Afl-taenia-mt

Julien Rembinski, Benjamin Dufour, Simon Lebreton, Frédéric Garreau

DGA-MI

May 12, 2020

Index

Mise en situation

Recherche de bugs dans des cibles complexes :

- Environnement Unix
- Boite-noire
- Grande surface de code
- Multi-threadées
- Processus dépendants les uns des autres

Nous avons le besoin d'un *fuzzer* adapté à ces contraintes.

Fuzzing

Le *fuzzing* en 1 minute:

1. Cible
2. Exécution de la cible avec des données semi-aléatoires en entrée
3. Vérification du comportement

Un bon *fuzzer* :

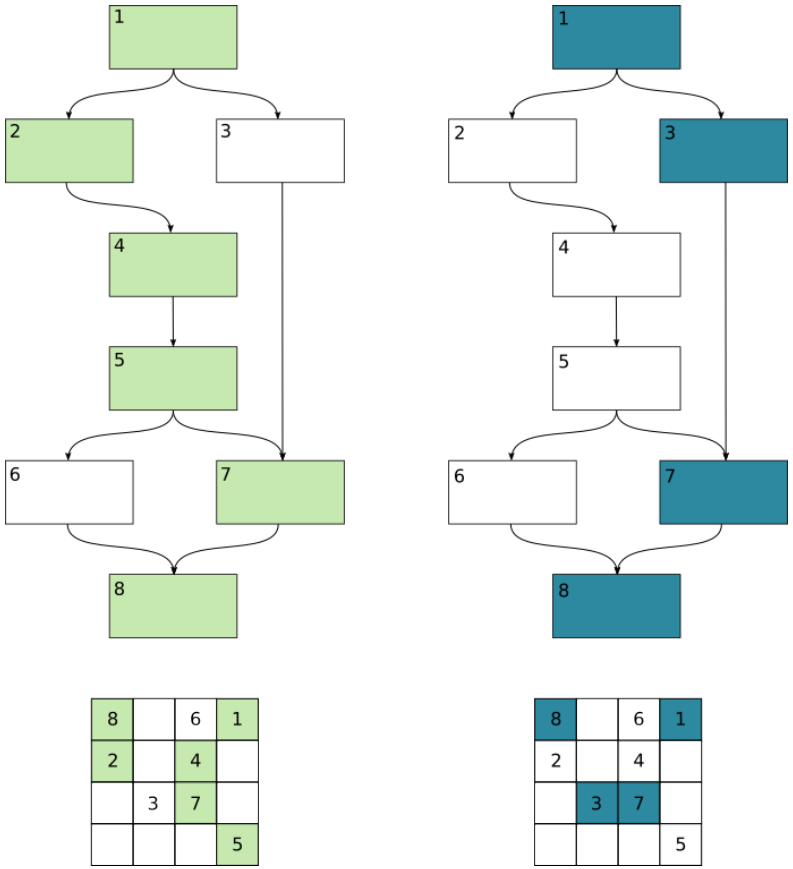
- Rapide
- Malin
- Honnête

afl, afl++, afl-qemu

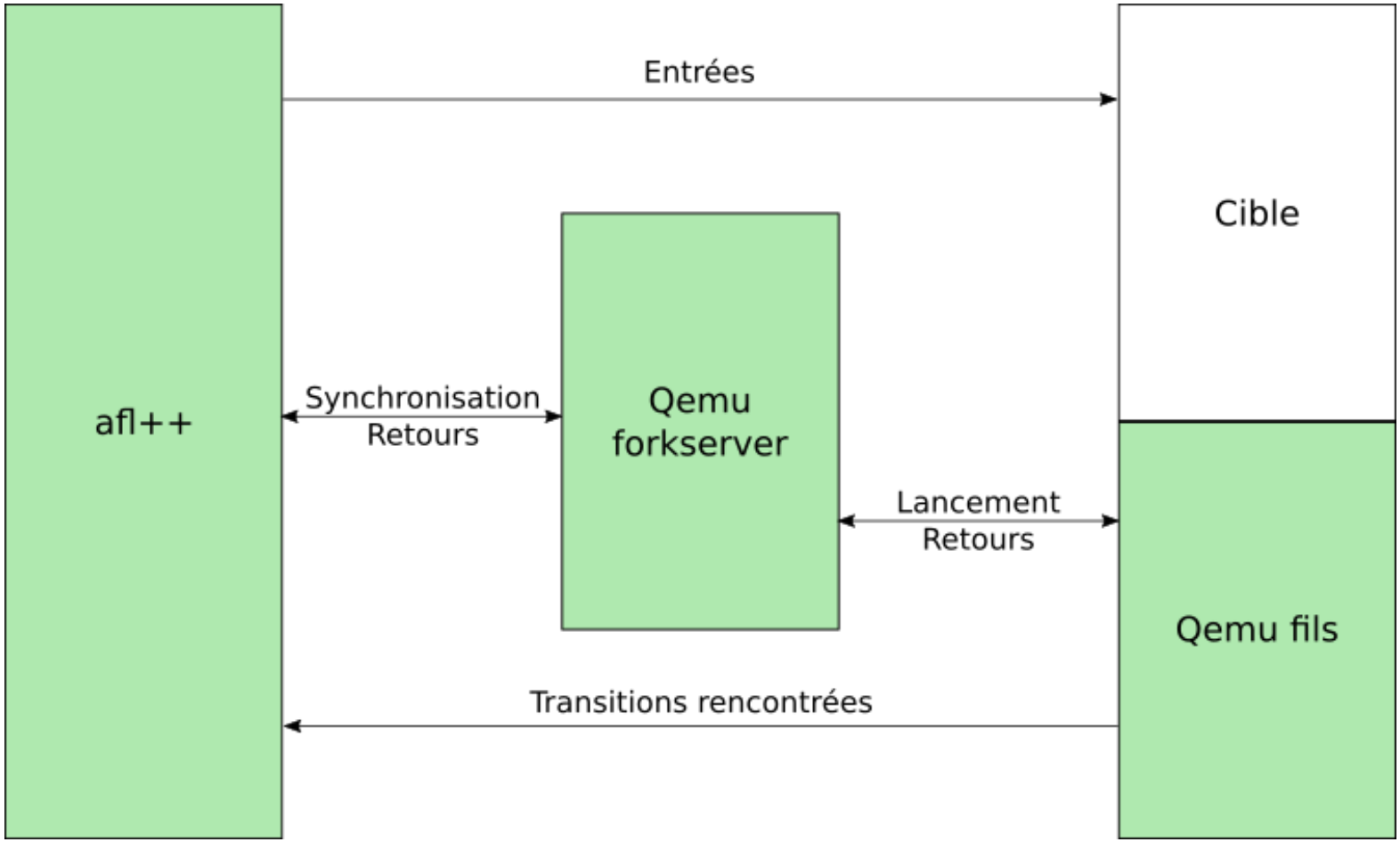
afl: une référence du *fuzzing* :

- Fuzzing par mutation
- Malin : mise en avant des entrées menant à de nouveaux chemins
- Rapide : code plein d'optimisations
- afl++
- afl++-qemu : afl++ pour boite noire

