SYNACKTIV

# Émulation et Fuzzing Black-Box avec Qiling

Bière Sécu Toulouse

28/11/2024

# whoami

SYNACKTIV

- Pôle Reverse engineering @ Synacktiv
- Passionné de recherche de vulnérabilités / exploitation
- CTF w/ ECSC Team France, RUBYGG, …


**@voydstack**

# Motivations

# Motivations

Pourquoi émuler ?

SYNACKTIV

- Cas d'usage variés
  - **Recherche de vulnérabilités**
  - **Fuzzing**
  - Debugging d'exploit
  - Analyse de malware
  - ...

# Motivations

SYNACKTIV

## Pourquoi émuler ?

- Certaines targets ne sont pas facilement accessibles
  - Systèmes embarqués, IoT, ...
  - Architecture spécifique (ARM, MIPS, ...)
  - Système à part entière (système de fichiers, périphériques hardware, ...)
  - Code très souvent propriétaire (analyse en boîte noire)
- Liberté d'analyse
  - Pas de contraintes matérielles
  - Pas de risque de casser le device
  - Analyse dynamique, instrumentation
  - Debugging
- Pas besoin d'acheter le matériel

# Motivations

SYNACKTIV

## Comment émuler ?

- Plusieurs solutions
- Émulation complète
  - QEMU system mode
  - Panda
  - Avatar2
  - …
- Émulation partielle
  - QEMU user mode
  - Unicorn
  - Qiling
  - …

# Motivations

SYNACKTIV

Que choisir ?

- Émulation complète
  - Avantages
    - Émulation complète du système
    - Très fidèle à la réalité
  - Inconvénients
    - Complexité
    - Temps de mise en place
- Émulation partielle
  - Avantages
    - Plus simple à mettre en place
    - Plus léger / rapide
  - Inconvénients
    - Moins fidèle à la réalité

**Émulation partielle**

Suffisante pour nos besoins (recherche de vulnérabilités / fuzzing)

