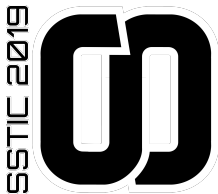
GUSTAVE : Fuzz It Like It's App

(feat. QEMU & AFL)

Stéphane Duverger, Anaïs Gantet

SSTIC, Rennes, 6 juin 2019



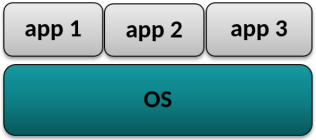
AIRBUS

Introduction

Cibles étudiées : systèmes d'exploitation embarqués

Particularités

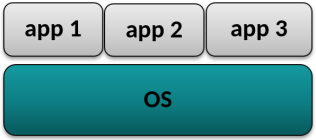
- Noyaux peu dynamiques
- Couches logicielles métier spécifiques
- Importance de la ségrégation spatiale et temporelle



Cibles étudiées : systèmes d'exploitation embarqués

Particularités

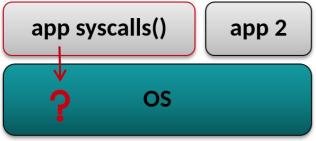
- Noyaux peu dynamiques
- Couches logicielles métier spécifiques
- Importance de la ségrégation spatiale et temporelle



Surface d'attaque considérée

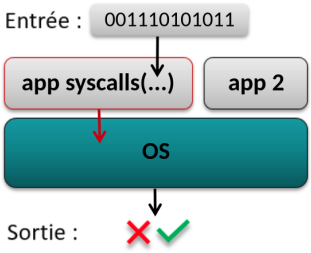
- Depuis une simple application
- Via les appels système

Méthode retenue : *coverage-guided fuzzing*



Coverage-guided fuzzing

- Génération d'entrées conditionnée par le code déjà couvert
- Collecte des informations de couverture de code
- Analyse du comportement de la cible

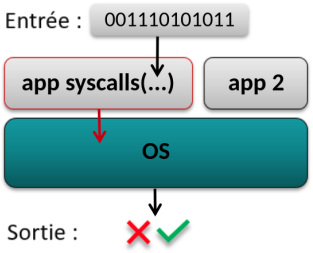


Coverage-guided fuzzing

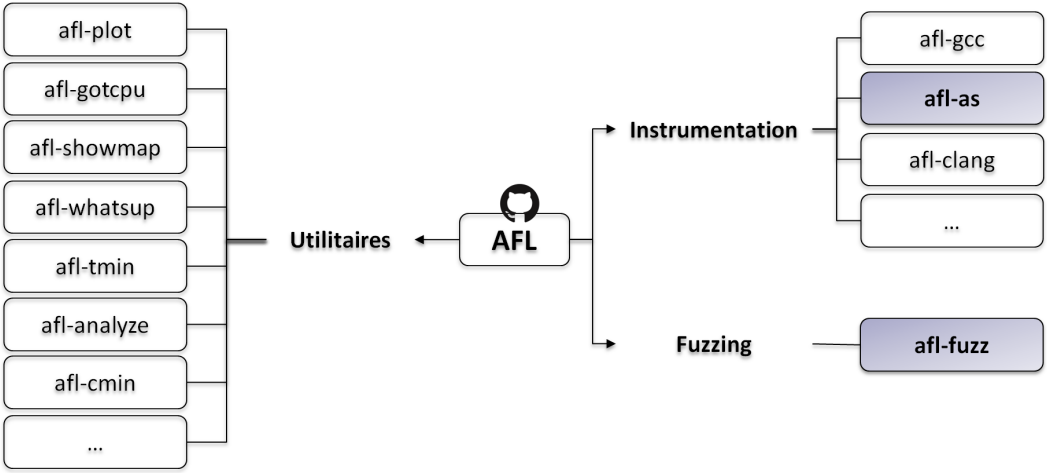
- Génération d'entrées conditionnée par le code déjà couvert
- Collecte des informations de couverture de code
- Analyse du comportement de la cible

