DE LA RECHERCHE À L'INDUSTRIE

Sibyl

Function divination tool

Camille Mougey

Sommaire

Principe

« Dans ce binaire / firmware / malware / shellcode / …, la fonction à l'adresse 0x1234 est un `memcpy` »

Principe

Approche statique

- FLIRT
- Polichombr, Gorille, BASS
- Machine learning (ASM considéré comme NLP)
- *Bit-precise Symbolic Loop Mapping*
- …

Approche dynamique / trace

- Entropie des données dans les boucles
- Avalanche de propagation de teinte

Approche proche de Sibyl

- Angr "identifier"[a] ≈ PoC pour le CGC

a. https ://github.com/angr/identifier

Principe

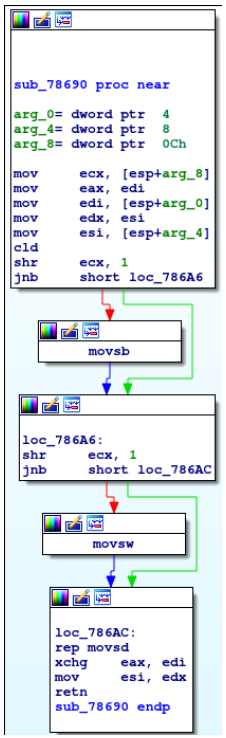


Figure – memcpy « naif »

Principe

Problème

Comment reconnaître quand optimisé / vectorisé / compilateur différent / **obscurci** ?

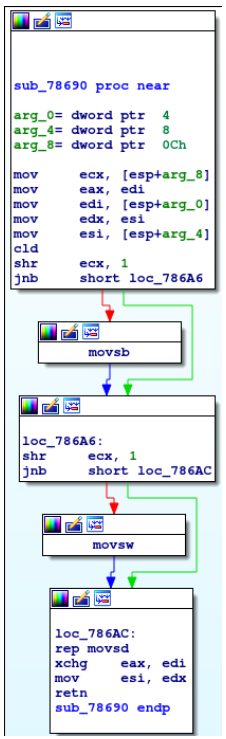


Figure – memcpy « naif »

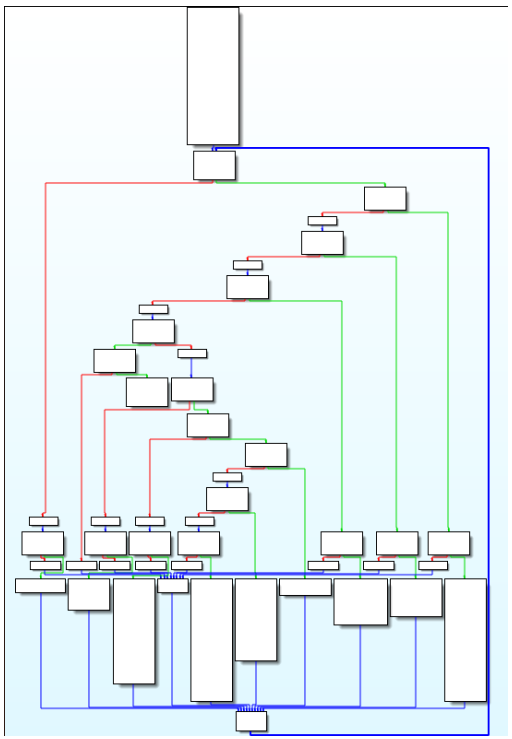


Figure – memcpy « obscurci »

Principe

Problème

Comment reconnaître quand optimisé / **vectorisé** / compilateur différent / obscurci ?

