Qiling Framework: Introduction

December, 2020

KaiJern Lau, kj -at- qiling.io
ZiQiao Kong, mio -at- qiling.io
ChenXu Wu, kabeor -at- qiling.io

Who are We

JD.COM

Beijing, Stays in the lab 24/7 by hoping making the world a better place

- IoT Research
- Blockchain Research
- Fun Security Research

Qiling Framework

Cross platform and multi architecture advanced binary emulation framework

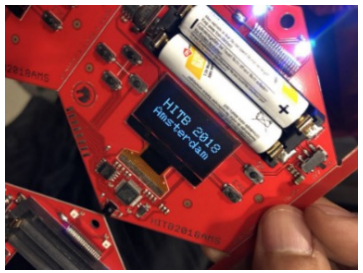
- https://qiling.io
- Lead Developer
- Founder

Badge Maker

Electronic fan boy, making toys from hacker to hacker

- Reversing Binary
- Reversing IoT Devices
- Part Time CtF player

Badge Designer for Hacking Conferences



Some Recent Talk (Partial)

- 2016, Qcon, Beijing, Speaker, nRF24L01 Hijacking
- 2016, Kcon, Beijing, Speaker, Capstone Unicorn Keystone
- 2017, Kcon, Beijing, IoT Hacking Trainer
- 2018, Kcon, Beijing, IoT Hacking Trainer
- 2018, Brucon, Brussel, Speaker, IoT Virtualization
- 2018, H2HC, San Paolo, Speaker, IoT Virtualization
- 2018, HITB, Beijing/Dubai, Speaker, IoT Virtualization
- 2018, beVX, Hong Kong, Speaker, HackCUBE - Hardware Hacking
- 2019, DEFCON USA, Qiling Framework Preview
- 2019, Zeronights, Qiling Framework to Public
- 2020, Nullcon GOA, Building Reversing Tools with Qiling
- 2020, HITB AMS, Building Reversing Tools with Qiling
- 2020, HITB Singapore, Training, How to Hack IoT with Qiling
- 2020, HITB UAE, Training, Lightweight Binary Analyzer
- 2020, Blackhat USA, Building IoT Fuzzer with Qiing
- 2020, Blackhat Singapore, Lightweight Binary Analyzer
- 2020, Blackhat Europe, Deep Dive Into Obfuscated Binary

Qiling Framework

- Emulate and instrument ARM, ARM64, MIPS, X86 and X8664
- Emulate and instrument Linux, MacOS, iphoneOS, Windows and FreeBSD
- High-level Python API access to register, CPU and memory
- 1.8k Github star, more than 12,000+ pypi download, 65+ contributors worldwide
- Contributor from Dell, Intel, SentinalOne and etc. From industry and academic.

About lazymio && kabeor && kevin

~ $ whoami
Lazymio

~ $ file Lazymio
The sheperd lab, JD security, Security Engineer.
CTF player, member of Lancet.
GeekPwn 2019 Hall of Fame.

~ $ ls -l Lazymio
Reverse engineering.
Binary analysis.
Writing code for fun.

~ $ which Lazymio
Github: https://github.com/wtdcode
Blog: https://blog.lazym.io/
Twitter: https://twitter.com/pwnedmio

Name: kabeor

Security Engineer at The Shepherd Lab, JD Security.

Core developer of Qiling.

BlackHat Asia & Europe 2020 - Speaker

China kanxue SDC 2020 - Speaker

HITB Training 2020 - Speaker

Github: https://github.com/kabeor

Blog: https://kabeor.cn

Twitter: https://twitter.com/Angrz3_K

Name: chfl4gs

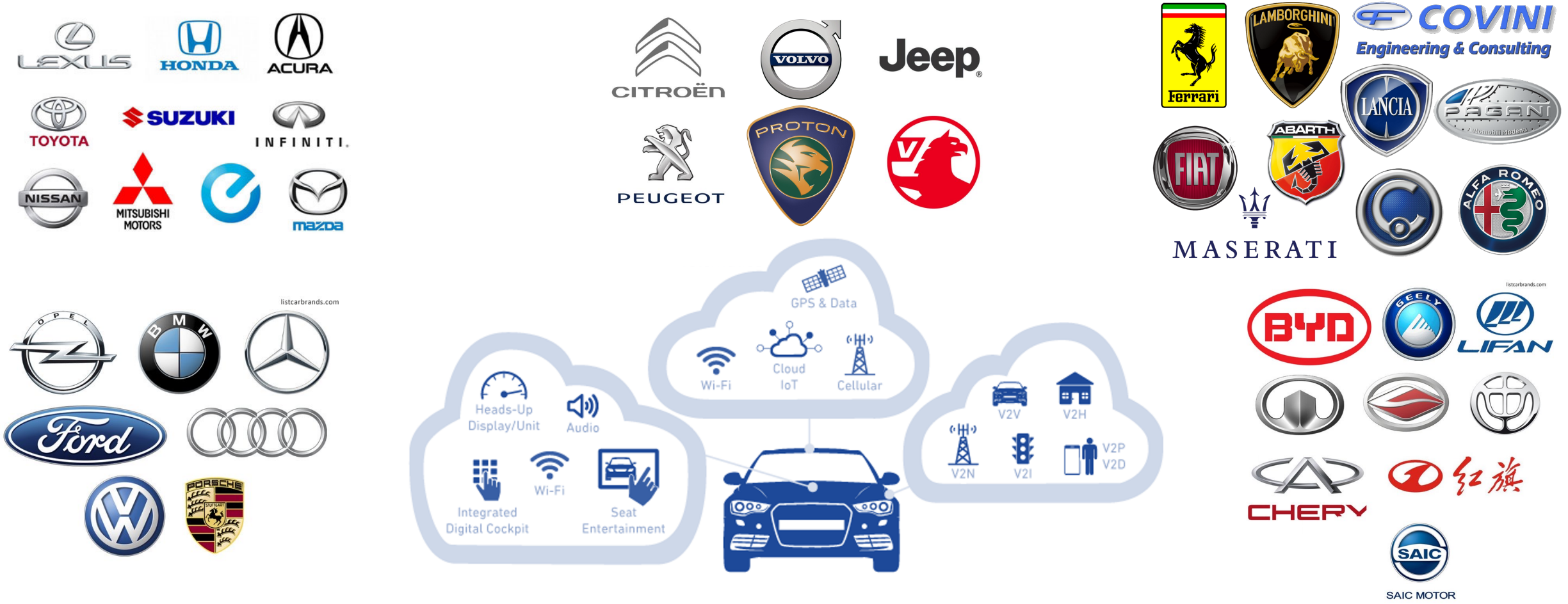
Security Engineer at The Shepherd Lab, JD Security.
Contributor of Qiling Framework
Core member of Malaysia Chapter, The Honeynet Project.
Core developer of Hex LiveCD project
SANS Advisory Board member