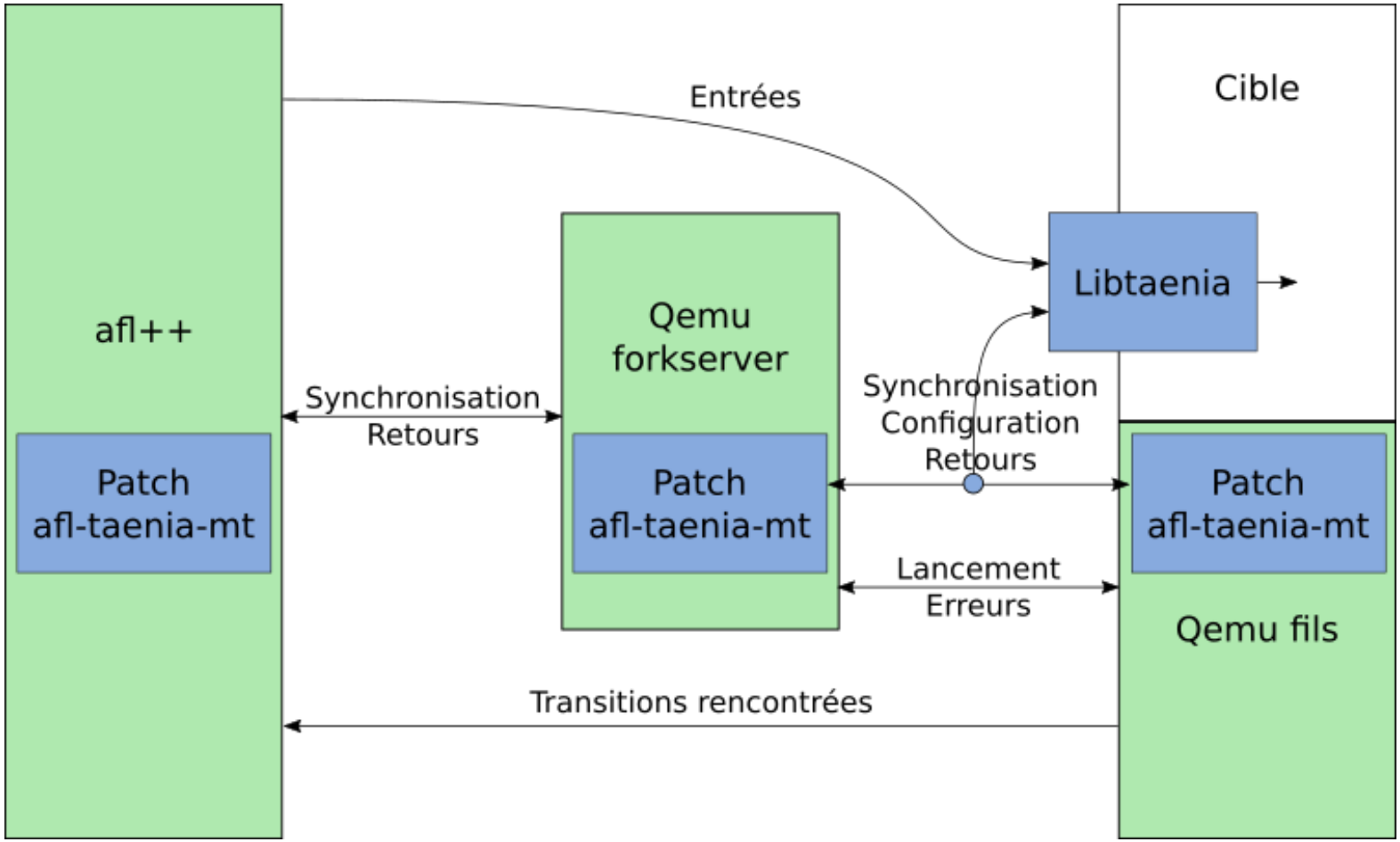
Suivi de graphe de flot de contrôle



afl++-qemu


Entrées
Cible
afl++
Synchronisation
Retours
Qemu
forkserver
Lancement
Retours
Qemu fils
Transitions rencontrées

afl-taenia-mt



Problématique - Fuzzing en mémoire et mode stateful

1. Longue initialisation
2. Réinitialiser le processus cible sans redémarrer l'écosystème = instabilités
3. Erreurs potentielles dans les autres processus de l'écosystème
4. Code ciblé enfoui dans la cible
5. Code ciblé dans bibliothèque externe

Fuzzing en mémoire et sans redémarrage

Injection de libtaenia dans la mémoire de la cible = thread de fuzzing

- Fuzzer n'importe quelle fonction du binaire
- Pas de redémarrage de la cible entre chaque exécution
- Autres threads de la cible continuent leurs traitements et communications inter-processus
- Bugs "stateful"
- Récupération des informations utiles depuis la mémoire du processus

Limites

- La stabilité des processus peut être affectée
- Les autres processus peuvent également être impactés

Mode stateful

- *Fuzzing* par session
- Crash ou timeout = sauvegarde de tous les messages

Rejeu

- Fonctionnalité de rejeu depuis la mémoire
- Capable de rejouer un message ou une session entière

Réduction automatiquement d'une session au plus petit sous-ensemble de messages requis pour déclencher le bug ? WIP !