# Qiling

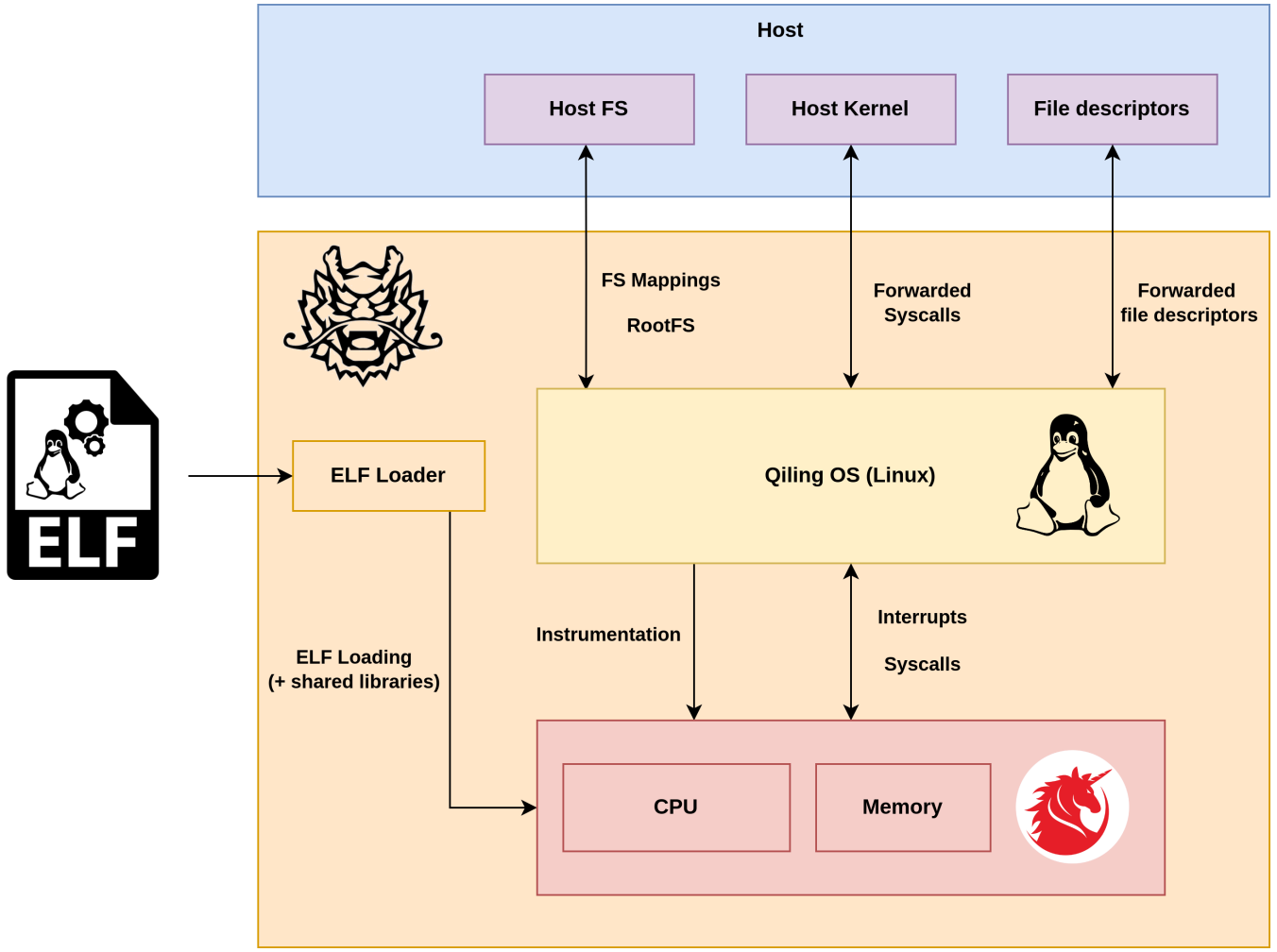
# Qiling

## Introduction

- Framework d'émulation open-source
  - Écrit en Python (facilement extensible)
  - Basé sur Unicorn (pour l'émulation CPU)
- Supporte de nombreuses architectures
  - x86, x86_64, ARM, MIPS...
- Émule des fonctionnalités de différents OS
  - Linux, Windows, macOS...
  - Appels systèmes, gestion de fichiers, réseau...
- Supporte différents formats d'exécutable
  - ELF, PE, Mach-O...
- API haut niveau et simple d'utilisation

# Qiling

## Architecture simplifiée



# Cas pratique

DLink DAP-1665

# Cas pratique

DLink DAP-1665

SYNACKTIV

- Boitier Wi-Fi (point d'accès, répéteur, ...)
- Firmware disponible sur le site du constructeur
- Embarque un système Linux, et un serveur web
  - Visiblement écrit en C
  - et d'autres services...
- CPU MIPS32 (Little Endian)
- Cible intéressante à émuler

# Cas pratique

## Préparation du terrain - Firmware

SYNACKTIV

- Récupération du firmware
  - Plusieurs moyens, non abordés ici
- Extraction du système de fichiers
  - `binwalk`, `unsquashfs`, ...
- Nettoyage du FS
  - Liens symboliques invalides
  - Répertoires inexistants (tmp, ...)
- Analyse statique rapide du serveur web

# Cas pratique

## Préparation du terrain - Qiling 101

SYNACKTIV

- Installation de Qiling `pip3 install qiling` → `venv` conseillé

```
from qiling import Qiling
from qiling.const import QL_VERBOSE

ROOTFS = '../squashfs-root'
BIN = f'{ROOTFS}/sbin/httpd'

ql = Qiling(argv=[BIN], rootfs=ROOTFS, verbose=QL_VERBOSE.DISABLED)

ql.run()
```

```
$ python3 ./emu-dlink-dap-1665.py
../squashfs-root/sbin/httpd: Cannot open /dev/null
open: Operation not permitted
```

Le binaire se lance !

# Émulation

## Émulation du serveur web

SYNACKTIV

- Serveur web → programme complexe
- Pas de solution magique pour l'émuler
- → Reverse-engineering nécessaire
- Résoudre progressivement les différentes erreurs
- Profiter un maximum des APIs de Qiling

# Émulation

SYNACKTIV

## Émulation du serveur web - problèmes rencontrés

- Fichiers manquants
  - `/dev/null`, `/dev/urandom`, PID file, ... → FS Mapping via Qiling
  - Fichier de configuration (option `-f`) → analyser les scripts d'init
- Lancement en mode démon par défaut → option `-n` pour le lancer en foreground

```
ql = Qiling(
    argv=[BIN, '-f', '/etc/webconfig', '-n'], # Ajout du fichier de config
    rootfs=ROOTFS,
    verbose=QL_VERBOSE.DEFAULT
)

# FS Mappings (forwarded to host filesystem)
ql.add_fs_mapper('/dev/null', '/dev/null')
ql.add_fs_mapper('/dev/urandom', '/dev/urandom')
ql.add_fs_mapper('/etc/webconfig', 'config')
ql.add_fs_mapper('/var/run/httpd.pid', 'httpd.pid')
```