*Nix systems junkie
Blue team
DFIR

Github: https://github.com/chfl4gs
Twitter: https://twitter.com/chfl4gs_

Make IoT Reverse Engineering Great

Today’s IoT Analysis



To under a firmware

First, you need a IoT, If not too expensive to own one

How We Fix It

- GPS
- Wi-Fi
- Bluetooth
- Audio
- Screen

No Actual Hardware Needed

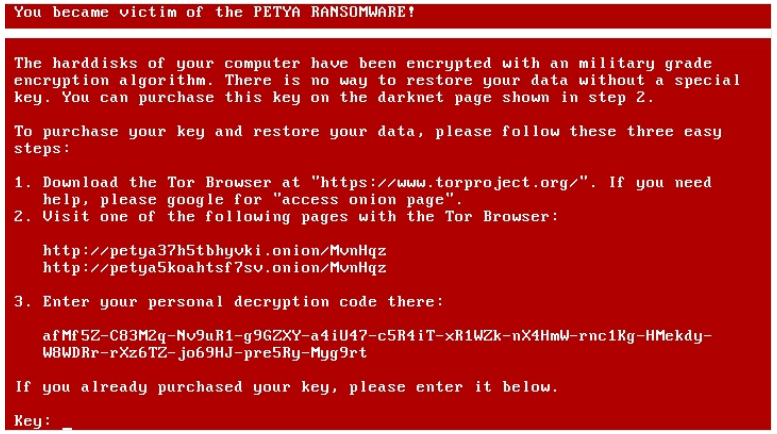
Qiling Framework

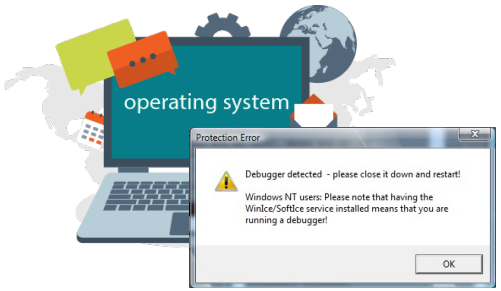
Bring the entire car firmware's binary into emulation

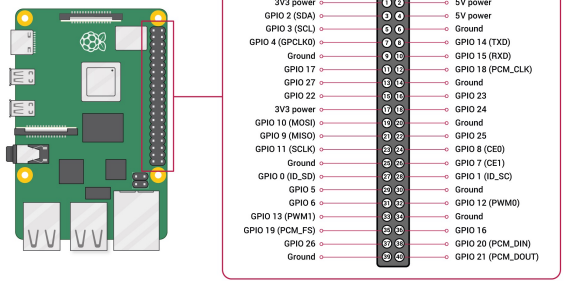
with virtual devices support

Wait, There are Virtual Machines

Current Virtual Machine Limitation







- MBR
- UEFI
- Smart Contract
- GPIO
- Anti-Anti Debug
- Qualcomm EDL

Most modern platform are either limited or NONE emulation or proper analysis tools

Qiling Framework

Overview

CPU Architecture

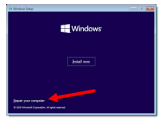
Loader

OS

Posix

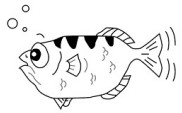
macOS®

EDL



MBR

Debugger



Extensions

coverage

idaplugin

report

sanitizers

windows sdk

Instrumentation

External Hardware Emulation

Qiling Framework

Features

- Cross platform: Windows, MacOS, Linux, BSD, UEFI, MBR
- Cross architecture: X86, X86_64, Arm, Arm64, MIPS, 8086
- Multiple file formats: PE, UEFI(PE), MachO, ELF, EDL (ELF), COM
- Emulate & sandbox machine code in a isolated environment
- Provide high level API to setup & configure the sandbox
- Fine-grain instrumentation: allow hooks at various levels (instruction/basic-block/memory-access/exception/syscall/IO/etc)
- Allow dynamic hotpatch on-the-fly running code, including the loaded library
- True Python framework, making it easy to build customized analysis tools on top
- GDBServer support - GDB/IDA/r2
- IDA Plugin
- OS profiling support

	8086	x86	x86-64	ARM	ARM64	MIPS
Windows (PE)	-	☑	☑	-	☐	-
Linux (ELF)	☐	☑	☑	☑	☑	☑
MacOS (MachO)	-	☐	☑	-	☐	-
BSD (ELF)	☐	☐	☑	☐	☐	☐
UEFI	-	☑	☑	-	-	-
DOS (COM)	☑	-	-	-	-	-
MBR	☑	-	-	-	-	-

Similarity

User Mode Emulation

qemu-usermode

- The TOOL
- Limited OS Support, Very Limited
- No Multi OS Support
- No Instrumentation
- **Syscall Forwarding**

usercorn

- Very good project !
- It's a Framework !
- Mostly *nix based only
- Limited OS Support (No Windows)
- Go and Lua is not hacker's friendly
- **Syscall Forwarding**

Binee

- Very good project too
- Only X86 (32 and 64)
- Limited OS Support
- Only PE Files
- Just a tool, we don't need a tool
- Again, is GO

WINE

- Limited ARCH Support
- Limited OS Support, only Windows
- Not Sandbox Designed
- No Instrumentation

Speakeasy

- Very good project too
- X86 32 and 64
- PE files and Driver
- Limited OS Support
- Only Windows

Zelos

- Very good project !
- It's a Framework !
- Linux based only (No Windows)
- Incomplete support for Linux multi arch

Framework

Framework, NOT Tools

EFI Fuzzer

Sentinel-One / efi_fuzz

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights

master | 2 branches | 0 tags | Go to file | Add file | Code

liba2k Adding docker support. (#5) 8d841cb 24 days ago 15 commits

NotMyUefiFault	Add STACK_UNINITIALIZED_MEMORY_LEAK case to generate stack infole...	27 days ago
images	Initial public commit	last month
samples	Initial public commit	last month
scripts	Adding docker support. (#5)	24 days ago
taint	Add taint propagation for uninitialized memory	27 days ago
tests	Adding docker support. (#5)	24 days ago
.gitignore	Update .gitignore	27 days ago
Dockerfile	Adding docker support. (#5)	24 days ago
README.md	Adding docker support. (#5)	24 days ago
efi_fuzz.py	Use functools.partial for auto_int	27 days ago
requirements.txt	Update requirements.txt	27 days ago
sanitizer.py	Add taint propagation for uninitialized memory	27 days ago

README.md

efi_fuzz

A simple, coverage-guided fuzzer for UEFI NVRAM variables. Based on Qiling and AFL++.
Written by Itai Liba (@liba2k) and Assaf Carlsbad (@assaf_carlsbad).