Suivi de bibliothèques

- libtaenia = mapping mémoire des bibliothèque dans la cible
- Suivi du code des bibliothèques externes qui nous intéressent

Suivi de processus

- Fin d'une session = santé de l'écosystème
- Script pour réinitialiser l'environnement complet

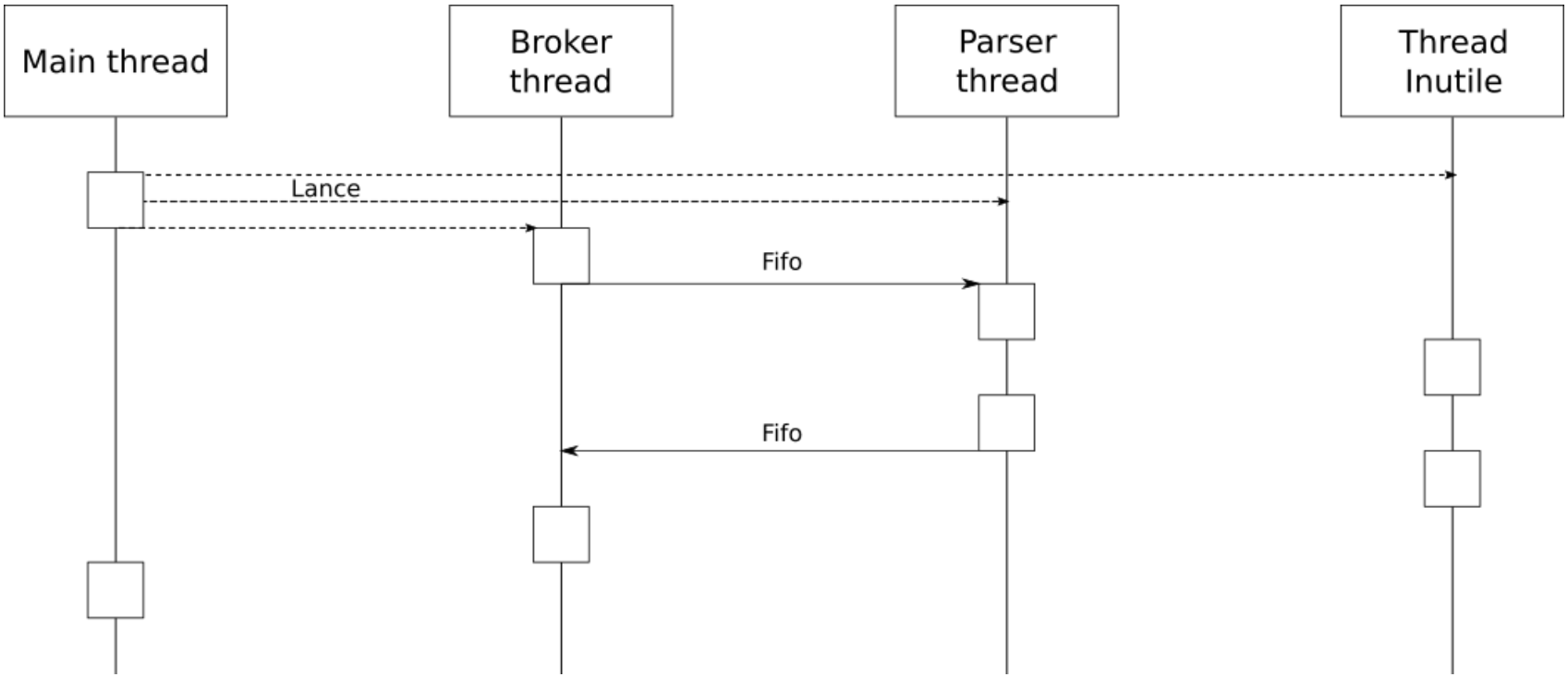
Problématique - Suivi de threads

1. Plusieurs threads manipulant les données.
2. Threads "inutiles" ne manipulant pas les données provenant d'afl.

- Mais afl ne sait pas différencier le code exécuté dans des threads différents.
- Mais afl ne permet pas de les ignorer.