Fuzzer retenu : `AFL`

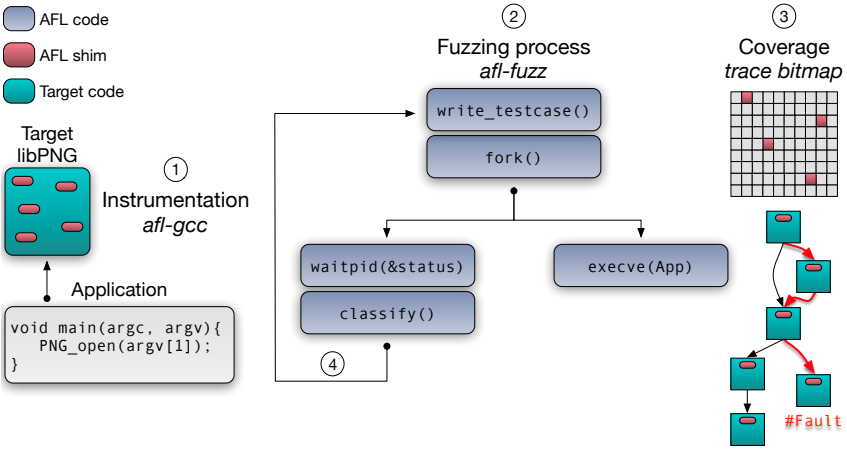
- Efficace sur de nombreux programmes
- Libre, open-source, livré avec plusieurs utilitaires
- **Notre objectif : Utiliser AFL pour fuzzer les OS**



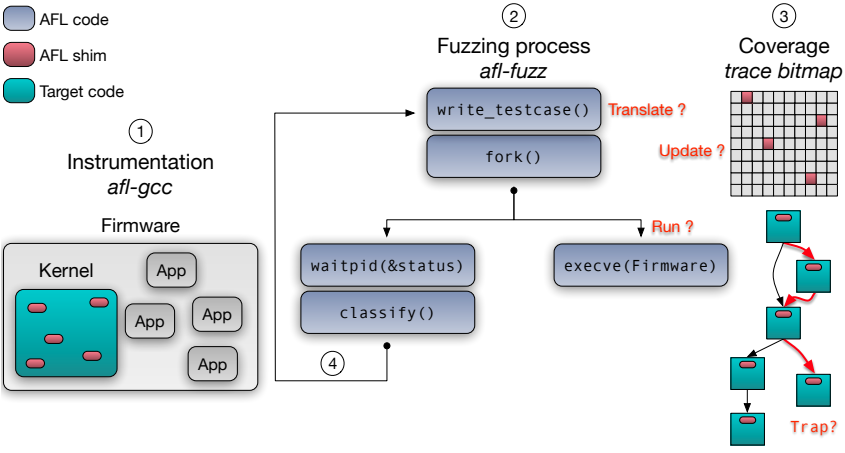
AFL : Un ensemble d'outils



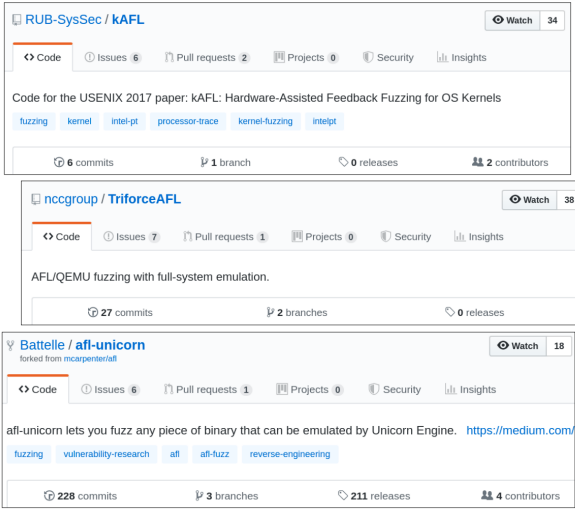
Exemple : *fuzzing* de `libpng`



Exemple : Et pour *fuzzer* un OS ?!



Des candidats ?...

