Obfuscation d'applications écrites en Python

python-pack

Ninon EYROLLES
neyrolles@quarkslab.com

Serge GUELTON
sguelton@quarkslab.com

QUARKSLAB
INNOVATIVE SECURITY

Qu'est-ce que l'obfuscation ?

Qu'est-ce que l'obfuscation ?

Pourquoi s'intéresser au langage Python ?

Le cas Dropbox

Client léger écrit en Python, « packé » et obfusqué.

« Packer » Python

- Embarque dans un seul binaire l'interpréteur, les modules dépendants (Python et natifs) et les données
- Utilise le concept de *frozen modules*

Défis scientifiques et techniques

- Obfuscations traditionnelles difficiles à mettre en œuvre
- Beaucoup de mélange interprété / natif
- bytecode facilitant la rétro - ingénierie
- Beaucoup de fonctionnalités d'introspection
- ...

Python et analyse statique

Liaison retardée

Analyse d'*aliasing* très complexe

```
1  b = list()     # vers quoi pointe list ?
2  a = range(n)   # vers quoi pointe range ? n ?
3  c = b or a     # vers quoi pointe c ?
```

Réflectivité

Possibilité de faire dépendre le flot de donnée de la structure du programme...

Évaluation dynamique

$\Rightarrow$ mot-clef `exec` et fonction intrinsèque `eval`

Surface d'attaque

cf. D. Kholia, P. Wegrzyn, *Looking inside the (Drop) box* in WOOT'13

Bibliothèques dynamiques

`LD_PRELOAD` pour rentrer facilement dans l'espace mémoire du processus et appeler une fonction de décodage.

API de CPython

⇒ Utilisation de `PyRun_SimpleString` pour exécuter du code externe.

Comparaison de bytecode

⇒ *Reverse* de la permutation d'*opcodes* par comparaison avec le bytecode des modules standards.

Décompilation

Des outils existent : `uncompyle` ou (mieux !) `pycdc`.

Python-Pack

README

Python-pack est un *packer* obfusquant pour Python développé à QUARKSLAB

Python2.7

L'interpréteur cible. Plusieurs transformations sont sensibles à la version...

Transformations

À trois niveaux :

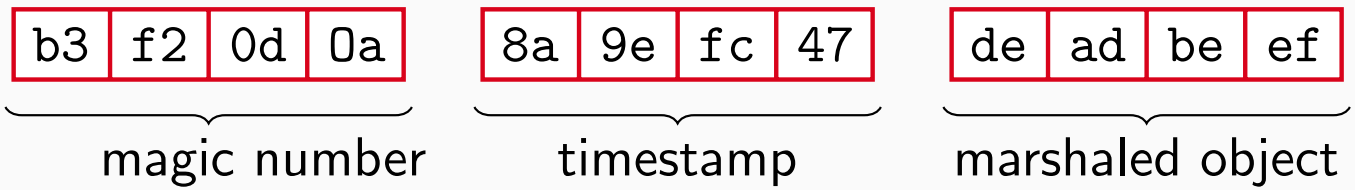
1. Source Python (source-à-source)
2. Modification de l'interpréteur
3. Hybride : modification conjointe des sources et de l'interpréteur

Écrit en Python *mais ne bootstrap pas encore... à cause des ressources externes (.c, .diff)*

Modifications de l'interpréteur

Magic Number

Un fichier Python :



Suppression de fonctionnalités

- Attribut `co_code`
- Import de modules non gelés

Ajout de *junk bytecode*

Émission d'un *bytecode* différent de l'interpréteur standard pour une opération donnée (manipulation de pile à l'aide de `ROT_TWO`...)
⇒ Perturbe (= fait *crash*er) le décompilateur `uncompyle`.

Modifications des *opcodes*

Permutations du jeu d'instruction

Effectuer une permutation sur les *opcodes* du jeu d'instruction pour rendre le *bytecode* plus ardu à comprendre.
e.g. `ROT_TWO` $\Rightarrow$ `DUP_TOP`

34	LOAD_GLOBAL
35	CALL_FUNCTION
36	POP_TOP

34 $\rightarrow$ 75
35 $\rightarrow$ 23
36 $\rightarrow$ 12
$\Rightarrow$

75	LOAD_FAST
23	LOAD_CONST
12	ROT_TWO

Ajout de nouveaux *opcodes*

Remplacement de séquences d'*opcodes* par un nouvel *opcode*.