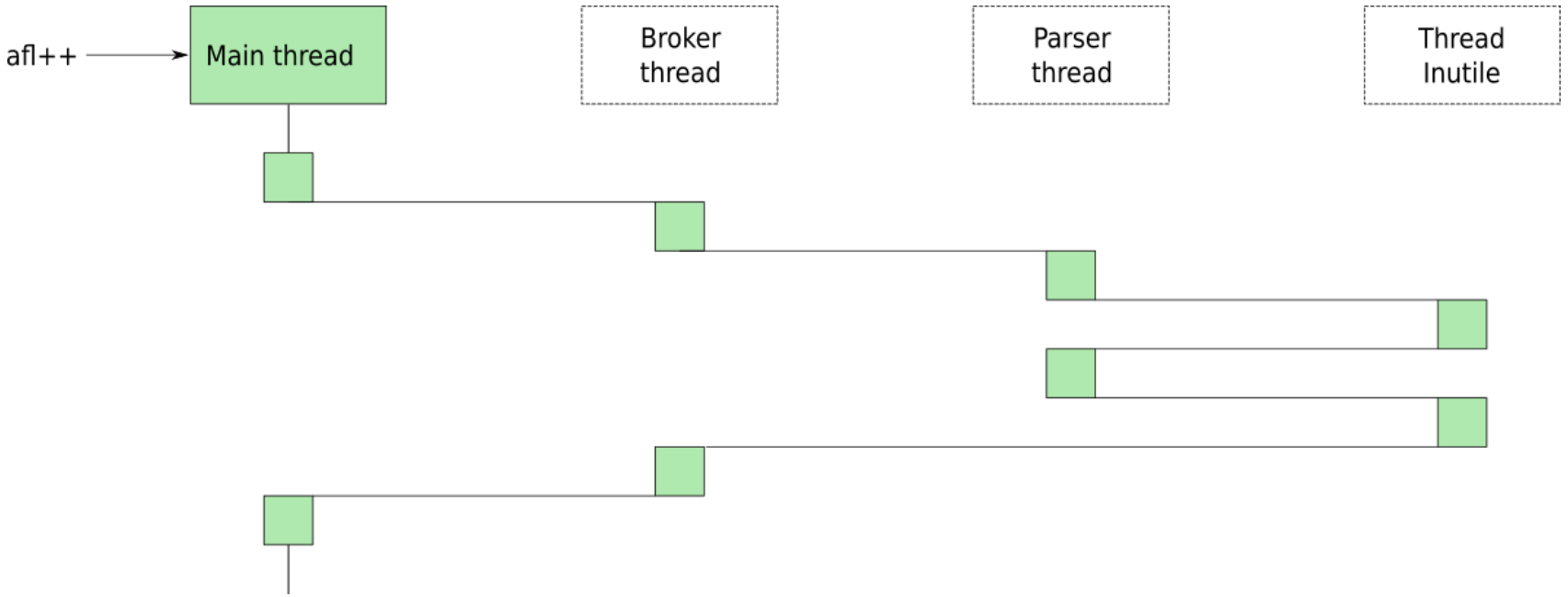
Le multi-thread

Soit l'exemple suivant composé de quatre *threads*.