Decoder

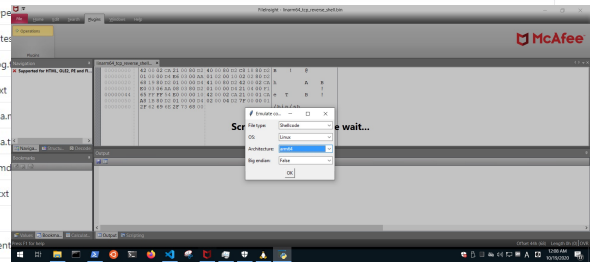
nmantani / FileInsight-plugins

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights

master | 1 branch | 16 tags | Go to file | Add file | Code

nmantani Use "py.exe --list" instead of hard-coded paths to check Python 3 ins... 63be4a5 2 days ago 214 commits

docs	- Add description of the "Parse file structure" plugin.	2 months ago
plugins/Ope		
.gitattributes		
ChangeLog.t		
LICENSE.txt		
README.ja.r		
README.ja.t		
README.md		
README.txt		
install.ps1		
requirement		



README.md

FileInsight-plugins: a decoding toolbox of McAfee FileInsight hex editor for malware analysis

FileInsight-plugins is a collection of plugins for McAfee FileInsight hex editor. It adds many capabilities such as decryption, decompression, searching XOR-ed text strings, scanning with a YARA rule, and more! It is useful for various kind of decoding tasks in malware analysis (e.g. extracing malware executables and decoy documents from malicious document files).

VAC3 Emulator

ioncodes / vacation3-emu

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights

master | 1 branch | 0 tags | Go to file | Add file | Code

ioncodes added link 0bb1447 on Sep 29 3 commits

images	added images and readme	last month
.gitignore	init	last month
README.md	added link	last month
emu.py	init	last month
typedefs.h	init	last month

README.md

vacation3

An emulator powered by Qiling to deobfuscate/decrypt VAC3 modules.

IoT Emulator

MacOS Emulator

Binary Fuzzer

IoT Fuzzer

Malware Sandbox

CTF Solver

IOS Emulator

Binary Decrypt

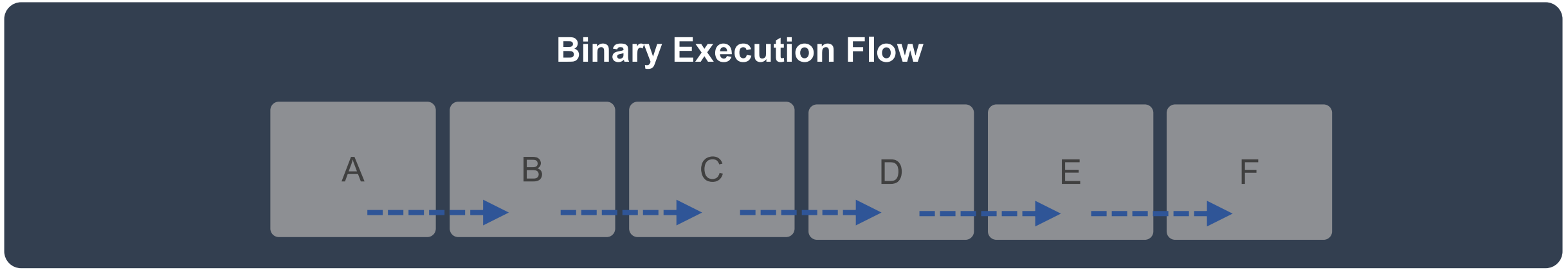
Qiling Framework

CPU Architecture | Loader | OS | Debugger | Extensions

Instrumentation (Qiling's API)