LOAD_GLOBAL
CALL_FUNCTION
POP_TOP

$\Rightarrow$

LOAD_GLOBAL
CALL_AND_POP

Modification du bytecode à la volée

Code auto-modifiant

Modification du byte code en cours d'exécution en *hack*ant l'API C de Python

```
1   def foo(b):
2       b += 1
3       ...
```

```
1   def foo(b):
2       modify_bytecode()
3       b -= 1
4       ...
```

À l'exécution

LOAD_GLOBAL
CALL_FUNCTION
POP_TOP
LOAD_FAST
LOAD_CONST
INPLACE_SUB

Modification du bytecode à la volée

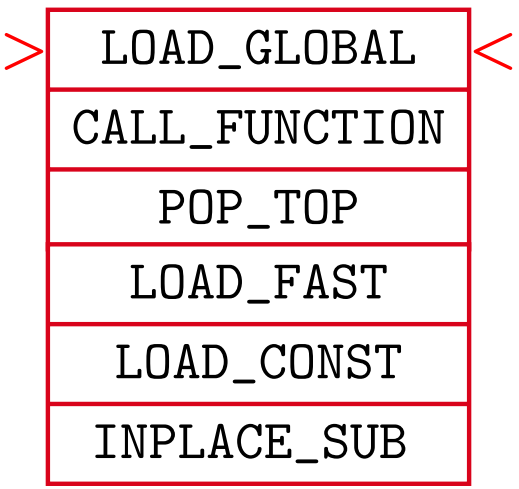
Code auto-modifiant

Modification du byte code en cours d'exécution en *hack*ant l'API C de Python

```
1   def foo(b):
2       b += 1
3       ...
```

```
1   def foo(b):
2       modify_bytecode()
3       b -= 1
4       ...
```

À l'exécution



Modification du bytecode à la volée

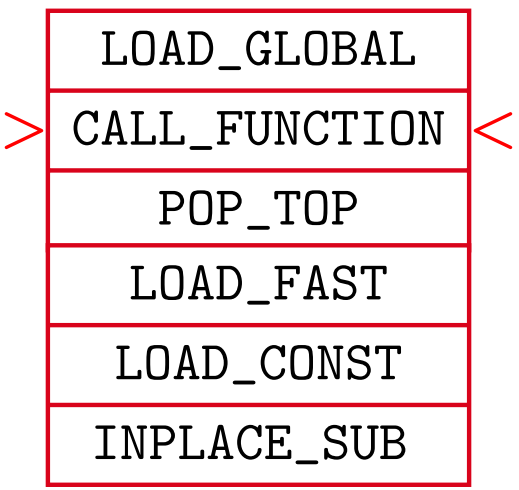
Code auto-modifiant

Modification du byte code en cours d'exécution en *hack*ant l'API C de Python

```
1   def foo(b):
2       b += 1
3       ...
```

```
1   def foo(b):
2       modify_bytecode()
3       b -= 1
4       ...
```

À l'exécution



Modification du bytecode à la volée

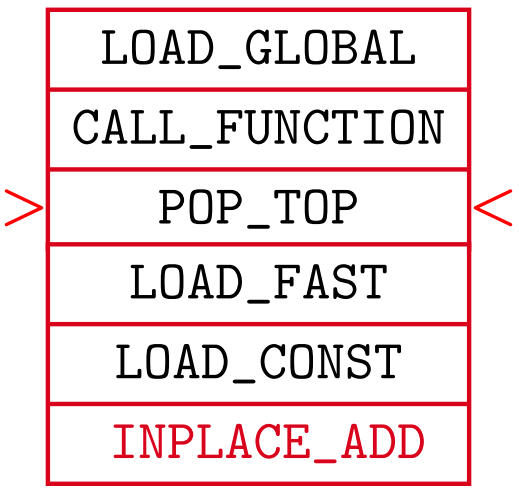
Code auto-modifiant

Modification du byte code en cours d'exécution en *hack*ant l'API C de Python

```
1   def foo(b):
2       b += 1
3       ...
```

```
1   def foo(b):
2       modify_bytecode()
3       b -= 1
4       ...
```

À l'exécution



Modification du bytecode à la volée

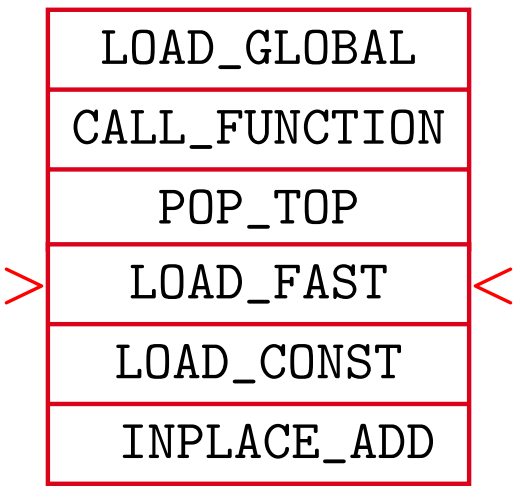
Code auto-modifiant

Modification du byte code en cours d'exécution en *hack*ant l'API C de Python

```
1   def foo(b):
2       b += 1
3       ...
```

```
1   def foo(b):
2       modify_bytecode()
3       b -= 1
4       ...
```