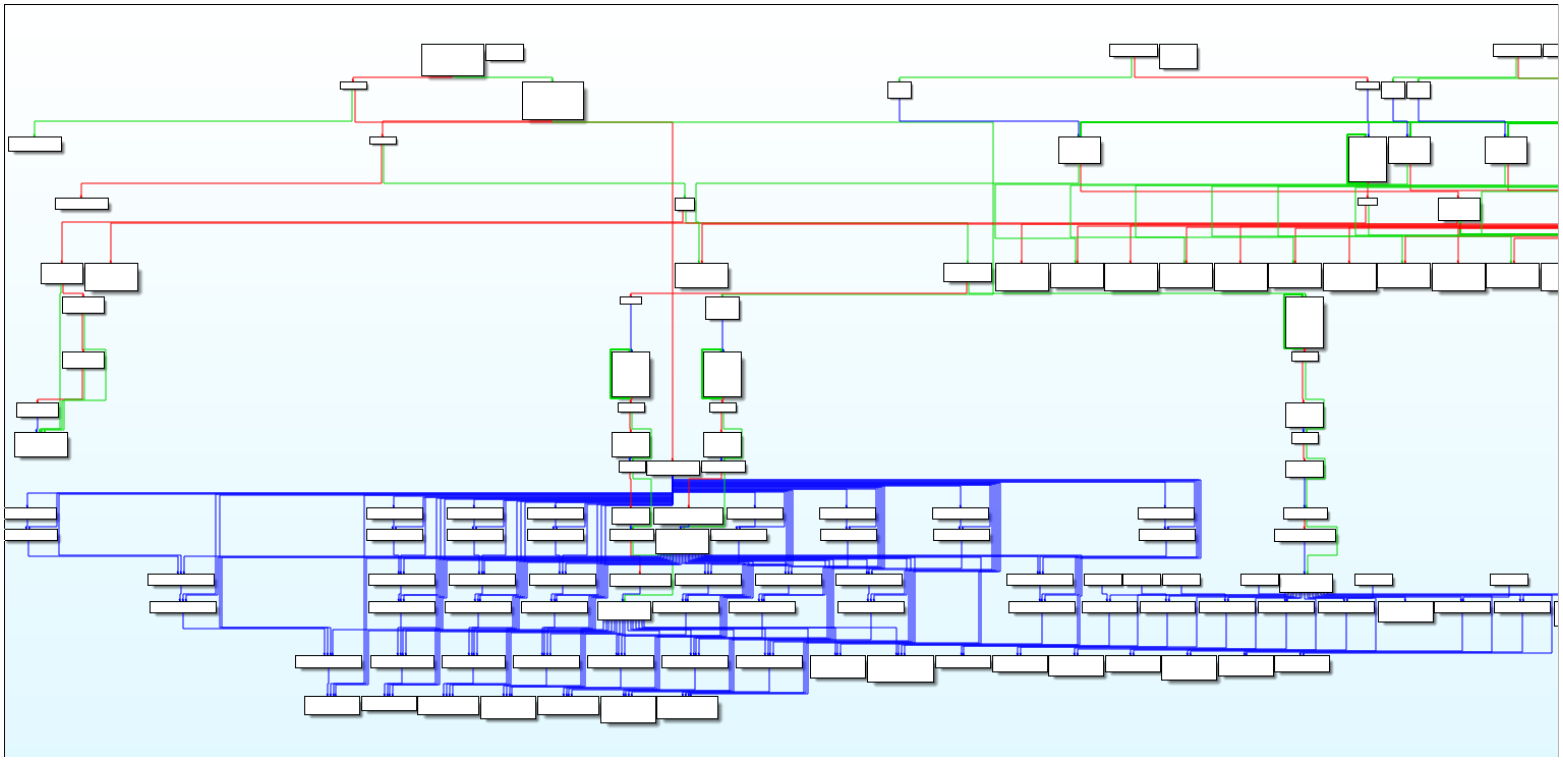


Figure – `memcpy` « SSE »