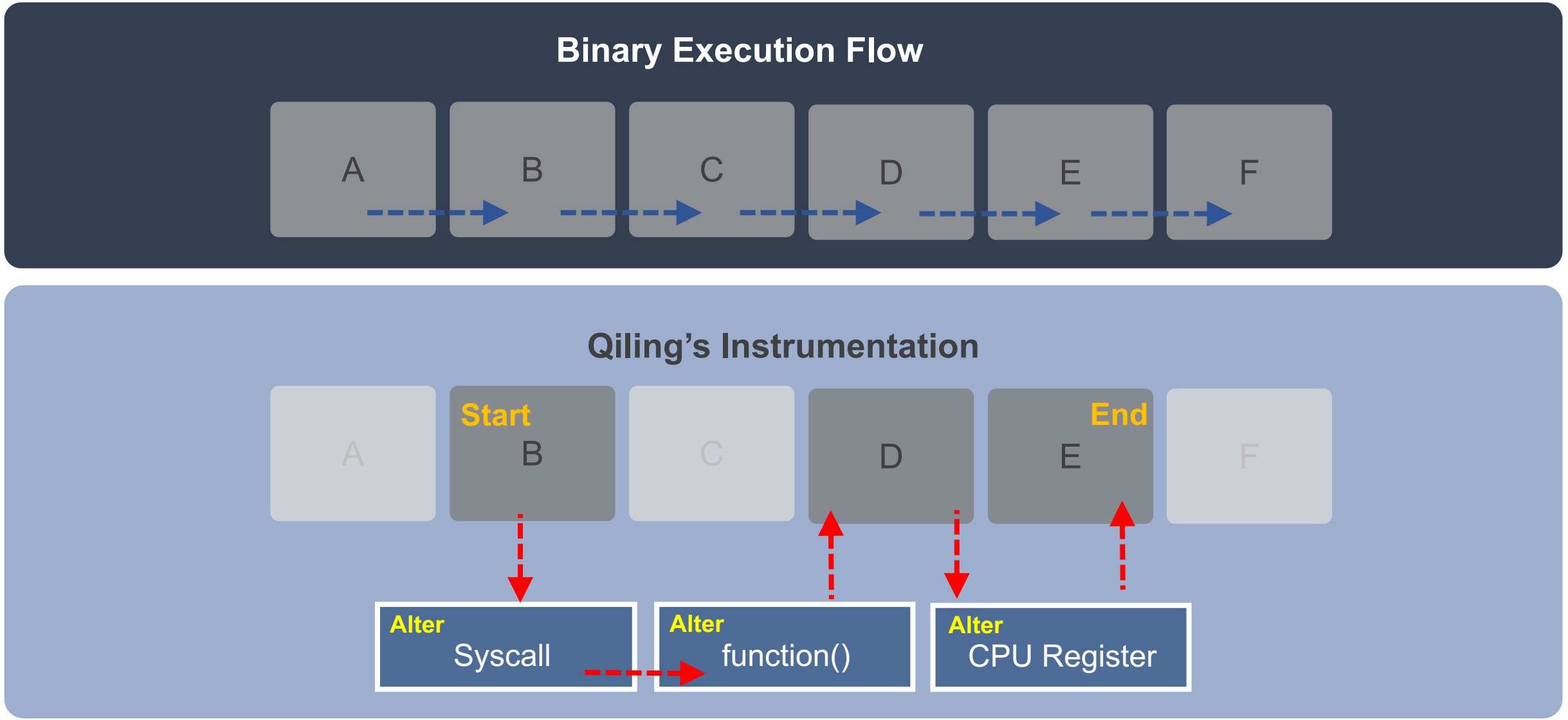
Instrumentation

What Is Instrumentation



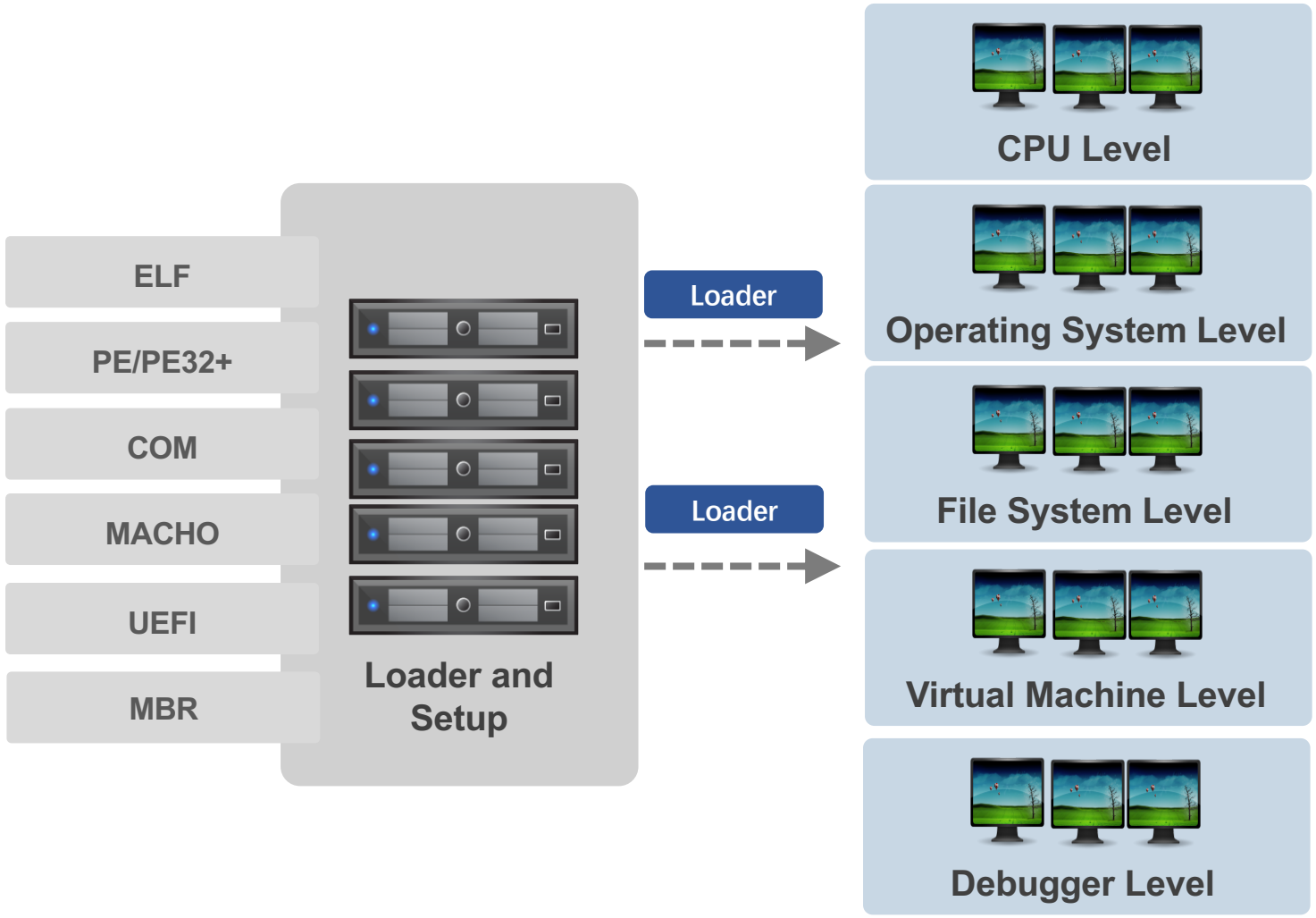
One Function After Another

What Is Instrumentation



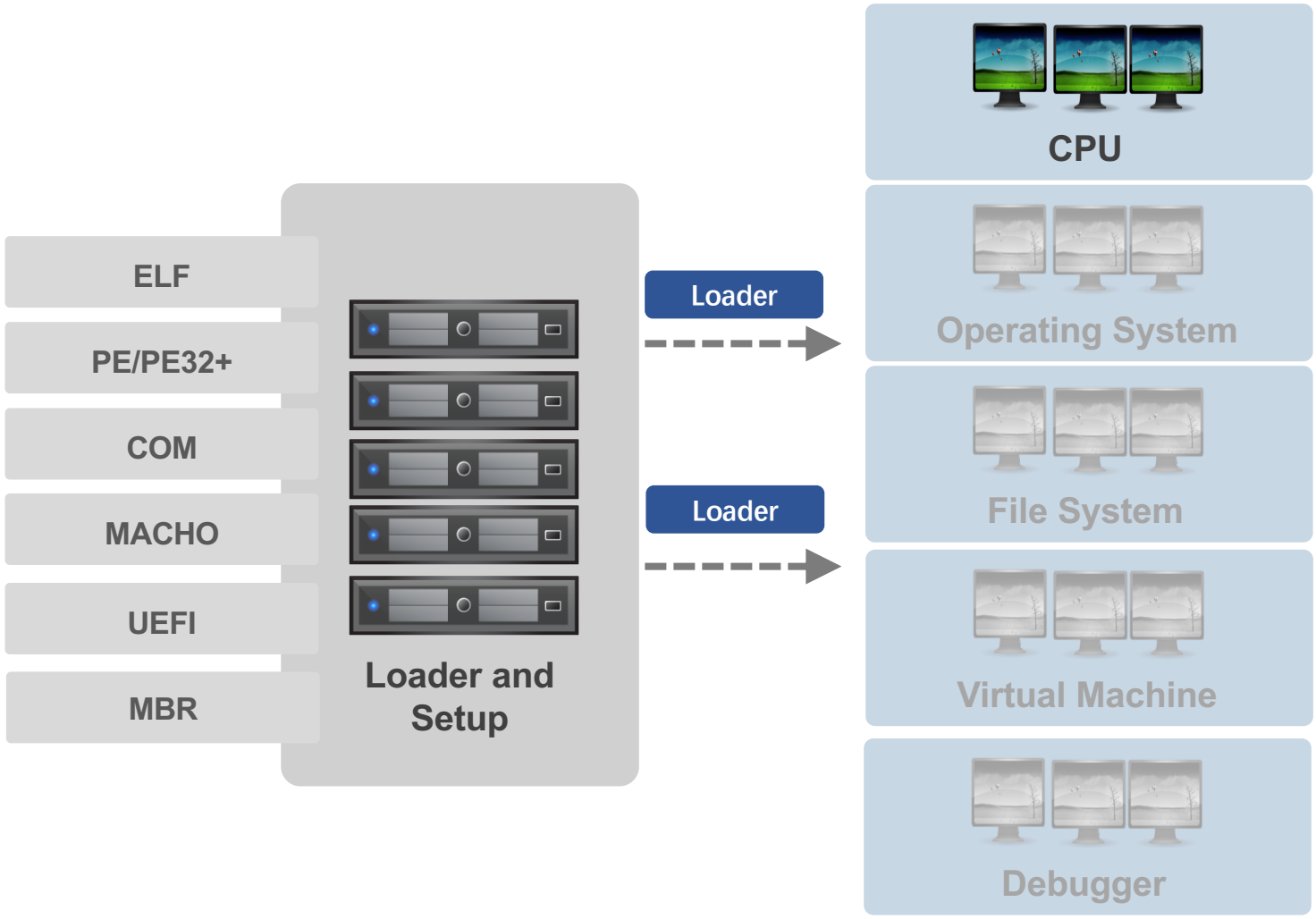
Qiling and APIs

Qiling Framework and Its Interface



More APIs: https://docs.qiling.io

CPU Instrumentation



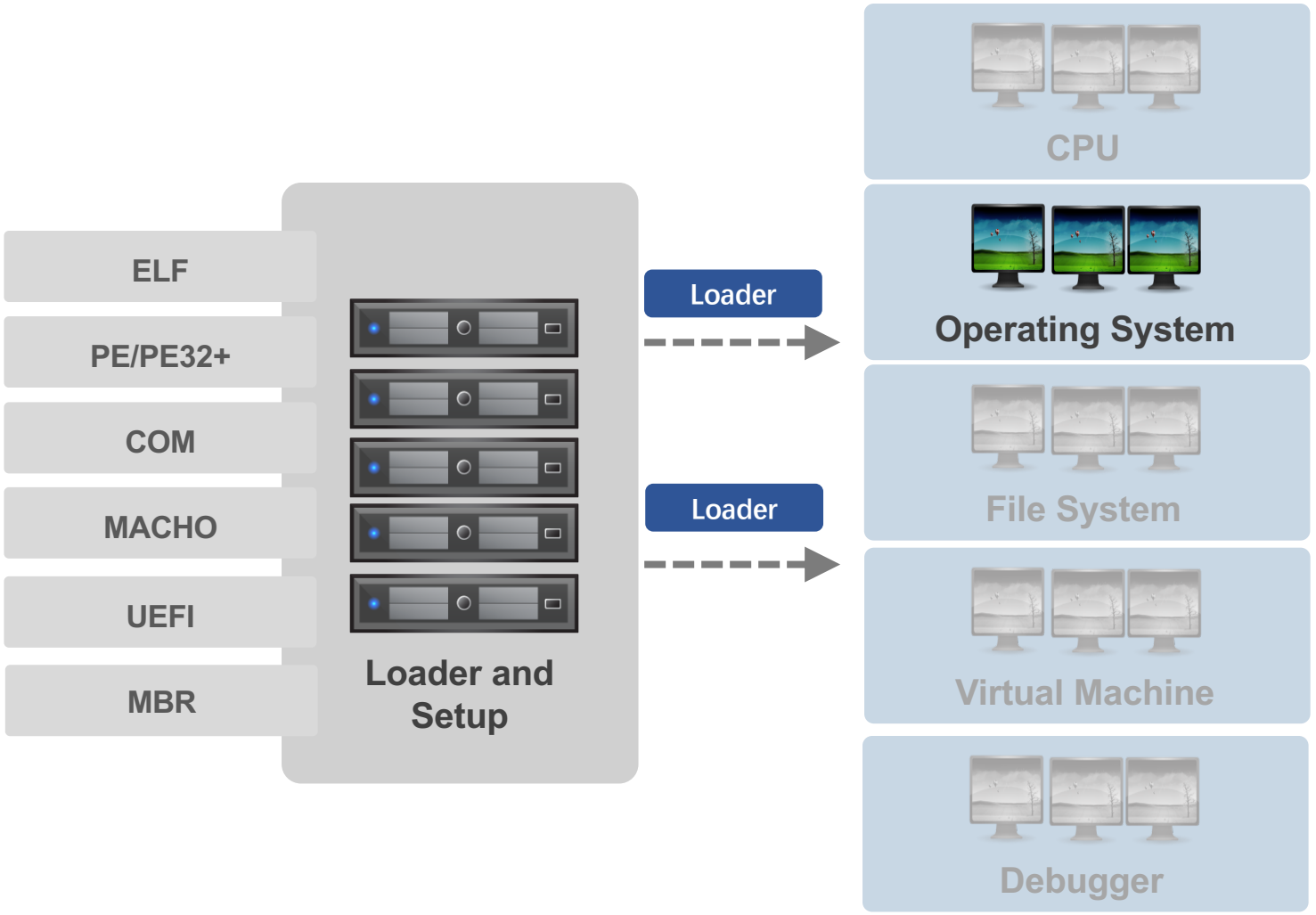
- Access to Register
- Reading register
  - ql.reg.eax
- Writing to register
  - ql.reg.eax = 0x41
- Differet Hooks
  - ql.hook_code()
  - ql.hook_address()
  - and more

CPU Instrumentation: Examples

```
def my_puts(ql):
    addr = ql.os.function_arg[0]
    print("puts(%s)" % ql.mem.string(addr))
    reg = ql.reg.read("rax")
    print("reg : 0x%x" % reg)
    ql.reg.rax = reg
    self.set_api = reg

def write_onEnter(ql, arg1, arg2, arg3, *args):
    print("enter write syscall!")