À l'exécution



Modification du bytecode à la volée

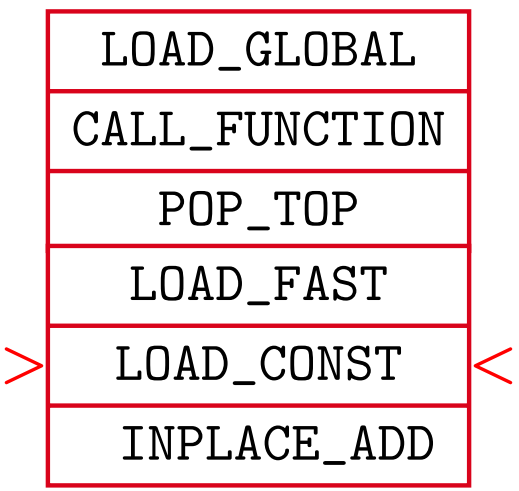
Code auto-modifiant

Modification du byte code en cours d'exécution en *hack*ant l'API C de Python

```
1   def foo(b):
2       b += 1
3       ...
```

```
1   def foo(b):
2       modify_bytecode()
3       b -= 1
4       ...
```

À l'exécution



Modification du bytecode à la volée

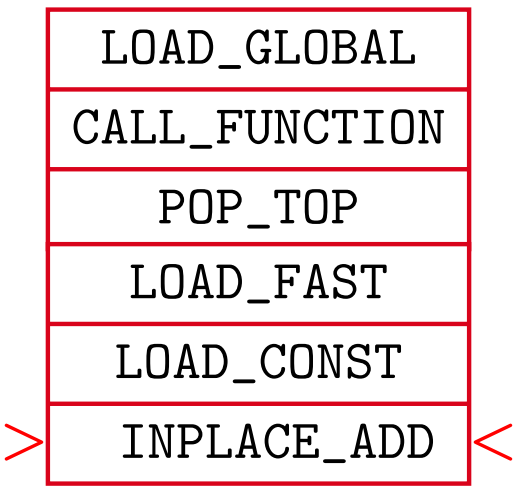
Code auto-modifiant

Modification du byte code en cours d'exécution en *hack*ant l'API C de Python

```
1   def foo(b):
2       b += 1
3       ...
```

```
1   def foo(b):
2       modify_bytecode()
3       b -= 1
4       ...
```

À l'exécution



Obfuscation des données

Chiffrement des chaînes de caractères constantes

- Compilation statique chaîne → chaîne chiffrée
- Modification de l'interpréteur pour déchiffrer les constantes (`LOAD_CONST`)

⇒ Les données n'apparaissent en clair qu'en mémoire.

Obfuscation hybride

Modification conjointe du code source et de l'interpréteur.

Compilation statique

Suppression des chargements dynamiques

- Modules natifs ⇒ embarqués statiquement dans l'interpréteur
- Bibliothèques externes ⇒ liées au binaire
- Modules Python ⇒ gelés puis liés au binaire
- Bibliothèques systèmes ⇒ liées au binaire

Techniquement difficile

Certaines bibliothèques vivent mal la liaison statique : `pthread`, `dl`, `libc`.


10

Empêcher la comparaison de bytecode

Idée

Modifier le code source de la bibliothèque standard pour rendre difficile l'analyse de la permutation d'*opcode*.

Mise en œuvre

Appliquer des transformations source-à-source au code de la bibliothèque standard.
L'objectif n'est pas d'obfusquer mais de générer des opcodes différents

l'obfuscation ne fait pas de mal pour autant...

Obfuscation source-à-source

Principe

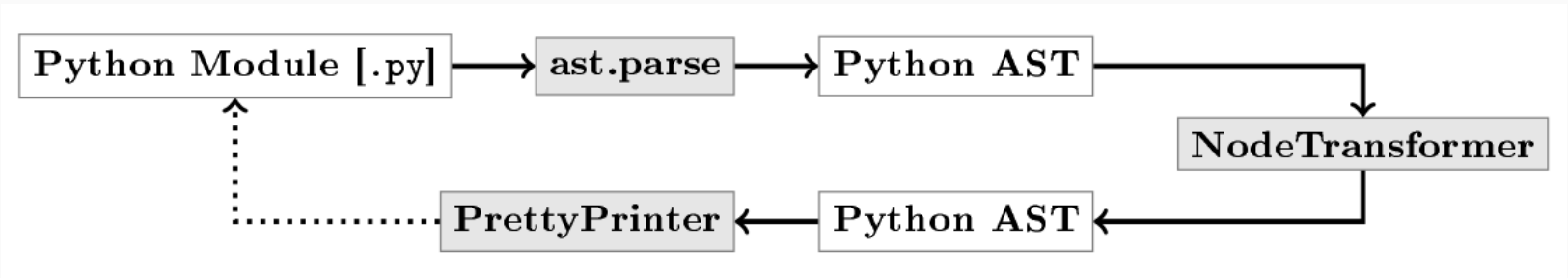


Figure: Flot de compilation pour le source-à-source Python

Modification du flot de contrôle

Déroulage de boucle

Si possible, les boucles `for` sont déroulées, sinon elles sont transformées en boucles `while`.