Principe

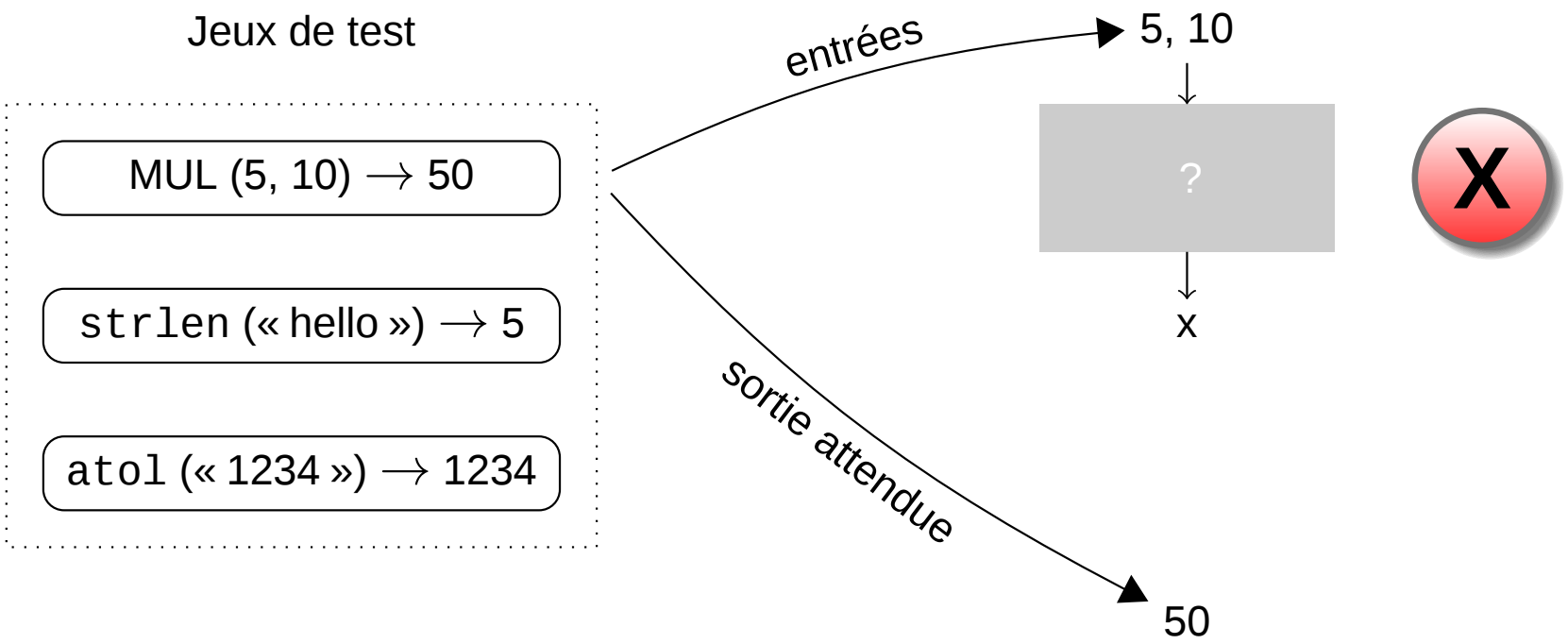
Idée

- Fonction = boîte noire
- Entrées choisies
- Sorties observées $\leftrightarrow$ Sorties attendues