    ql.reg.rsi = arg2 + 1
    ql.reg.rdx = arg3 - 1
    self.set_api_onenter = True

def write_onexit(ql, arg1, arg2, arg3, *args):
    print("exit write syscall!")
    ql.reg.rax = arg3 + 1
    self.set_api_onexit = True

ql = Qiling(["../examples/rootfs/x8664_linux/bin/x8664_args","1234test", "12345678", "bin/x8664_hello"],  "../e
ql.set_syscall(1, write_onEnter, QL_INTERCEPT.ENTER)
ql.set_api('puts', my_puts)
ql.set_syscall(1, write_onexit, QL_INTERCEPT.EXIT)
ql.mem.map(0x1000, 0x1000)
ql.mem.write(0x1000, b"\xFF\xFE\xFD\xFC\xFB\xFA\xFB\xFC\xFC\xFE\xFD")
ql.mem.map(0x2000, 0x1000)
ql.mem.write(0x2000, b"\xFF\xFE\xFD\xFC\xFB\xFA\xFB\xFC\xFC\xFE\xFD")
ql.run()
```

- Access to Register
- Reading register
  - eax = ql.reg.eax
- Writing to register
  - ql.reg.eax = 0x41
- Differet Hooks
  - ql.hook_code()
  - ql.hook_address()
  - and more