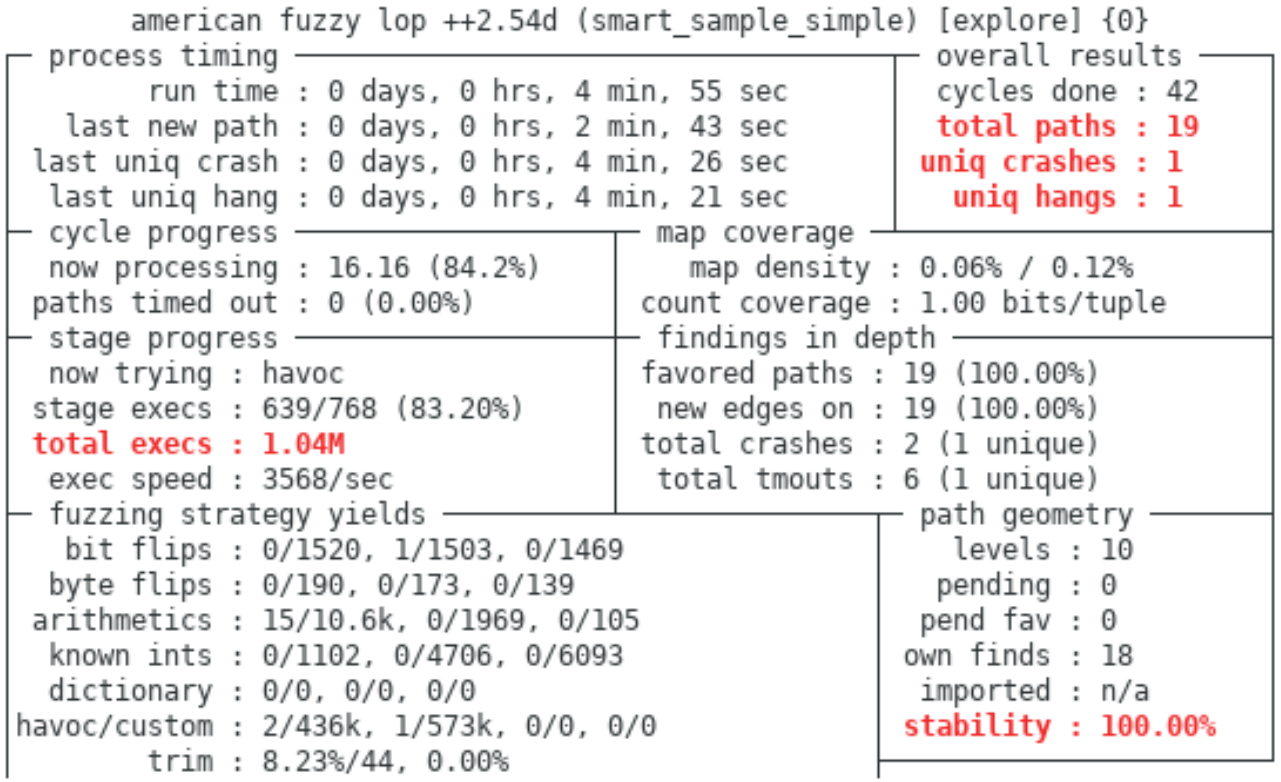
Afl et le multi-thread

Afl le traitera comme un processus *mono-thread*.



