est faite par la même instruction. Deux instructions sont considérées comme étant les mêmes si et seulement si elles partagent la même adresse.

Exemple

Classe:	01	02	03	04	01	02	03	04	01	02	03	04	05
Donnée:	63	66	F5	F3	76	DC	B1	C1	F6	BC	4D	21	7E

Regroupement

```
63  66  F5  F3
76  DC  B1  C1
F6  BC  4D  21
                7E
```

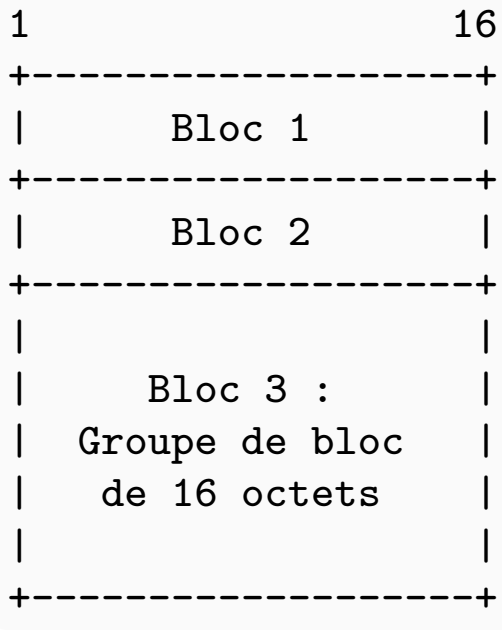
Classe d'équivalence

Application au jeu de données

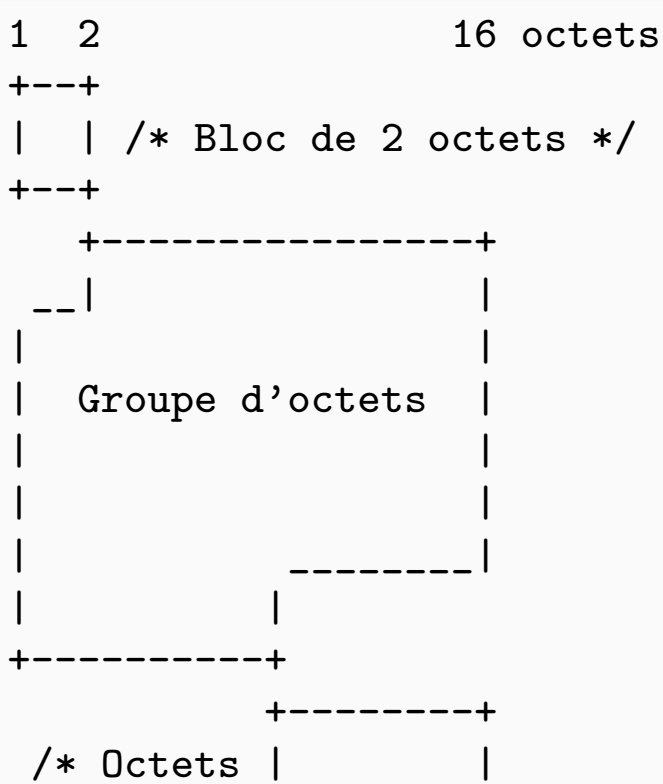
```
1                   16 octets
+-------------------+
|      Bloc 1       |
+-------------------+
|      Bloc 2       |
+-------------------+
|                   |
|     Bloc 3 :      |
|  Groupe de bloc   |
|   de 16 octets    |
|                   |
+-------------------+
```

Classe d'équivalence

Application au jeu de données



Application au bloc en sortie (sens inverse, dernière écriture)