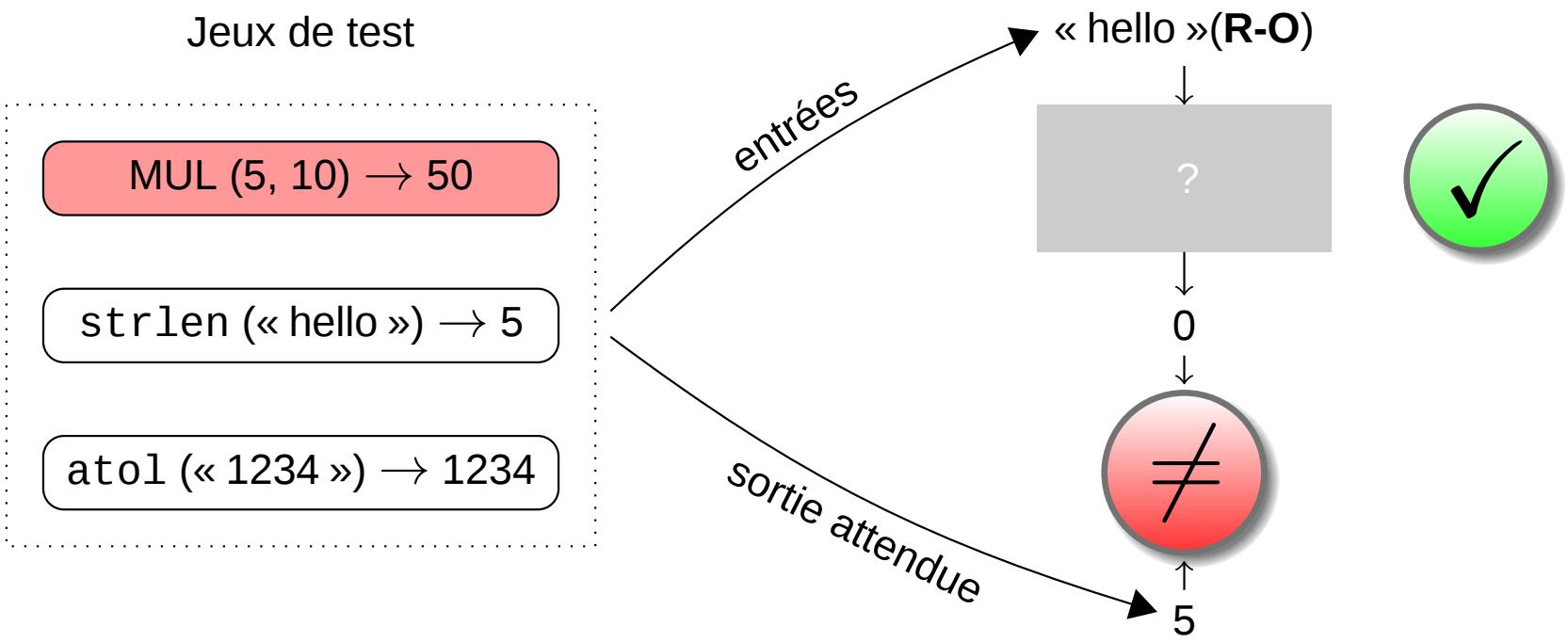
Spécifiquement

- Entrées = $\{$ arguments, mémoire initiale $\}$
- Sorties = $\{$ valeur de retour, mémoire finale $\}$
- Environnement minimaliste : $\{$ binaire mappé, stack $\}$