# Émulation

Émulation du serveur web - problèmes rencontrés

SYNACKTIV

- `setsockopt` échoue sur la socket serveur

```
[=] setsockopt(sockfd=0x4, level=0xffff, ...) = -0x1 (EPERM)
setsockopt: cannot bind to device
```

- Solution simple: toujours retourner `0` 😄

```
ql.os.set_syscall('setsockopt', lambda ql: 0)
```

# Émulation

SYNACKTIV

## Émulation du serveur web - problèmes rencontrés

- `getsockopt` : option spécifique non implémentée par Qiling

```
[x] Syscall ERROR: ql_syscall_getsockopt DEBUG: Could not convert emulated socket option 80 to a socket option name
```

- Ajout d'un hook pour traiter l'option `80`

```
def getsockopt_hook(ql: Qiling) -> None:
    if ql.arch.regs.a2 == 80:
        ql.arch.regs.v0 = -1 # Return value
        ql.arch.regs.pc = ql.arch.regs.ra # Skip function

ql.os.set_api('getsockopt', getsockopt_hook)
```

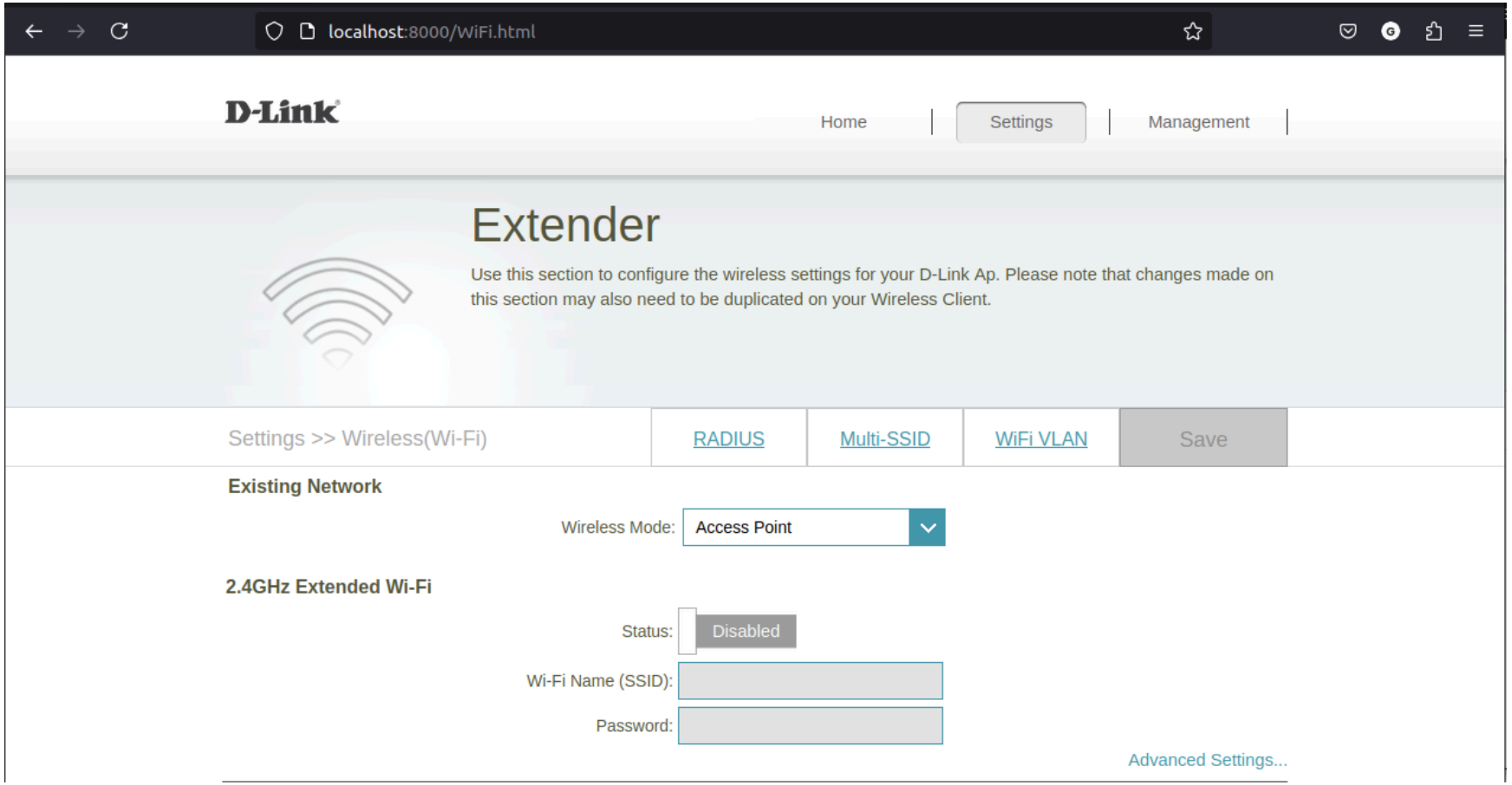
### Hook set_api

Hook placé sur une fonction de la libc, Qiling résoud les symboles importés par le binaire