Reconstruction de fonction

```
 1  def make_C3(inp):
 2
 3          C3 = [inp]
 4          for i in xrange(10):
 5                  tmp = []
 6                  tmp.append(inp[0] ^ table1[(0x100*i)+inp[13]])
 7                  tmp.append(inp[1] ^ table2[inp[14]])
 8                  tmp.append(inp[2] ^ table2[inp[15]])
 9                  tmp.append(inp[3] ^ table2[inp[12]])
10                  tmp.append(inp[4] ^ tmp[0])
11                  tmp.append(inp[5] ^ tmp[1])
12                  tmp.append(inp[6] ^ tmp[2])
13                  tmp.append(inp[7] ^ tmp[3])
14                  tmp.append(inp[8] ^ tmp[4])
15                  tmp.append(inp[9] ^ tmp[5])
16                  tmp.append(inp[10] ^ tmp[6])
17                  tmp.append(inp[11] ^ tmp[7])
18                  tmp.append(inp[12] ^ tmp[8])
19                  tmp.append(inp[13] ^ tmp[9])
20                  tmp.append(inp[14] ^ tmp[10])
21                  tmp.append(inp[15] ^ tmp[11])
22                  C3.append(tmp)
23                  inp = tmp
24
25          return C3
```

Comparaison `make_c3` et dérivation de clé AES

```
def make_C3(self,inp):

        C3 = [inp]
        for i in range(10):
                tmp = []
                tmp.append(inp[0] ^ table1[(0x100*i)+inp[13]])
                tmp.append(inp[1] ^ table2[inp[14]])
                tmp.append(inp[2] ^ table2[inp[15]])
                tmp.append(inp[3] ^ table2[inp[12]])
                tmp.append(inp[4] ^ tmp[0])
                tmp.append(inp[5] ^ tmp[1])
                tmp.append(inp[6] ^ tmp[2])
                tmp.append(inp[7] ^ tmp[3])
                tmp.append(inp[8] ^ tmp[4])
                tmp.append(inp[9] ^ tmp[5])
                tmp.append(inp[10] ^ tmp[6])
                tmp.append(inp[11] ^ tmp[7])
                tmp.append(inp[12] ^ tmp[8])
                tmp.append(inp[13] ^ tmp[9])
                tmp.append(inp[14] ^ tmp[10])
                tmp.append(inp[15] ^ tmp[11])
                C3.append(tmp)
                inp = tmp

        return C3
```

AES Key expansion

```
for size in range(expandedKeySize):

        for k in range(4):
                word[k] = expandedKey[(size - 4) + k]

        if size % sizeKey == 0:

                word = rotate(word)
                for i in range(4):
                        word[i] = getSBoxValue(word[i])

                word[0] = word[0] ^ getRconValue(rconIteration)

                rconIteration += 1;

        for m in range(4):
                expandedKey[size] = expandedKey[size - sizeKey] ^ t[m]
                size += 1
```

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
   - Quelques indices
   - Identification d'une WhiteBox AES-CBC dynamique
   - Résultat
5. Le moment écolo : les instructions équivalentes
6. Bonus

Identification d'une WhiteBox AES-CBC dynamique

Identification

- Tentative de reproduction des entrées / sorties
- $\Rightarrow$ Les résultats ne correspondent pas
- $\Rightarrow$ Les étapes de chiffrement sont entièrement effectuées sur des états dérivés, la clef est aussi en paramètre
- $\Rightarrow$ WhiteBox " dynamique "

Identification d'une WhiteBox AES-CBC dynamique

Identification

- Tentative de reproduction des entrées / sorties
- $\Rightarrow$ Les résultats ne correspondent pas
- $\Rightarrow$ Les étapes de chiffrement sont entièrement effectuées sur des états dérivés, la clef est aussi en paramètre
- $\Rightarrow$ WhiteBox " dynamique "

Intérêt dans une DRM

- Une perte de temps pour l'analyste;
- L'impossibilité de connaître ni l'entrée ni la sortie;
- L'impossibilité de reproduire l'algorithme sur une autre plate-forme (dans le cas de l'interopérabilité, par exemple);
- L'impossibilité d'utiliser l'algorithme inverse s'il ne nous est pas fourni.

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
   - Quelques indices
   - Identification d'une WhiteBox AES-CBC dynamique
   - Résultat
5. Le moment écolo : les instructions équivalentes
6. Bonus

Résultat

Attaque

1. Algorithme homomorphique à l'opération XOR
2. Propriétés mathématiques imposées
3. Nombre de candidats restreint

$\Rightarrow$ Calcul des fonctions de dérivation

On arrive finalement à pouvoir lire/modifier les valeurs chiffrées par l'algorithme, un AES-CBC 128 bits.