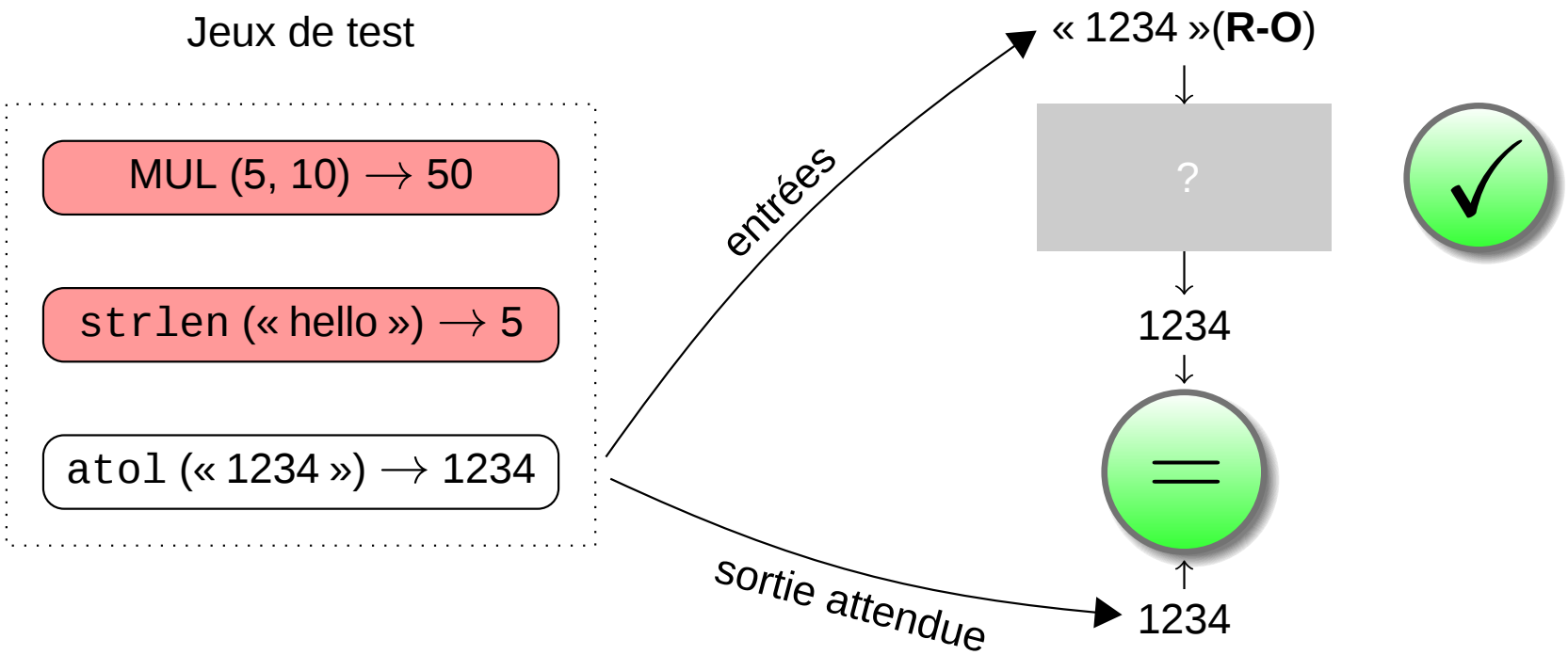
Principe


Jeux de test
MUL (5, 10) ⟶ 50
strlen (« hello ») ⟶ 5
atol (« 1234 ») ⟶ 1234
entrées
5, 10
?
x
X
sortie attendue
50

Principe


Jeux de test
MUL (5, 10) → 50
strlen (« hello ») → 5
atol (« 1234 ») → 1234
entrées
« hello »(R-O)
?
0
sortie attendue
5

Principe


Jeux de test
MUL (5, 10) → 50
strlen (« hello ») → 5
atol (« 1234 ») → 1234
entrées
« 1234 »(R-O)
?
1234
sortie attendue
1234

Principe

Jeux de test

MUL (5, 10) $\longrightarrow$ 50

`strlen` (« hello ») $\longrightarrow$ 5

`atol` (« 1234 ») $\longrightarrow$ 1234

`atol`

Principe

Le diable est dans les détails

- Fonctions attendant un *pointeur*, on donne l'entier 2
  - $\rightarrow$ Résistance aux crashs

Principe

Le diable est dans les détails

- Fonctions attendant un *pointeur*, on donne l'entier 2
  - $\rightarrow$ Résistance aux crashs
- Fonctions attendent un *compteur*, on donne un *pointeur*
  - $\rightarrow$ Détection des trèèèès longues boucles / temps d'exécution

Principe

Le diable est dans les détails

- Fonctions attendant un *pointeur*, on donne l'entier 2
  - $\rightarrow$ Résistance aux crashs
- Fonctions attendent un *compteur*, on donne un *pointeur*
  - $\rightarrow$ Détection des trèèèès longues boucles / temps d'exécution
- Un appel peut ne pas être suffisant
  - (2, 2) $\rightarrow$ `Func` $\rightarrow$ 4
  - `add`, `mul`, `pow` ?
  - $\rightarrow$ Politique de test : « test1 & (test2 || test3) »

Principe

Et tout le reste

- Pas d'interférence entre les tests
- *Embarrassingly parallel*
- Éviter les pertes de temps (instanciation de classe, stack d'appel, …)
- Description de test « architecture agnostique », « ABI agnostique »
- …

→ et si on en faisait un outil ?