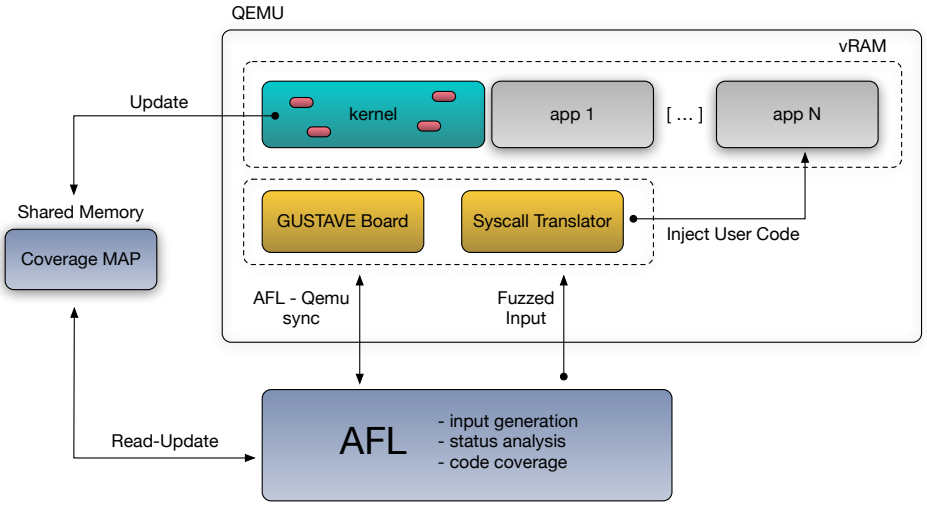


Nos contraintes/souhaits

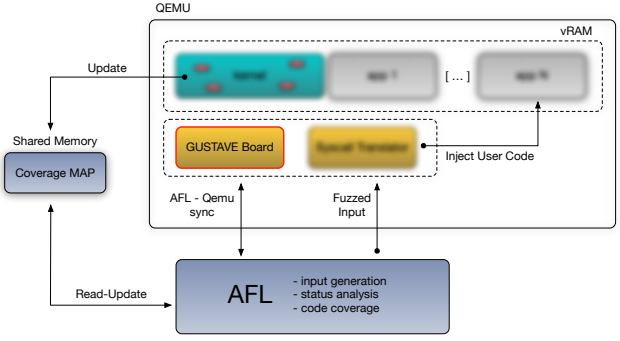
- Indépendant de l'architecture matérielle
- Pas de modification d'`afl-fuzz`
- Pas de dev spécifiques dans l'OS cible
- Outil libre, facilement maintenable

...Conclusion : **"Build your own!" :)**

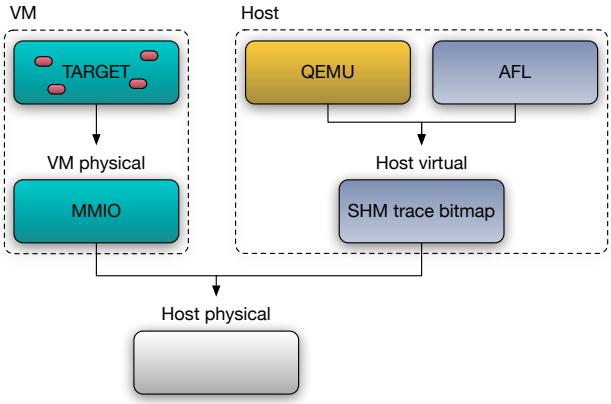
Fonctionnement


QEMU
vRAM
Update
kernel
app 1
[...]
app N
Shared Memory
Coverage MAP
GUSTAVE Board
Syscall Translator
Inject User Code
AFL - Qemu sync
Fuzzed Input
Read-Update
AFL
- input generation
- status analysis
- code coverage

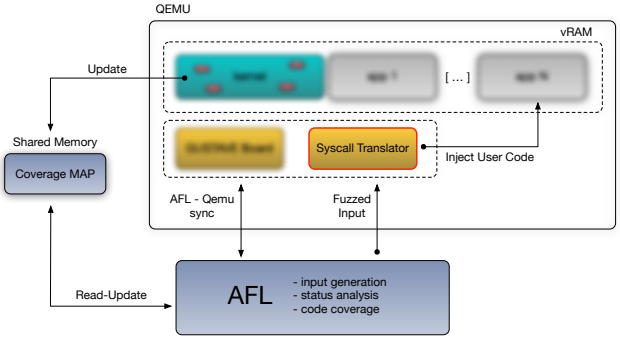
- Implémentation d'une *board* QEMU
  - par architecture matérielle
  - synchronisation avec AFL (*fork-server*)
  - snapshot restauration de la VM
- Pas de modification d'`AFL`
- Pas de modification du TCG
  - instrumentation à la compilation
  - usage de `KVM` possible
  - évite filtrage dynamique



- AFL crée une SHM dans l'hôte
- La cible accède une adresse MMIO arbitraire
- GUSTAVE la redirige vers la bitmap d'AFL
- Aucun surcoût à l'exécution (*like it's app*)



- Transformer les données brutes en programmes
- Séquences d'appels système
- Spécifique à l'architecture et à la cible