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
   - Présentation
   - Version industrielle
6. Bonus

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
   - Présentation
   - Version industrielle
6. Bonus

Instruction équivalente - Base

Méthode basique

Pour $x \in [0, 2^{32} - 1]$:

$$f(x) = (16 * x + 16) mod 2^{32}$$

pourrait être réécrite :

$f(x) = 129441535 - 1793574399 * (1584987567 * (3781768432 * x + 2881946191) - 4282621936)$

Instruction équivalente - Base

Méthode basique

Pour $x \in [0, 2^{32} - 1]$:

$$f(x) = (16 * x + 16) mod 2^{32}$$

pourrait être réécrite :

$f(x) = 129441535 - 1793574399 * (1584987567 * (3781768432 * x + 2881946191) - 4282621936)$

Simplification

Fonction simplifiée par les passes de compilation modernes (en particulier la propagation de constantes)

Instruction équivalente - Avancée

MBA : Mixed Boolean Arithmetic

En mixant des transformations logiques et arithmétiques :

$$(x + y) \equiv ((x \wedge y) + (x \vee y))$$

$$(x + y) \equiv ((x \oplus y) + 2 \times (x \wedge y))$$

$$(x \oplus y) - y \equiv (x \wedge \not{y}) - (x \wedge y)$$

Instruction équivalente - Avancée

MBA : Mixed Boolean Arithmetic

En mixant des transformations logiques et arithmétiques :

$$(x + y) \equiv ((x \wedge y) + (x \vee y))$$

$$(x + y) \equiv ((x \oplus y) + 2 \times (x \wedge y))$$

$$(x \oplus y) - y \equiv (x \wedge \not{y}) - (x \wedge y)$$

Simplification

- Non simplifiées par les passes des compilateurs
- Ni par MatLab, Maple, Mathematica ou encore Z3

Instruction équivalente - Avancée

MBA : Mixed Boolean Arithmetic

En mixant des transformations logiques et arithmétiques :

$$(x + y) \equiv ((x \wedge y) + (x \vee y))$$

$$(x + y) \equiv ((x \oplus y) + 2 \times (x \wedge y))$$

$$(x \oplus y) - y \equiv (x \wedge \not{y}) - (x \wedge y)$$

Simplification

- Non simplifiées par les passes des compilateurs
- Ni par MatLab, Maple, Mathematica ou encore Z3

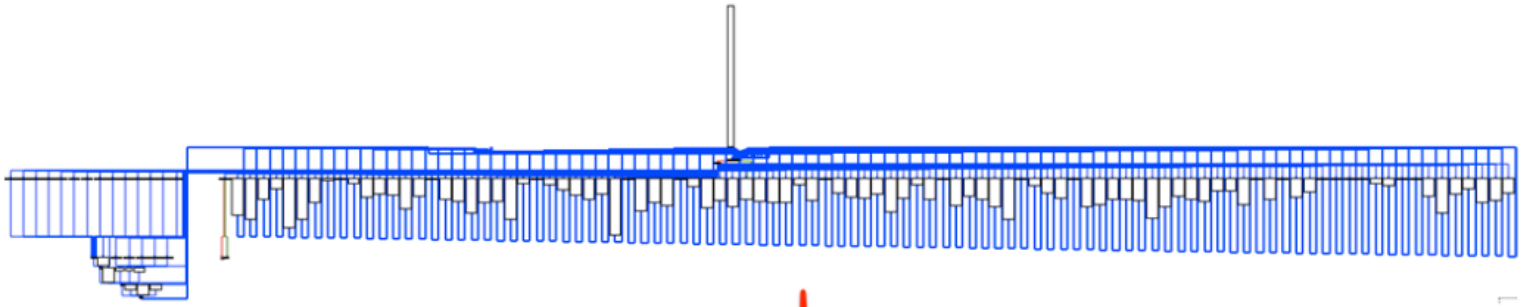
Simplification effective

- Lorsque les équations sont identifiées, capitalisation via le moteur de simplification de Miasm
- En utilisant l'algorithme de génération de ces expressions

Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
   - Présentation
   - Version industrielle
6. Bonus