Principe

Sibyl

- Open-source, GPL
- Version actuelle : 0.2
- CLI + Plugin IDA
- `/doc`
- D’autres contributeurs, notamment F. Desclaux & P. Graux

`https://github.com/cea-sec/Sibyl`

« Une sibylle est une « prophétesse », une femme qui fait œuvre de divination. »

Principe


```
# Dependencies
$> sudo apt install python-pyparsing python-dev

# Elfesteem
$> git clone github.com/serpilliere/elfesteem && cd elfesteem && \
   sudo python setup.py install

# Miasm
$> git clone github.com/cea-sec/miasm && cd miasm && \
   sudo python setup.py install

# Sibyl
$> git clone github.com/cea-sec/Sibyl && cd Sibyl && \
   sudo python setup.py install
```

Démonstration

Démonstration

- Présentation de la CLI
- Configuration
- Exemple : firmware
- Exemple : fonction inconnue optimisée, via IDA

Ajout d'un test

Base de signature

- Située dans `sibyl/test/*`
- Voir `doc/ADD_TESTS.md`

strlen
strnicmp
strcpy
strncpy
strcat
strncat
strcmp
strchr
strrchr
strnlen
strspn
strpbrk
strtok

strsep
memset
memmove
stricmp
strrev
memcmp
bzero
strncmp
memcpy
abs
a64l
atoi
isalnum

isalpha
isascii
isblank
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit

Ajout d'un test

Création de la classe Python représentant le test

```
1 from sibyl.test.test import Test, TestSet
2
3 class TestStrlen(Test):
4     func = "strlen"
```

Ajout d'un test

Préparation du premier test :

Allocation d'une chaîne de caractère en **read-only**

```
5       # Test1
6       my_string = "Hello, w%srld !"
7
8       def init(self):
9           # Alloue une chaine en READ-only
10          self.my_addr = self._alloc_string(self.my_string)
11
12          # Passe le pointeur dans le premier argument
13          self._add_arg(0, self.my_addr)
```

Ajout d'un test

Vérification du résultat

```
14      def check(self):
15          # Vérifie le résultat attendu
16          return self._get_result() == len(self.my_string)
```

Ajout d'un test

Second test pour éviter les fonctions retournant toujours 15 (faux positif trivial)

```
17      # Test2 : évite les fonctions qui retourne toujours 15
18      def init2(self):
19          self.my_addr = self._alloc_string(self.my_string * 4)
20          self._add_arg(0, self.my_addr)
21
22      def check2(self):
23          return result == len(self.my_string * 4)
```

Ajout d'un test

« Politique »de test

```
24      # Politique de test : test1 & test2
25      tests = TestSetTest(init , check) & TestSetTest(init2 , check2)
```

Ajout d'un test


```
1  class TestStrlen(Test):
2      func = "strlen"
3
4      # Test1
5      my_string = "Hello, w%srld !"
6
7      def init(self):
8          # Alloue une chaine en READ-only
9          self.my_addr = self._alloc_string(self.my_string)
10
11         # Passe le pointeur dans le premier argument
12         self._add_arg(0, self.my_addr)
13
14     def check(self):
15         # Vérifie le résultat attendu
16         return self._get_result() == len(self.my_string)
17
18     # Test2 : évite les fonctions qui retourne toujours 15
19     def init2(self):
20         self.my_addr = self._alloc_string(self.my_string * 4)
21         self._add_arg(0, self.my_addr)
22
23     def check2(self):
24         return result == len(self.my_string * 4)
25
26     # Politique de test : test1 & test2
27     tests = TestSetTest(init, check) & TestSetTest(init2, check2)
```

Focus sur l'apprentissage

Ajout de nouveaux tests

- « à la main »pour chaque test
- assez rapide, mais ingrat
- compliqué à l'échelle d'une bibliothèque

→ Apprentissage automatique !