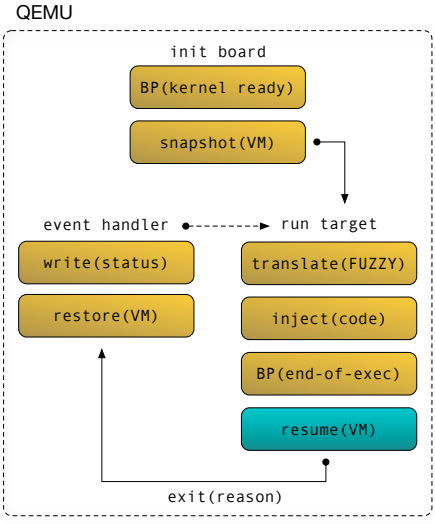


AFL classifie les *test-cases*

- Fin d'exécution normale
- *Time-out*
- Faute (*abort*, *segv*)

GUSTAVE intercepte des évènements

- *Timers* dans QEMU
- *Breakpoints* internes
  - Fin du test injecté
  - Fautes contrôlées : `panic`, `reboot`
- Pas de *véritable* garde-fou noyau
  - Détection d'accès illégitimes *silencieux*
  - Définition d'oracles mémoire



Utilisation

- Instrumentation à la compilation (`afl-gcc`/`afl-as`)
- Optimiser le système selon vos critères :
  - Deux applications basiques, peu de *scheduling*
  - Scénario complexe d'échanges entre applications
  - Focalisé sur un appel système spécifique

```
$ CC=afl-gcc make
[CC] partition: afl-gcc -c -W partition.c -o partition.o
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 125 locations (32-bit, non-hardened mode, ratio 100%).
```

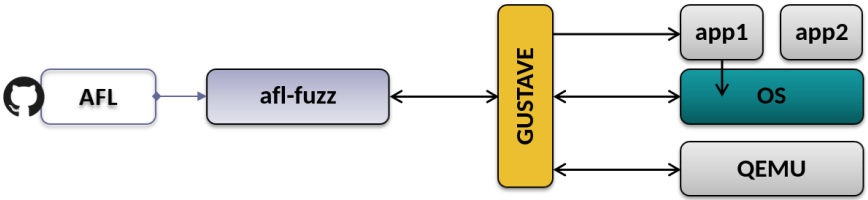
```
$ alias afl="afl-fuzz -d -t 10000 -i /tmp/afl_in \
-o /tmp/afl_out -- qemu-system-ppc -M afl        \
-nographic -bios rom.bin -gustave"

$ afl config/pok_ppc_single.json
```

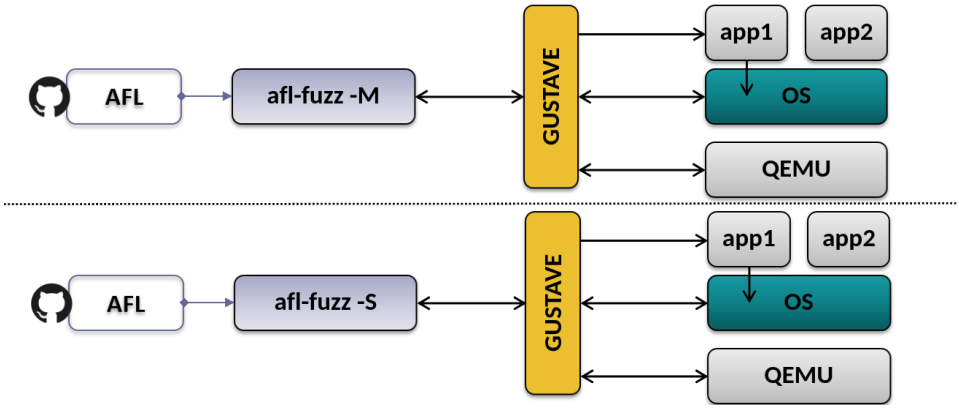
```
{   "user-timeout": 10000,
    "qemu-overhead": 10,
    "vm-state-template": "/tmp/afl.XXXXXX",
    "afl-control-fd": 198,
    "afl-status-fd": 199,
    "afl-trace-size": 65536,
    "afl-trace-env": "__AFL_SHM_ID",
    "afl-trace-addr": 3758096384,
    "vm-part-base": 221184,
    "vm-part-size": 380768,
    "vm-part-off": 4,
    "vm-nop-size": 65536,
    "vm-fuzz-inj": 221188,
    "vm-size": 0,
    "vm-part-kstack": 0,
    "vm-part-kstack-size": 0,
    "vm-fuzz-ep": 4,
    "vm-fuzz-ep-next": 8,
    "vm-panic": 4293949504,
    "vm-cswitch": 0,
    "vm-cswitch-next": 0
}
```

Demo. It's time to fuzz!