# Émulation

## Émulation du serveur web

- Ça fonctionne ! 🎉



# Émulation

## Émulation du serveur web

SYNACKTIV

- ~30 lignes de Python → Serveur web quasi fonctionnel !
- Un peu de travail de RE / debugging *under the hood*
- Certaines fonctionnalités non implémentées
  - Exécution de CGI
  - Configuration du serveur incomplète (vhosts, ...)

# Fuzzing

Utilisation d'AFL++ avec Qiling

# Fuzzing

## Utilisation d'AFL++ avec Qiling



- Qiling se base sur Unicorn → UnicornAFL supporté !
- Extension de Qiling ( `qiling.extensions.afl` )
- API similaire à UnicornAFL

# Fuzzing

## Quoi cibler ?

- Exécution de CGI non gérée
- Mais… parsing de requêtes HTTP fonctionnel !
- Nécessite un peu plus de travail d'instrumentation…

# Fuzzing

## Méthodologie

SYNACKTIV

- But: faire renvoyer l'input courant via un `read` sur une socket client
  - Sans utiliser de socket réelle
- Idée: Faire retourner à `accept` un *fd* spécial agissant comme un *pipe*
  - `read` sur ce *fd* renvoie l'input du fuzzer
- Peut poser problème
  - Opérations sur des sockets sont spécifiques
  - Solution: instrumentation !

# Fuzzing

## Fake file descriptor

SYNACKTIV

```
# Fake file descriptor
class fuzz_pipe:
    FAKE_FD = 42

    def read(self, n: int) -> bytes:
        global testcase
        read_testcase = testcase[:n]
        testcase = testcase[n:]
        return read_testcase

    def write(self, data: bytes) -> int:
        ql.log.info(f'Sent testcase: {data}')
        return len(data)

    def close(self) -> None:
        ...
```

# Fuzzing

Hijacking de la socket client

SYNACKTIV

```
# Accept hook
def accept_fakefd(ql: Qiling) -> None:
    ql.arch.regs.v0 = fuzz_pipe.FAKE_FD
    ql.arch.regs.pc = ql.arch.regs.ra

ql.os.set_api('accept', accept_fakefd)
```

- Définition du *fd* `42` vers notre pipe

```
fuzz_fd = fuzz_pipe()
ql.os.fd[fuzz_pipe.FAKE_FD] = fuzz_fd
```