Focus sur l'apprentissage

Version naive

1. Prendre un *snapshot* du contexte (mémoire + registres) *avant*
2. Faire tourner la fonction
3. Prendre un *snapshot* mémoire *après*

Focus sur l'apprentissage

Mais …

- « bruit »lié à l'implémentation
  - variable locale
  - constantes du binaire (par ex : SBOX AES)
- différent *packing* de structure
- changement d'architecture
- changement de convention d'appel (registre $\rightarrow$ stack)
- …

Focus sur l'apprentissage

Pré-requis

- Un binaire exécutant *au moins une fois* la fonction
  - Tests de regression
  - Malware
  - Binaire « maison »pour l'occasion

Focus sur l'apprentissage

Pré-requis

- Un binaire exécutant *au moins une fois* la fonction
  - Tests de regression
  - Malware
  - Binaire « maison »pour l'occasion
- Signature *complète* de la fonction
  - Comprend le type des arguments, et structures associées
  - Obtenu des sources
  - Ou d'un reverse partiel de la fonction

Focus sur l'apprentissage

Méthode

1. Trace d'exécution de la fonction
2. Élagage des runs trop semblables (*branch coverage*)
3. Rejeu en *Dynamic Symbolic Execution* (DSE) avec symbolisation des arguments
4. Re-concretisation des symboles (non-pointeur) pour éviter l'explosion combinatoire
5. Traduction en *C-like* des accès mémoires
6. Génération du test Python

Focus sur l'apprentissage

Exemple : OpenSSL SHA1_Update

- Binaire : `sha1test`
- Header : `openssl/sha.h`

```
1  typedef struct SHAstate_st {
2      unsigned int h0, h1, h2, h3, h4;
3      unsigned int Nl, Nh;
4      unsigned int data[16];
5      unsigned int num;
6  } SHA_CTX;
7
8  typedef int size_t;
9
10 int SHA1_Update(SHA_CTX *c, const void* data, size_t len);
```

Focus sur l'apprentissage

Méthode

- Trace d'exécution

```
Début de fonction
RAX: 8000000000000000
RBX: 1
RCX: 7ffd9b689a40
...
Lecture en 7ffd9b689a40, 32 bits, valeur 0x12345678
...
Fin de fonction
```

Focus sur l'apprentissage

Méthode

- Rejeu en *Dynamic Symbolic Execution* (DSE) avec symbolisation des arguments

```
Lecture en @[arg0_c + 65], valeur 0x62
...
Écriture en @[arg0_c], valeur 0x1
...
```

Focus sur l'apprentissage

Méthode

- Traduction en *C-like* des accès mémoires

```
Lecture en arg0_c->data[1], valeur 0x62
...
Écriture en arg0_c->Nh, valeur 0x1
...
```

Focus sur l'apprentissage

Méthode

- Génération du test Python

```
1  class TestSHA1_Update(TestHeader):
2      '''This is an auto-generated class, using the Sibyl learn module'''
3      func = ``SHA1_Update''
4
5      def init1(self):
6          # arg0_c
7          base0_ptr_size = self.field_addr("arg0_c", "(arg0_c)->num") + \
8                           self.sizeof("(arg0_c)->num")
9          base0_ptr = self._alloc_mem(base0_ptr_size, read=True, write=True)
10         ...
11         # (arg1_data)[1]
12         input0_ptr = base1_ptr + self.field_addr("arg1_data", "(arg1_data)[1]")
13         ...
14         self._add_arg(0, base0_ptr) # arg0_c
15         ...
16         # *(arg1_data) = 0x61
17         self._write_mem(base1_ptr, self.pack(0x61, self.sizeof("*(arg1_data)")))
18         # (arg1_data)[1] = 0x62
19         self._write_mem(input0_ptr, self.pack(0x62, self.sizeof("(arg1_data)[1]")))
20         ...
```