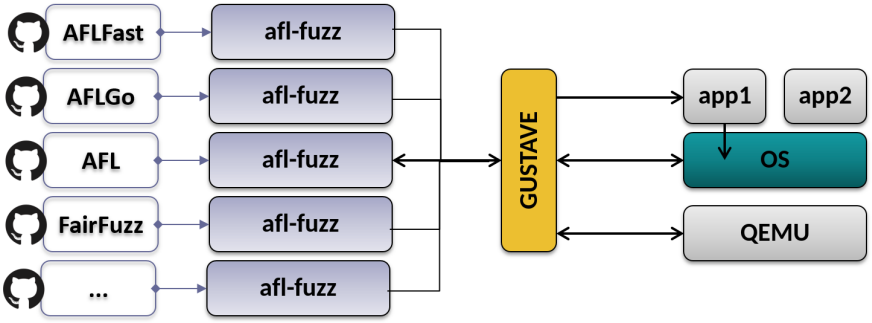
Usage basique



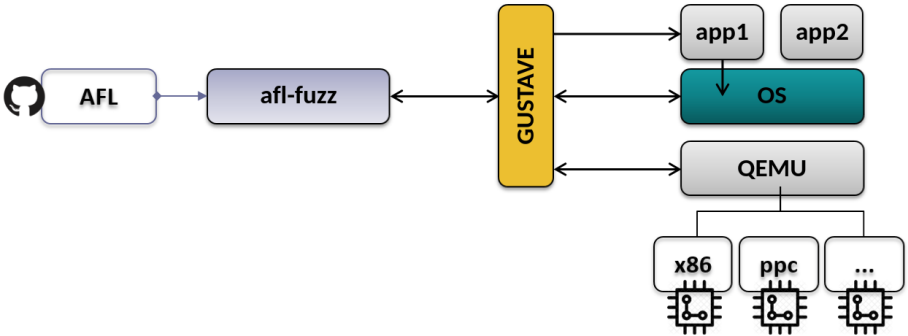
Usage multicœur



Autres dérivés d'afl-fuzz



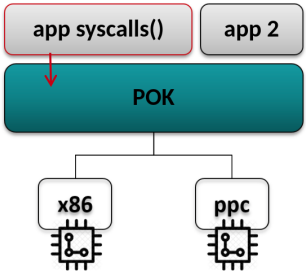
Diverses architectures matérielles



Cas d'usage : POK

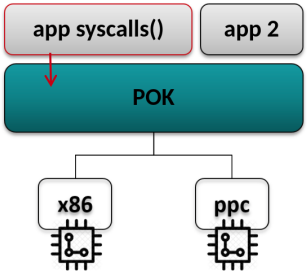
Cible intéressante

- Petit OS, open-source
- Vérifié formellement à 90%
- ... avec des vulnérabilités d'implémentation de ségrégation mémoire :)



Cible intéressante

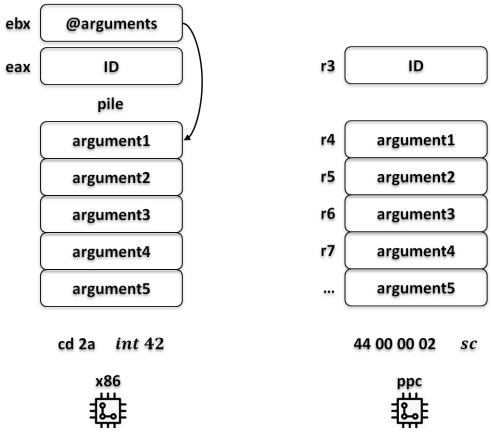
- Petit OS, open-source
- Vérifié formellement à 90%
- ... avec des vulnérabilités d'implémentation de ségrégation mémoire :)