Équation de transfert de la fonction à analyser



```
int f(int x) {
    result = (0xed*((((((((((((((((((((((- ((((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26
0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFF
F)*0x2))+0xFF)&0xFE)+(((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*
x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF))*0xE587A503)
0xB717A54D)*0xAD17DB56)+0x60BA9824)&0xFFFFFF46)*0xA57C144B)+(((((((- ((((((((0xe5*x + 0xF7)&0xFF + ( 0x0 <<
 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&
0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0x2))+0xFF)&0xFE)+(((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*
0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 <<
8)&0xFFFFFFFF))*0xE587A503)+0xB717A54D)*0xE09C02E7)+0xB5ED2776)((((((((((- ((((((((0xe5*x + 0xF7)&0xFF +
( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6
)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0x2))+0xFF)&0xFE)+(((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF
)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 <<
 8)&0xFFFFFFFF))*0xE587A503)+0xB717A54D)*0xAD17DB56)+0x60BA9824)&0xFFFFFF46)*0xA57C144B)+(((((((-
((((((((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF +
( 0x0 << 8)&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0x2))+0xFF)&0xFE)+(((((((0xe5*x +
0xF7)&0xFF + ( 0x0 << 8)&0xFFFFFFFF)*0xFFFFFE26)+0x55)&0xFE)+(((0xe5*x + 0xF7)&0xFF + ( 0x0 << 8)
&0xFFFFFFFF)*0xED)+0xD6)&0xFF&0xFF + ( 0x0 << 8)&0xFFFFFFFF))*0xE587A503) ...
    return result;
}
```

Identification des variables puis de la fonction d'origine, ici un XOR 0x5C

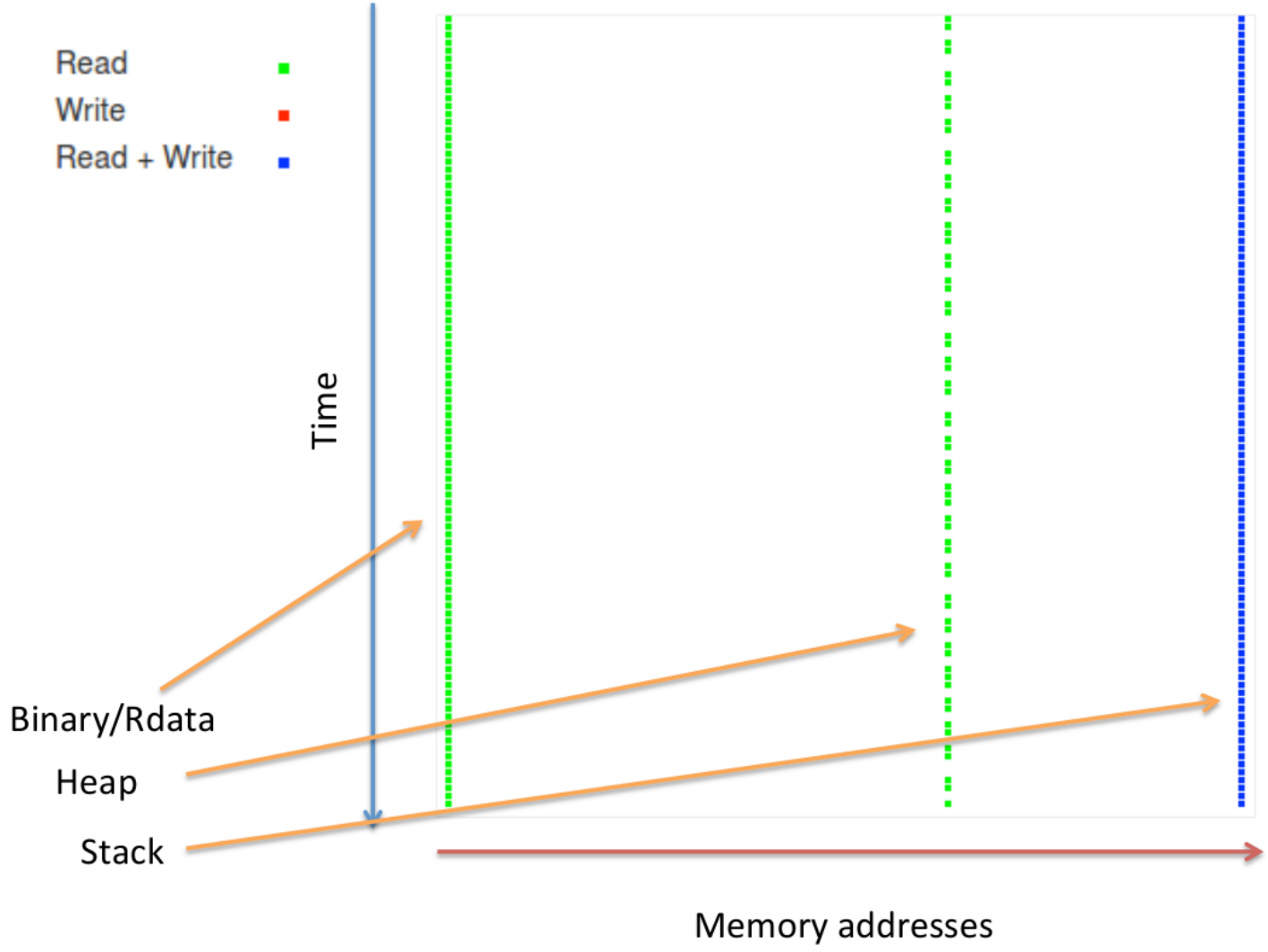
```
int f(int x) {
    x = (0xe5*x + 0xF7) % 0x100;
    v1 = 0x0;
    v2 = 0xFE;
    v0 = (x&0xFF + ( v1 << 8)&0xFFFFFFFF);
    v3 = (((((v0*0xFFFFFE26)+0x55)&v2)+(v0*0xED)+0xD6)&0xFF&0xFF + ( v1 << 8)&0xFFFFFFFF);
    v4 = ((((((- (v3*0x2))+0xFF)&v2)+v3)*0xE587A503)+0xB717A54D);
    v5 = (((((v4*0xAD17DB56)+0x60BA9824)&0xFFFFFF46)*0xA57C144B)+(v4*0xE09C02E7)+0xB5ED2776);
    v7 = ((((v5*0xC463D53A)+0x3C8878AF)&0xCC44B4F4)+(v5*0x1DCE1563)+0xFB99692E);
    v6 = (v7&0x94);
    v8 = ((((v6+v6+(- (v7&0xFF&0xFF + ( v1 << 8)&0xFFFFFFFF)))*0x67000000)+0xD000000) >> 0x18);
    result = ((v8*0xFFFFB22D)+(((v8*0xAE)|0x22)*0xE5)+0xC2)&0xFF & 0xFFFFFFFF;
    result = (0xed*(result-0xF7)) % 0x100;

    return result;
}
```

```
int f(int x) {
    return (x & 0xFF) ^ 0x5C;
}
```