# Fuzzing

## Interface avec AFL++

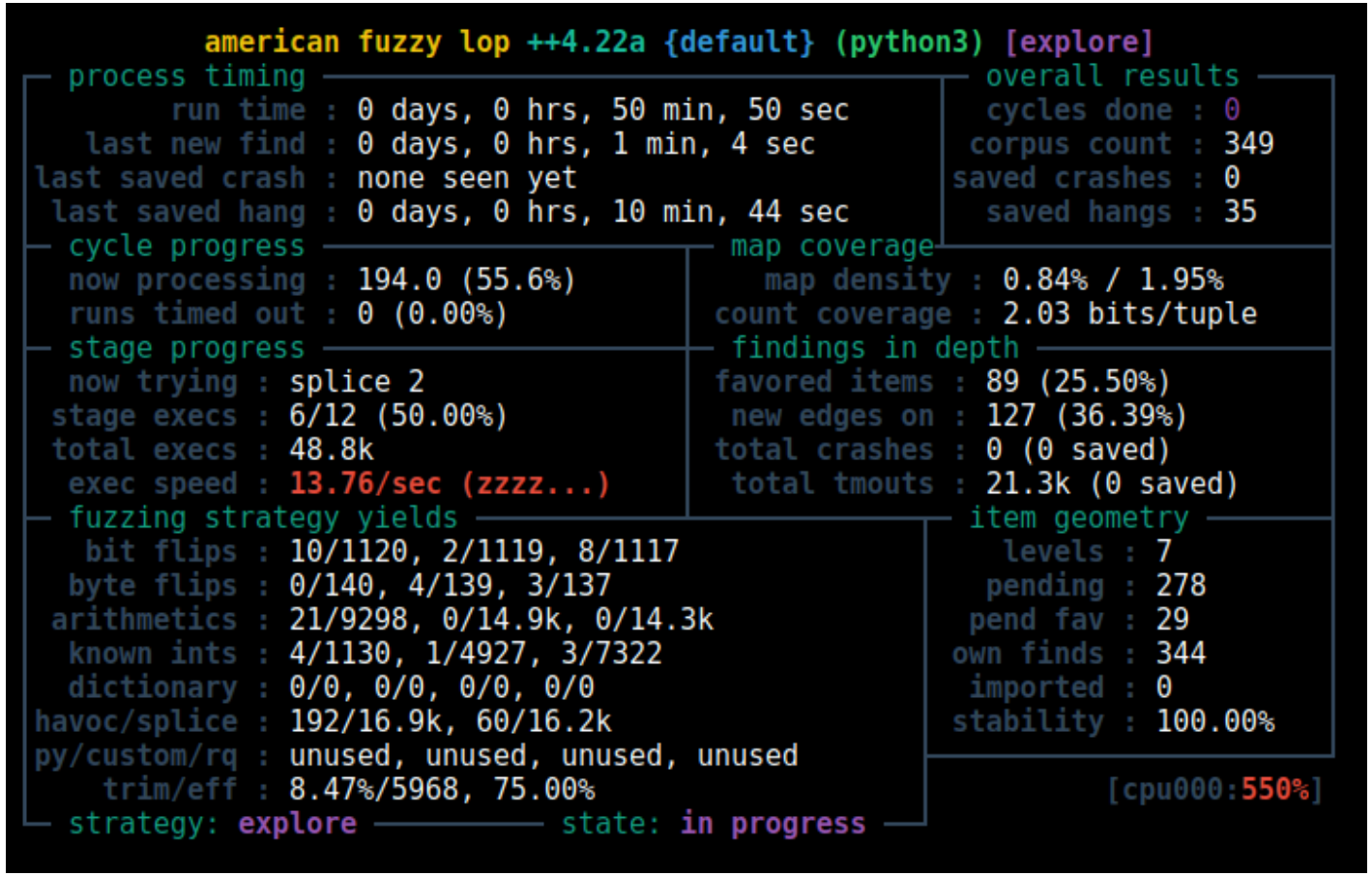
SYNACKTIV

```
testcase_file = sys.argv[1]
testcase = None

def load_testcase(ql: Qiling, input: bytes, persistent_iter: int) -> None:
    global testcase
    testcase = input

def start_afl(ql: Qiling) -> None:
    try:
        afl.ql_afl_fuzz(
            ql=ql,
            input_file=testcase_file,
            place_input_callback=load_testcase,
            exits = [FUZZ_END, ql.os.exit_point]
        )
    except UcAflError as e:
        ql.log.error(f'Error initializing AFL: {e}')

ql.hook_address(start_afl, FUZZ_START)
ql.run(end=FUZZ_END)
```

# Fuzzing

It works!

SYNACKTIV



# Conclusion

# Conclusion

SYNACKTIV

- Qiling: un framework puissant pour l'instrumentation de binaires
- Simple d'utilisation, API riche
- En développement, mais déjà très fonctionnel
  - Ne pas hésiter à patcher le code au besoin
- Beaucoup d'autres fonctionnalités disponibles (non abordées ici)
- Mais pas magique non plus → reverse-engineering nécessaire

# Conclusion

Après une petite recherche sur Github...

**michielboland/mathopd** · src/request.c

```
1371                    s = r->range_s;
1372                    if (s) {
1373                            if (parse_range_header(r, s) == -1)
1374                                    log_d("ignoring Range header \"%s\"", s);
1375                    }
1376            } else if (r->method == M_POST) {
1377                    if (r->in_content_length == 0) {
```

**mathopd** Public

Watch 8 | Fork 11 | Star 56

master | 1 Branch | 3 Tags | Go to file t | Add file | Code

**michielboland** Bump version number, adjust CHANGES — 4fe9239 · 7 months ago — 1,638 Commits

**About**

Very small, yet very fast HTTP server

www.mathopd.org

# Conclusion

SYNACKTIV

Après une petite recherche sur Github...

mathopd Public

Fork 11 Star 56

master 1 Branch 3 Tags

michielboland Bump version number, adjust CHANGES 4fe9239 · 7 months ago 1,638 Commits

About

Very small, yet very fast HTTP server

www.mathopd.org

# Merci !

Avez-vous des questions ?

SYNACKTIV

https://www.linkedin.com/company/synacktiv

https://twitter.com/synacktiv

https://synacktiv.com