Efforts d'ingénierie

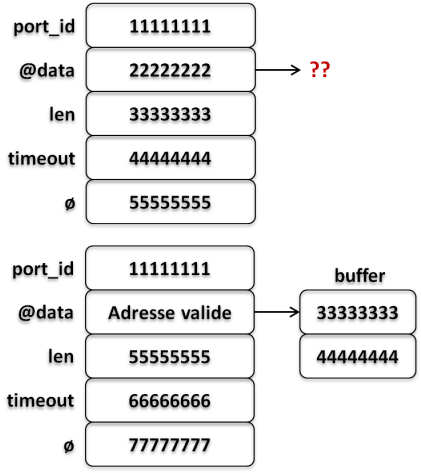
- Analyse
  - Compréhension de la mécanique des appels système / ABI
  - Compréhension de la ségrégation mémoire
- Développements spécifiques
  - De traducteurs d'entrées compatibles à POK
  - D'oracles mémoire correspondant à la logique mémoire de POK

- 50 services noyau exposés aux programmes utilisateur via des appels système
- Logique d'implémentation différente suivant l'architecture matérielle



Diverses possibilités d'interprétation des entrées
(exemple de `POK_SYSCALL_MIDDLEWARE_QUEUEING_SEND`)

- Traitement identique quel que soit le type d'argument
- Traitement spécifique pour les pointeurs de structures
  - Adresse mémoire valide
  - Fuzzing déporté sur le contenu pointé

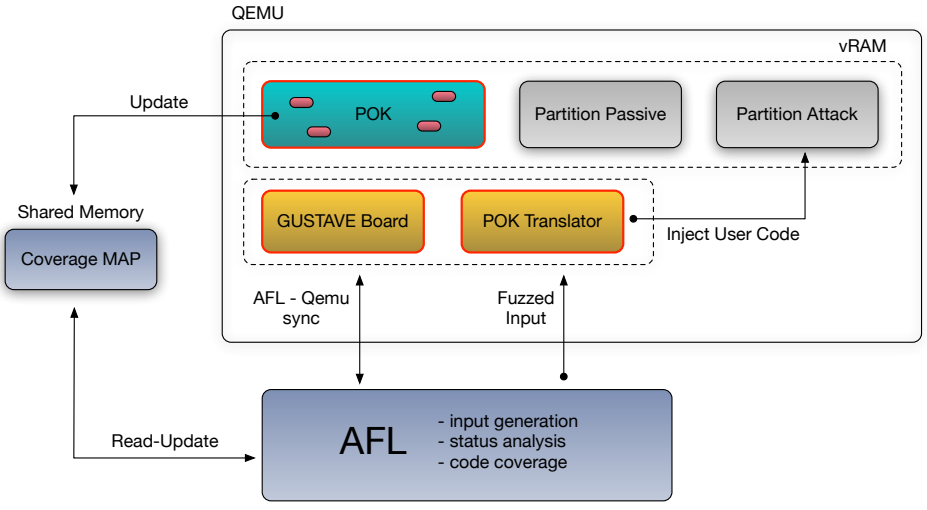


MMU POK

- 1 paire de segments code/données pour chaque programme utilisateur
- 1 paire de segments code/données pour le noyau – en FLAT !! :(
  - Implique une vérification au niveau logiciel

Oracle `GUSTAVE`

- Mapping exclusif des plages mémoire utilisateur / noyau
- Interception des fautes de pages sur les zones non mappées


QEMU
vRAM
Update
POK
Partition Passive
Partition Attack
Shared Memory
Coverage MAP
GUSTAVE Board
POK Translator
Inject User Code
AFL - Qemu sync
Fuzzed Input
Read-Update
AFL
- input generation
- status analysis
- code coverage

Résultats pour POK

Vulnérabilités trouvées

- Vulnérabilité attendue retrouvée
  - Absence de vérification d'une adresse passée en argument
  - Conséquence : possibilité d'écriture arbitraire
- Détection automatique de tous les autres appels système vulnérables au même problème
  - 25 autres possibilités d'écriture arbitraire

Performances

- environ 500 tests/sec
- Stabilité (déterminisme) du fuzzing proche de 100%

Conclusion

GUSTAVE aujourd'hui

- AFL peut fuzzer des OS embarqués (*like it's app*)
- Intégration dans des *boards* x86 et PPC
- Conditionné par :
  - Support de l'OS dans QEMU
  - Compréhension de l'ABI/stratégie de ségrégation mémoire
- Open-source (framework + exemples pour POK)

GUSTAVE demain

- Grammaire pour *syscalls* par architecture (JSON ?)
- Optimisations, oracles novateurs, ...

Merci pour votre attention :)
Questions ?

stephane.duverger@airbus.com
anais.gantet@airbus.com

@AirbusSecLab
https://github.com/airbus-seclab/gustave