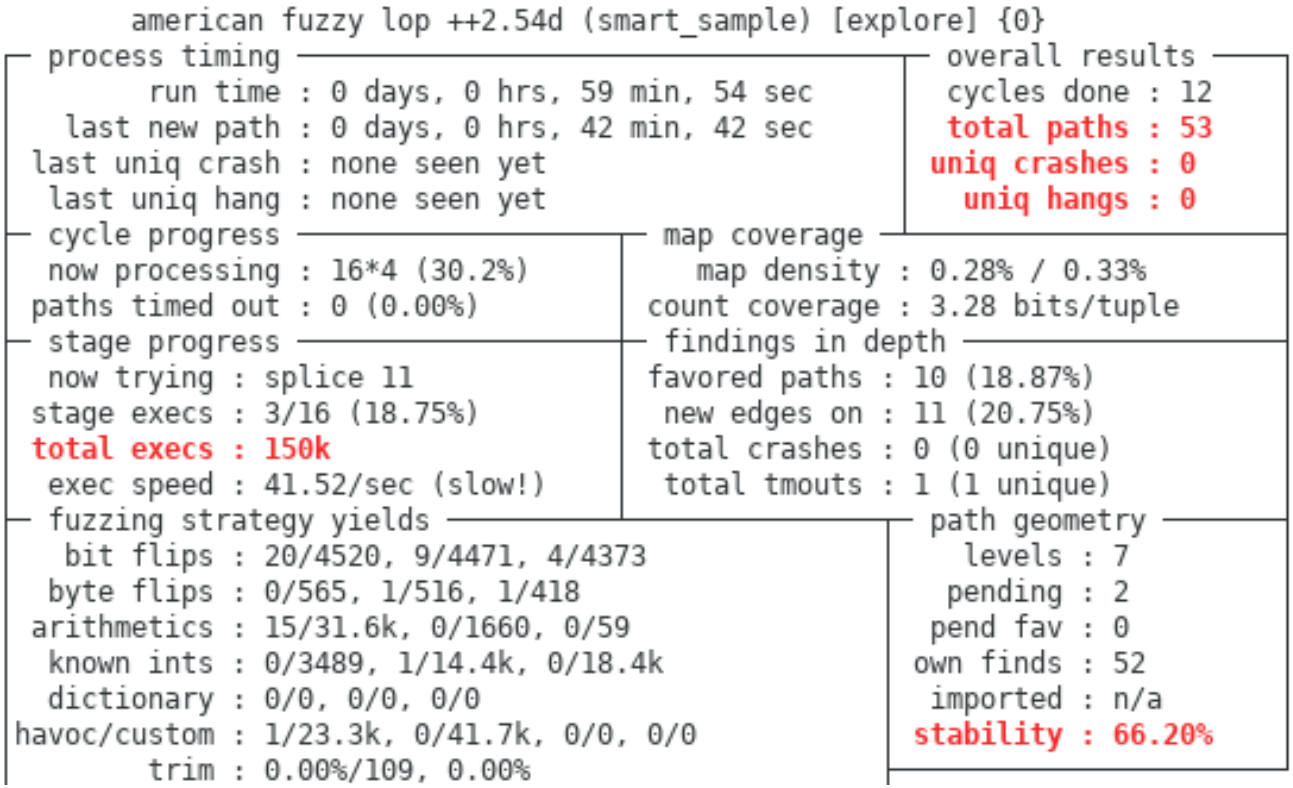
Illustration

Supposons que nous pouvons isoler le code du parser et le *fuzzer* directement en *mono-thread*.



Conséquences

Fuzzons maintenant l'architecture *multi-thread* précédente depuis le broker:



Le code a été parcouru superficiellement.

Le suivi de threads

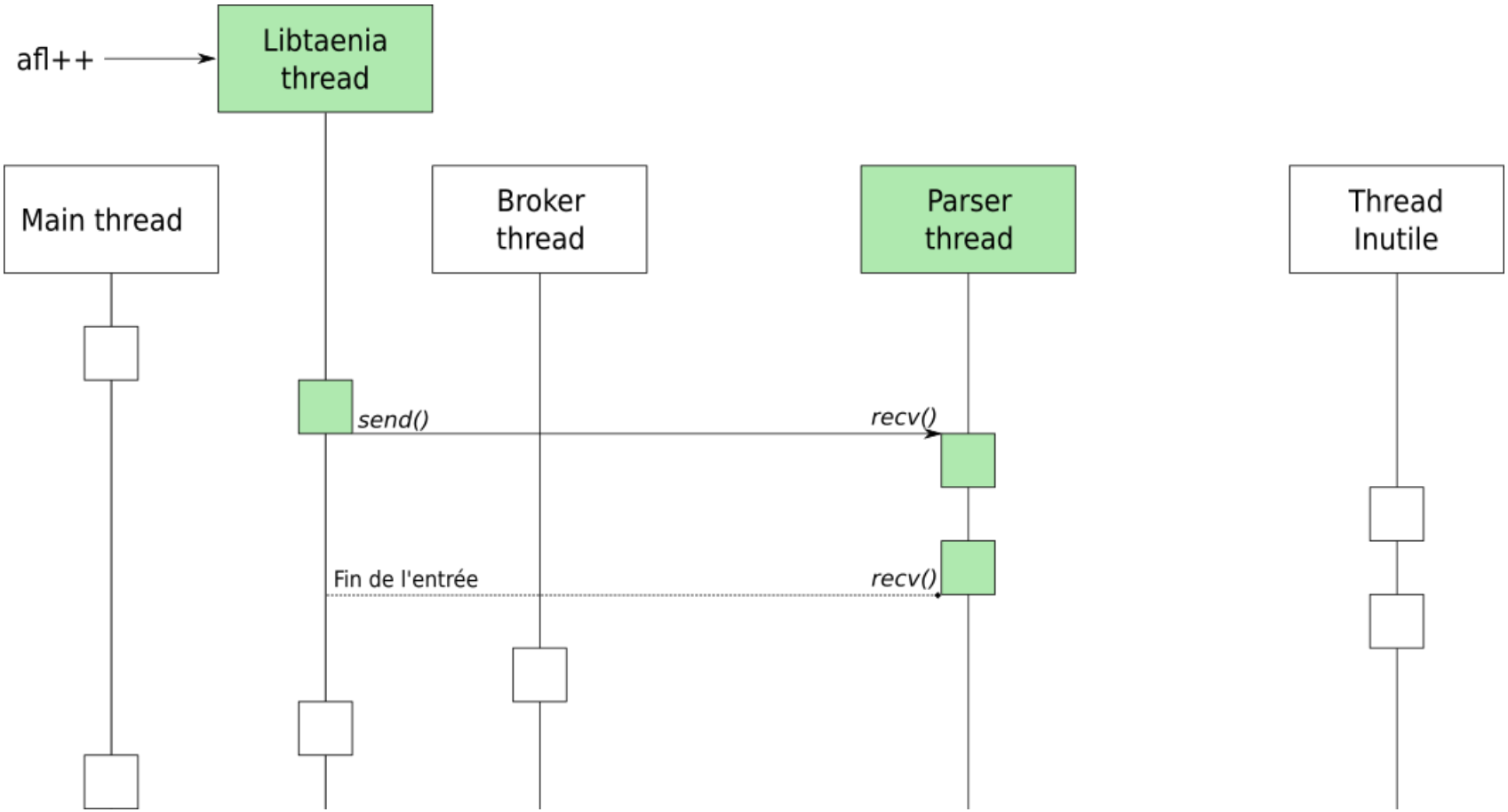
Modification des sondes d'afl-qemu pour les limiter aux *threads* utiles.