Operating System Instrumentation



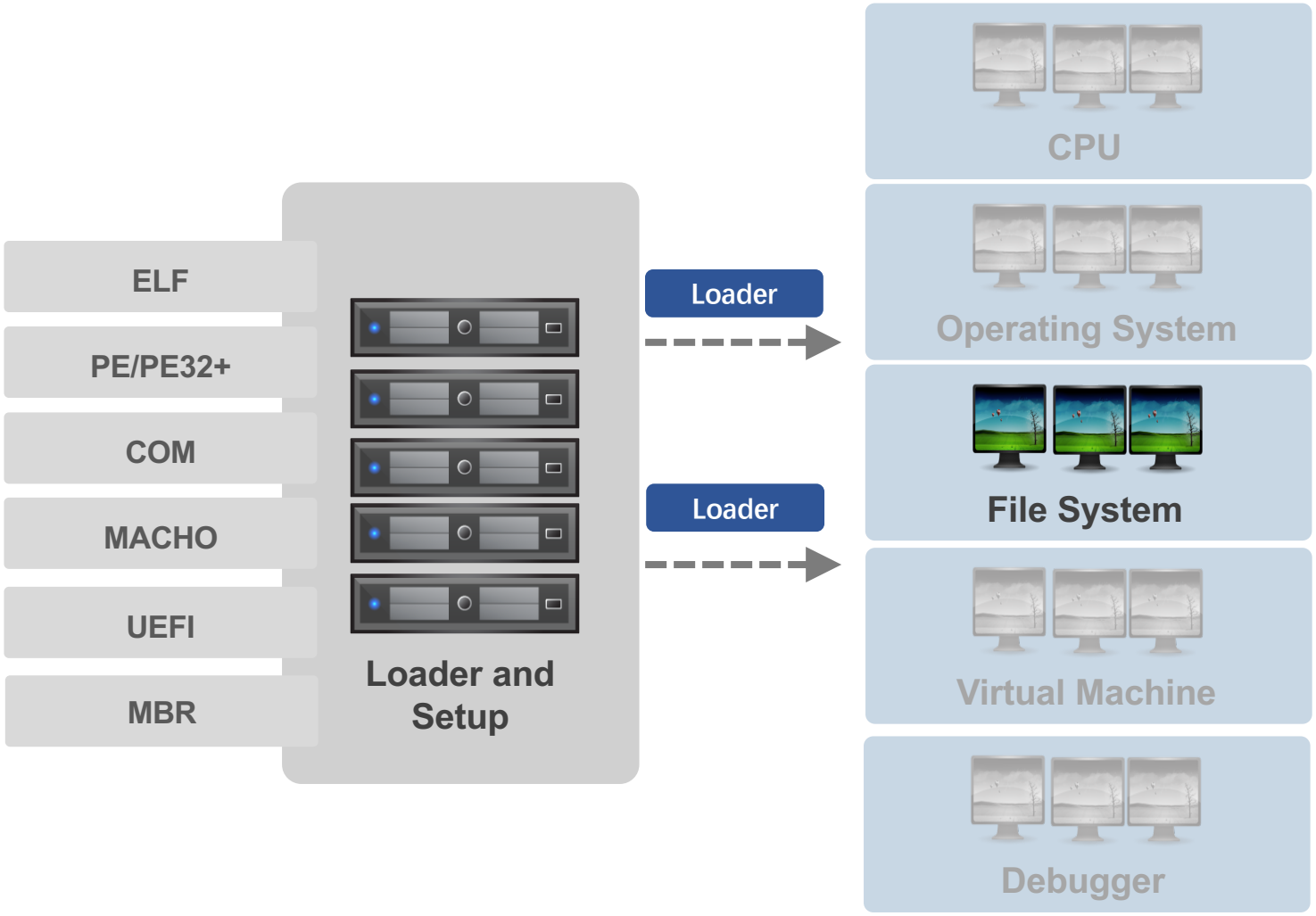
- Access to memory
  - ql.mem.read()
  - ql.mem.write()
- Search pattern from memory
  - ql.mem.search()
- Stack related operation
  - ql.stack_pop
  - ql.stack_push
- Syscall replacement
  - ql.set_syscall()
  - ql.set_api()
- Replace library call with
  - ql.set_api()

Example: Operating System

```python
from qiling import *

def my_syscall_write(ql, write_fd, write_buf, write_count, *args, **kw):
    regreturn = 0

    try:
        buf = ql.mem.read(write_buf, write_count)
        ql.nprint("\n+++++++++\nmy write(%d,%x,%i) = %d\n+++++++++" % (write_fd, write_buf, write_count, regreturn))
        ql.os.fd[write_fd].write(buf)
        regreturn = write_count
    except:
        regreturn = -1
        ql.nprint("\n+++++++++\nmy write(%d,%x,%i) = %d\n+++++++++" % (write_fd, write_buf, write_count, regreturn))
        if ql.output in (QL_OUTPUT.DEBUG, QL_OUTPUT.DUMP):
            raise

    ql.os.definesyscall_return(regreturn)


if __name__ == "__main__":
    ql = Qiling(["rootfs/arm_linux/bin/arm_hello"], "rootfs/arm_linux", output = "debug")
    # Custom syscall handler by syscall name or syscall number.
    # Known issue: If the syscall func is not be implemented in qiling, qiling does
    # not know which func should be replaced.
    # In that case, you must specify syscall by its number.
    ql.set_syscall(0x04, my_syscall_write)

    # set syscall by syscall name
    #ql.set_syscall("write", my_syscall_write)

    ql.run()
```

- Access to memory
  - ql.mem.read()
  - ql.mem.write()
- Search pattern from memory
  - ql.mem.search()
- Stack related operation
  - ql.stack_pop
  - ql.stack_push
- Syscall replacement
  - ql.set_syscall()
  - ql.set_api()
- Replace library call with
  - ql.set_api()