Focus sur l'apprentissage

Méthode

- Génération du test Python

```
1   def check1(self):
2       return all((
3           # Check output value
4           self._get_result() == 0x1,
5           # (arg0_c)->Nh == 0x0
6           self._ensure_mem(self.input4_ptr, self.pack(0x0, self.sizeof("(arg0_c)->Nh"))),
7           # (arg0_c)->Nl == 0x18
8           self._ensure_mem(self.input1_ptr, self.pack(0x18, self.sizeof("(arg0_c)->Nl"))),
9           # *((arg0_c)->data) == 0x636261 (without considering 0x3 offset(s))
10          self._ensure_mem_sparse(self.output0_ptr,
11                                  self.pack(0x636261,
12                                            self.sizeof("*((arg0_c)->data)")),
13                                  [0x3]),
14          # (arg0_c)->num == 0x3
15          self._ensure_mem(self.input3_ptr, self.pack(0x3, self.sizeof("(arg0_c)->num"))),
16      ))
17  ...
18  TestSetTest(init1, check1) & TestSetTest(init2, check2) & ...
```

Focus sur l'apprentissage

Démonstration

- Apprentissage de SoftFp [a] (bibliothèque de softfloat)
- Recherche dans un binaire obscurci (data-flow + control-flow) utilisant cette bibliothèque

a. SoftFp, release du 20-12-2016 : `https://bellard.org/softfp/softfp-2016-12-20.tar.gz`

Limites et futur

En théorie

Sibyl

- est *complet* : signature connue + fonction présente $\longrightarrow$ fonction trouvée
- n'est pas *correct* : possibilité de faux positif [a]

a. problème impossible dans le cas général sans parcourir l'espace des entrées

Limites et futur

Limite en pratique

- Erreur du moteur d’émulation
- Erreur dans l’écriture du test
- Mauvaise ABI

Limites et futur

Limites en pratique

- Appel de fonction externes / `malloc`
  - PoC, en cours d'implementation !

Limites et futur

Limites en pratique

- Appel de fonction externes / `malloc`
  - PoC, en cours d'implementation !
- Fonction non apprenable
  - arithmétique de pointeur ($ptr < 0x7fffffff$)
  - $(((ptr \oplus 0x7fffffff) \& (ptr \oplus (ptr + (-0x7fffffff)))) \oplus (ptr + (-0x7fffffff))) \oplus ptr \oplus 0x7fffffff).msb$

Limites et futur

Limites en pratique

- Appel de fonction externes / `malloc`
  - PoC, en cours d'implementation !
- Fonction non apprenable
  - arithmétique de pointeur ($ptr < 0x7fffffff$)
  - $(((ptr \oplus 0x7fffffff) \& (ptr \oplus (ptr + (-0x7fffffff)))) \oplus (ptr + (-0x7fffffff))) \oplus ptr \oplus 0x7fffffff).msb$
- Il faut que ce soit une fonction connue de Sibyl !

Limites et futur

Futur

- Plus de test !
  - Test sur de nouvelles bibliothèques
  - Complétion des tests d’apprentissage (mutants)

Limites et futur

Futur

- Plus de test !
  - Test sur de nouvelles bibliothèques
  - Complétion des tests d'apprentissage (mutants)
- Plus robuste !
  - Amélioration des capacités d'apprentissage
  - Gestion des appels aux fonctions externes

Limites et futur

Futur

- Plus de test !
  - Test sur de nouvelles bibliothèques
  - Complétion des tests d'apprentissage (mutants)
- Plus robuste !
  - Amélioration des capacités d'apprentissage
  - Gestion des appels aux fonctions externes
- Plus de fonctionnalités !
  - Détection automatique de l'ABI
  - Détection des addresses de fonctions
  - Mode « pourquoi j'ai tort ? »

Limites et futur

Sibyl

- `github.com/cea-sec/Sibyl`
- README.MD#Basic usage
- `/doc`
- Issue + PR !

Limites et futur

Commissariat à l'énergie atomique et aux énergies alternatives CEA/DAM
Centre de Bruyères-le-Châtel | 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 | F. +33 (0)1 69 26 40 00
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019