Transformation des instructions `if` et `while`

Les instructions `if` sont transformées en boucle `while` grâce à l'utilisation de prédicats opaques.

```
1  # original code
2  if cond1:
3      work()
```

```
1  # obfuscated if
2  opaque_pred = 1
3  while opaque_pred & cond1:
4      work()
5      opaque_pred = 0
```

Modifications du flot de contrôle (2)

Traduction automatique en code pseudo-fonctionnel

Inspiré par la sémantique dénotationelle...

```
1  def fibo(n):
2    return n if n < 2 else (fibo(n - 1) + fibo(n - 2))
```

```
1  fibo = (lambda n: (lambda _: (_.__setitem__('$', ((_['n'] if ('
       n' in _) else n) if ((_['n'] if ('n' in _) else n) < 2)
       else ((_['fibo'] if ('fibo' in _) else fibo)(((_['n'] if (
       'n' in _) else n) - 1)) + (_['fibo'] if ('fibo' in _) else
        fibo)(((_['n'] if ('n' in _) else n) - 2))))), _)[(-1)])
       ({'n': n, '$': None})['$'])
```

Modifications des opérateurs binaires

Opérations entières

- *Mixed Boolean Arithmetic* (*cf. Information Hiding in Software with Mixed Boolean-Arithmetic Transforms* de Zhou et al.)
- Nécessite un typage statique

Typage par traçage

1. encapsuler statiquement chaque appel à l'opérateur par un appel qui loggue l'opération
2. lancer l'application sur sa suite de validation
3. conserver uniquement les opérateurs dont les opérandes sont toujours des entiers

Par exemple :

```
1  a + b == a - ~b - 1
```

Validation

Protocole de validation

Après transformation d'un programme déterministe, pour les mêmes entrées, on obtient les mêmes sorties.

Tests unitaires

Chaque phase d'obfuscation est testée sur des μcodes.

Mini scénarii

Cas extraits de `https://wiki.python.org/moin/SimplePrograms`
$\Rightarrow$ Un seul fichier source, dépendances sur `expat`, `csv`...

Cas concret

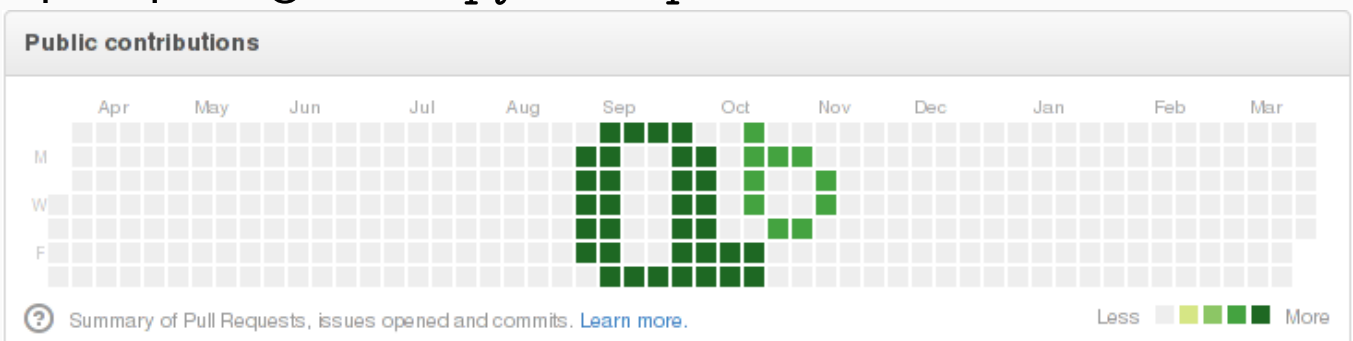
gitfiti

https://github.com/gelstudios/gitfiti

- Un seul fichier
- Quelques dépendances non triviales (urllib2...)
- Démarre plusieurs fils d'exécution

It works!TM

Après passage dans python-pack :



Les maux de la faim

- Génération automatique de *Junk Code*
- Meilleur contrôle de l'API CPython
- Couplage avec un obfuscateur niveau C
- Intégration dans `cxFreeze`

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com