File System Instrumentation



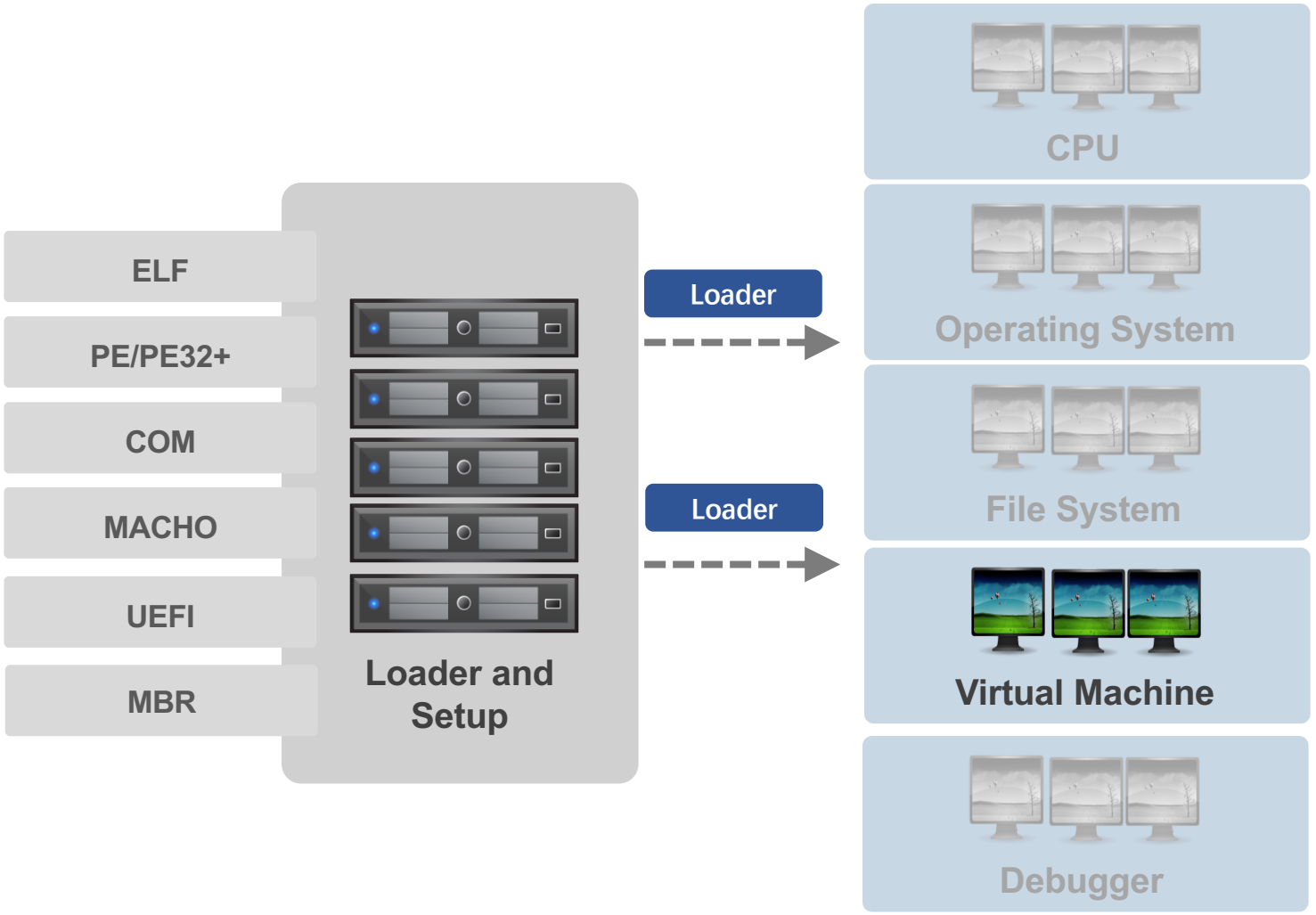
- Map host file
  - ql.fs_mapper()
- Hijack accessed file
  - ql.fs_mapper(hijack_func)
- Stdio replacement
  - stdin
  - stdout
  - stderr
- Patch file's memory before execution
  - ql.patch

Example: File System Instrumentation

```python
from qiling import *
from qiling.os.mapper import QlFsMappedObject

class Fake_urandom(QlFsMappedObject):

    def read(self, size):
        return b"\x01" # fixed value for reading /dev/urandom

    def fstat(self): # syscall fstat will ignore it if return -1
        return -1

    def close(self):
        return 0

if __name__ == "__main__":
    ql = Qiling(["rootfs/x86_linux/bin/x86_fetch_urandom"], "rootfs/x86_linux")
    ql.add_fs_mapper("/dev/urandom", Fake_urandom())
    ql.run()
```

- Map host file
  - ql.fs_mapper()
- Hijack accessed file
  - ql.fs_mapper(hijack_func)
- Stdio replacement
  - stdin
  - stdout
  - stderr
- Patch file's memory before execution
  - ql.patch

Virtual Machine Instrumentation



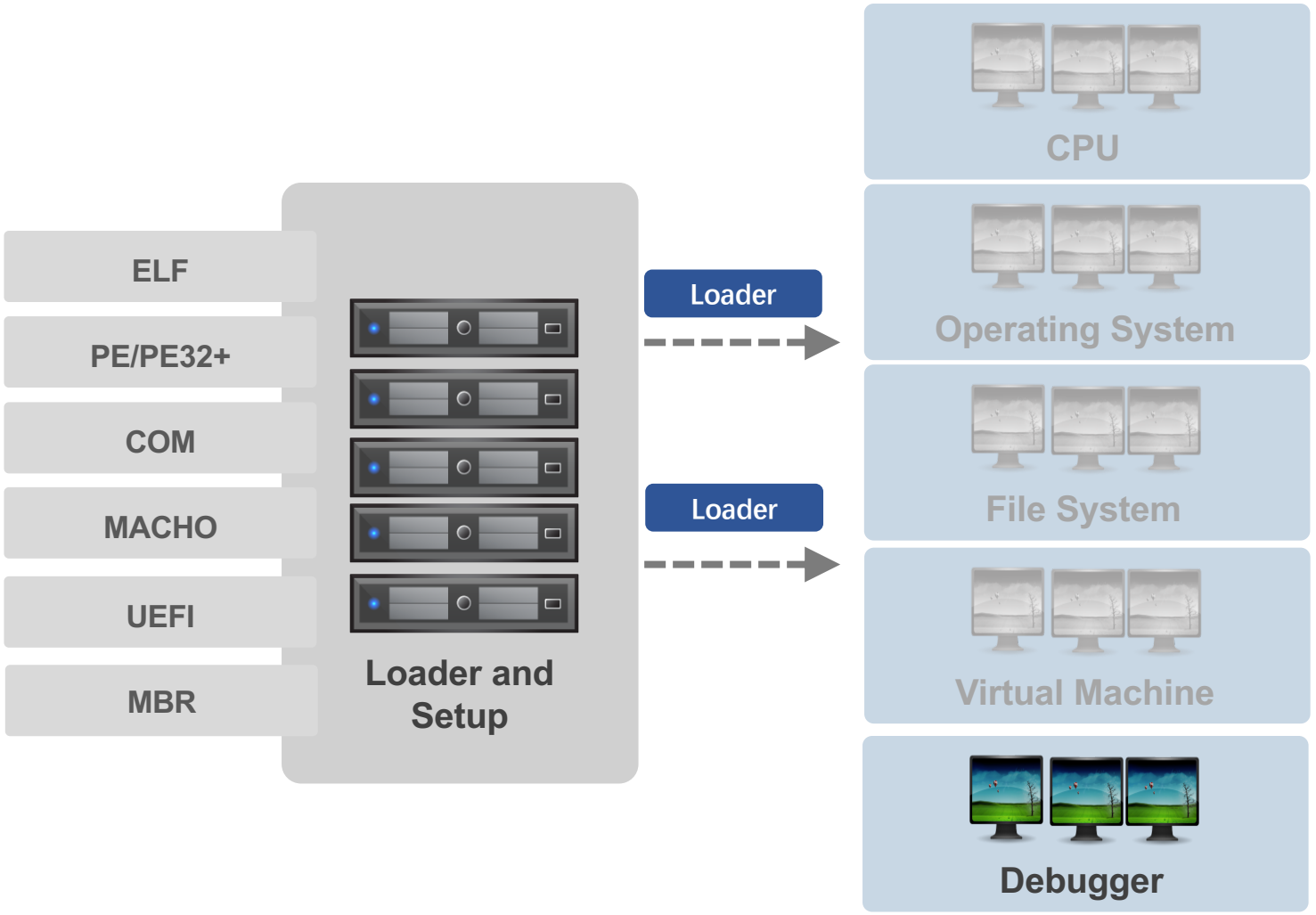
- Save current state
  - ql.save()
- Restore current state
  - ql.restore()
- Save/restore memory only
  - ql.mem.save()
- Save/restore register only
  - ql.reg.save()