- Propagation : les données échangées entre les *threads* sont contaminantes.
- *Hook* : interception des fonctions d'échange de données *inter-threads* avec (jmp / call hooks).

Hypothèses centrales :

- Une donnée ne peut être lue qu'une seule fois.
- Lorsqu'un *thread* lit une nouvelle donnée, nous considérons qu'il a fini de traiter la donnée précédente.

Illustration d'afl-taenia-mt



Conséquences

Un bug "stateful" supplémentaire trouvé.

```
          american fuzzy lop ++2.54d (smart_sample) [explore] {0}
┌─ process timing ─────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 4 min, 55 sec   │  cycles done : 26     │
│   last new path : 0 days, 0 hrs, 4 min, 31 sec   │  total paths : 18     │
│ last uniq crash : 0 days, 0 hrs, 4 min, 29 sec   │ uniq crashes : 2      │
│  last uniq hang : 0 days, 0 hrs, 4 min, 33 sec   │   uniq hangs : 1      │
├─ cycle progress ─────────────┬─ map coverage ──────┴───────────────────────┤
│  now processing : 17.26 (94.4%)  │   map density : 0.04% / 0.10%          │
│ paths timed out : 0 (0.00%)      │ count coverage : 1.00 bits/tuple       │
├─ stage progress ─────────────┼─ findings in depth ─────────────────────────┤
│  now trying : splice 14          │ favored paths : 18 (100.00%)           │
│ stage execs : 75/144 (52.08%)    │  new edges on : 18 (100.00%)           │
│ total execs : 1.09M              │ total crashes : 3 (2 unique)           │
│  exec speed : 3742/sec           │  total tmouts : 4 (1 unique)           │
├─ fuzzing strategy yields ────────┴──────────────┬─ path geometry ─────────┤
│   bit flips : 0/1528, 2/1510, 1/1474           │    levels : 10          │
│  byte flips : 0/191, 0/173, 0/137              │   pending : 0           │
│ arithmetics : 16/10.7k, 0/429, 0/0             │  pend fav : 0           │
│  known ints : 0/1159, 0/4844, 0/6028           │ own finds : 17          │
│  dictionary : 0/0, 0/0, 0/16                   │  imported : n/a         │
│havoc/custom : 0/467k, 0/594k, 0/0, 0/0         │ stability : 94.03%      │
│       trim : 3.54%/36, 0.00%                   │                         │
└────────────────────────────────────────────────┴─────────────────────────┘
```

Conclusion

Le code d'afl-taenia-mt est disponible sur le github de la DGA : https://github.com/DGA-MI-SSI
Avec l'exemple de test utilisé dans cette présentation.
Utiliser cette PoC nécessite :

- D'identifier le code à fuzzer.
- De trouver un point d'entrée.
- D'identifier et hooker les fonctions de communications inter-threads.
- D'adapter la PoC.

Merci

julien.rembinski@def.gouv.fr
benjamin-m1.dufour@def.gouv.fr