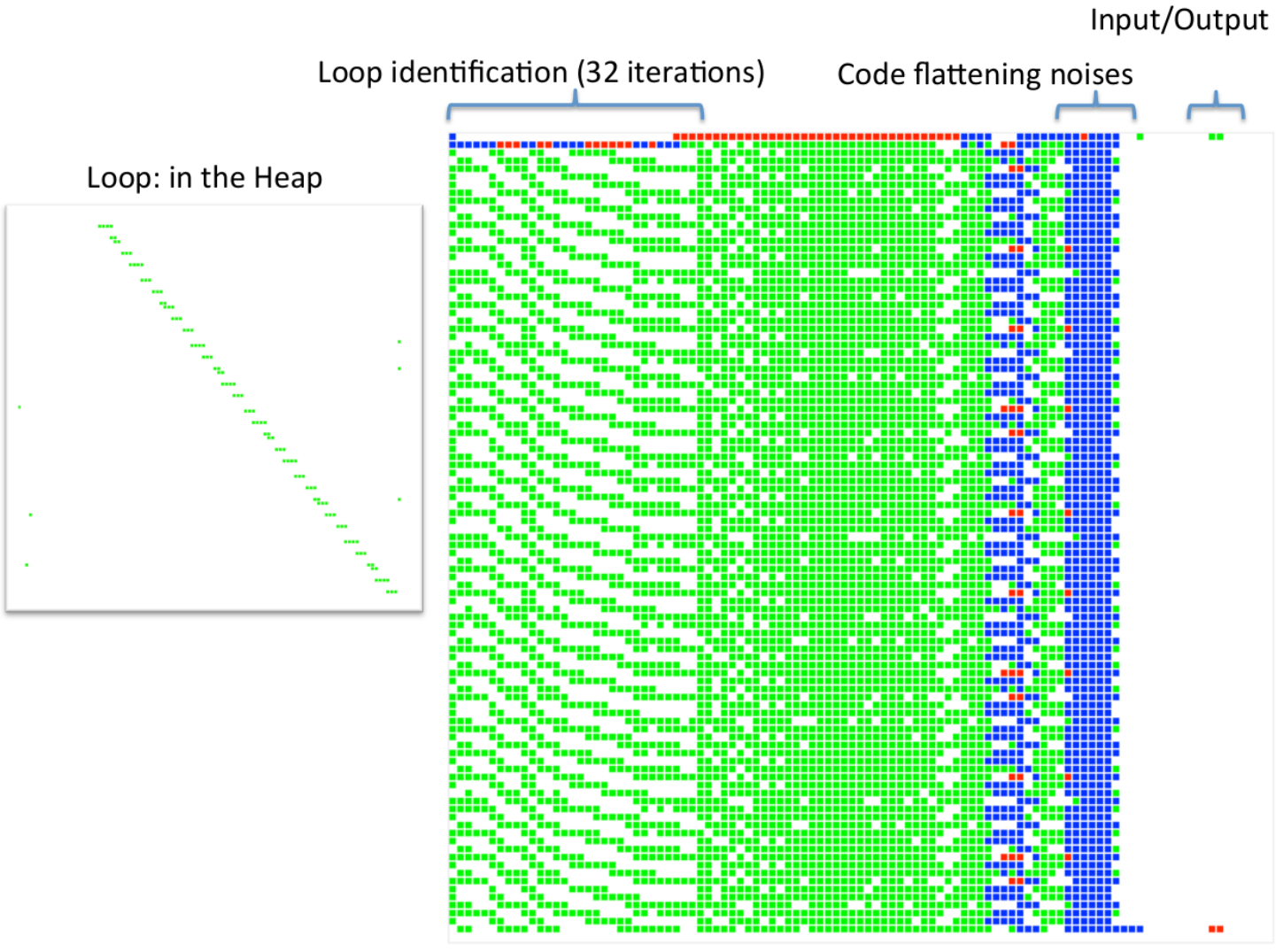
Plan

1. Première couche : Code flattening
2. pTra
3. Reconstruction d'un algorithme : RSA-OAEP
4. Reconstruction d'une fonction de chiffrement "whiteboxée" : AES-CBC
5. Le moment écolo : les instructions équivalentes
6. Bonus

Représentation des accès mémoire en fonction du temps



Agrandissement de la pile, détection de boucles



So . . .

Conclusion

Intérêt de l'approche

- A permis d'analyser une DRM à l'état de l'art de l'obscurcissement
- Outil de plus dans la panoplie de l'analyste
- Peut aussi être utilisée en analyse de code malveillant, recherche de vulnérabilités, ...

Conclusion

Intérêt de l'approche

- A permis d'analyser une DRM à l'état de l'art de l'obscurcissement
- Outil de plus dans la panoplie de l'analyste
- Peut aussi être utilisée en analyse de code malveillant, recherche de vulnérabilités, ...

L'obscurcissement

- Sujet d'actualité, de plus en plus élaboré
- Initiative publique O-LLVM, encore trop jeune
- Les appareils, même mobiles, ont suffisamment de ressources pour se permettre d'en gâcher une grande partie

Conclusion

Intérêt de l'approche

- A permis d'analyser une DRM à l'état de l'art de l'obscurcissement
- Outil de plus dans la panoplie de l'analyste
- Peut aussi être utilisée en analyse de code malveillant, recherche de vulnérabilités, ...

L'obscurcissement

- Sujet d'actualité, de plus en plus élaboré
- Initiative publique O-LLVM, encore trop jeune
- Les appareils, même mobiles, ont suffisamment de ressources pour se permettre d'en gâcher une grande partie

Notre approche n'est pas meilleure qu'une autre; elle offre simplement un autre angle d'attaque !

Questions?

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com