Example: Virtual Machine Instrumentation

```python
def save_context(ql, *args, **kw):
    ql.save(cpu_context=False, snapshot="snapshot.bin")

def patcher(ql):
    br0_addr = ql.mem.search("br0".encode() + b'\x00')
    for addr in br0_addr:
        ql.mem.write(addr, b'lo\x00')


def check_pc(ql):
    print("=" * 50)
    print("[!] Hit fuzz point, stop at PC = 0x%x" % ql.reg.arch_pc)
    print("=" * 50)
    ql.emu_stop()

def my_sandbox(path, rootfs):
    ql = Qiling(path, rootfs, output="debug", verbose=5)
    ql.add_fs_mapper("/dev/urandom","/dev/urandom")
    ql.hook_address(save_context ,0x10930)
    ql.hook_address(patcher, ql.loader.elf_entry)
    ql.hook_address(check_pc,0x7a0cc)
    ql.run()

if __name__ == "__main__":
    nvram_listener_therad =  threading.Thread(target=nvram_listener, daemon=True)
    nvram_listener_therad.start()
    my_sandbox(["rootfs/bin/httpd"], "rootfs")
```

- Save current state
  - ql.save()
- Restore current state
  - ql.restore()
- Save/restore memory only
  - ql.mem.save()
- Save/restore register only
  - ql.reg.save()

Debugger



- Open API for RSP compatible Debugger
- Build In debugger – Qdbg
  - Able to reverse debug

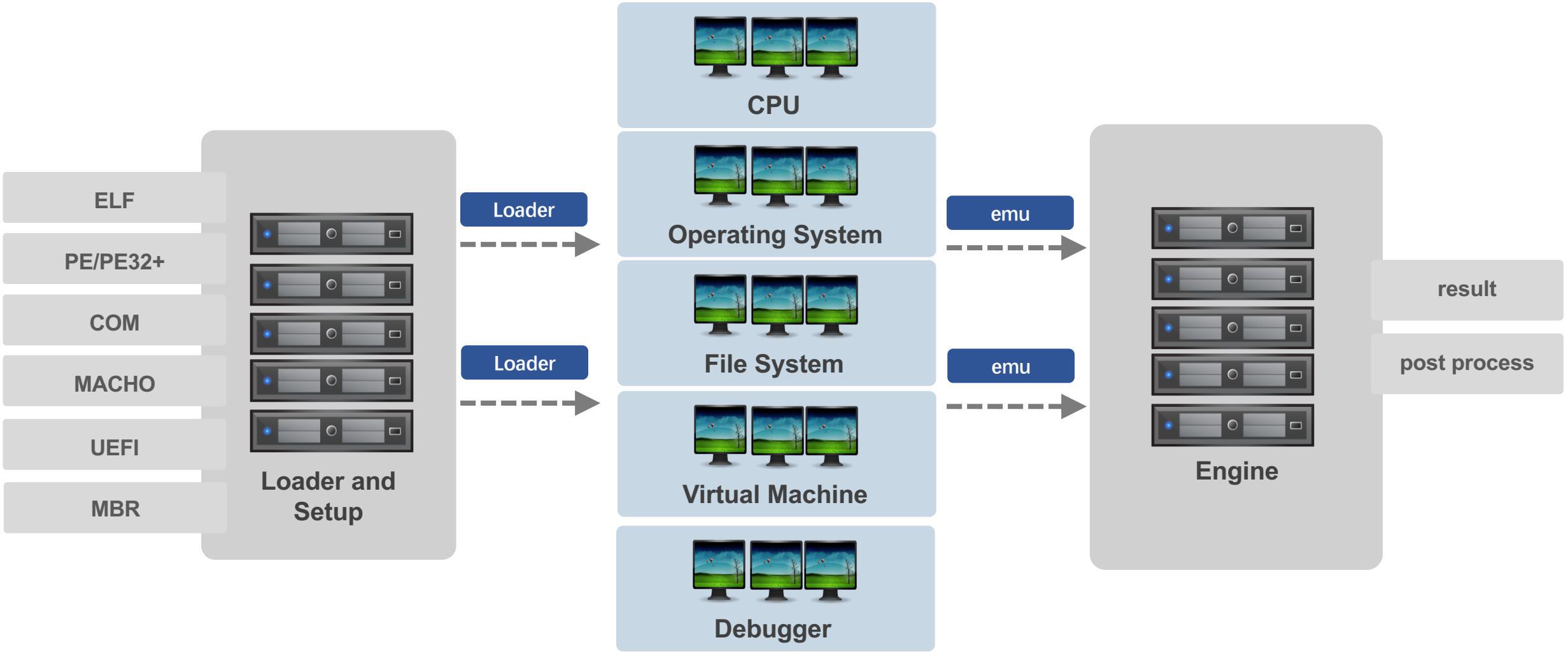
Example: Debugger

```python
def run_sandbox(path, rootfs, output):
    ql = Qiling(path, rootfs, output = output)
    ql.multithread = False
    ql.debugger = "qdb:rr" # switch on record and replay with rr
    # ql.debugger = "qdb:" # enable qdb without options
    ql.run()


if __name__ == "__main__":
    run_sandbox(["rootfs/arm_linux/bin/arm_hello"], "rootfs/arm_linux", "debug")
```

- Open API for RSP compatible Debugger
- Build In debugger – Qdbg
  - Able to reverse debug

```python
from qiling import *

if __name__ == "__main__":
    ql = Qiling(["rootfs/x8664_linux/bin/x8664_hello"], "rootfs/x8664_linux", output = "debug")
    ql.debugger = "gdb:0.0.0.0:9999"
    ql.run()
```

Qiling Framework: In a Nutshell



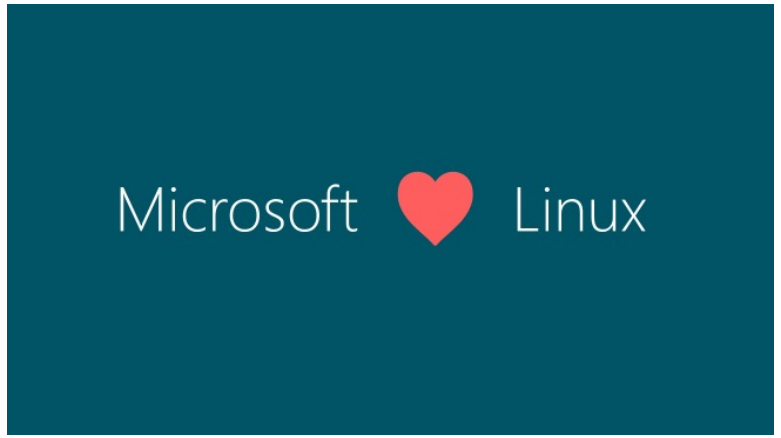
Base OS can be Windows/Linux/BSD or OSX

And not limited to ARCH

Demo

Demo Setup

- **ONLY If you wish to try yourself**



- Required OS
  - Ubuntu 18.04 / 20.04
  - WSL2

- Install Qiling Framework
  - sudo apt-get update
  - sudo apt-get upgrade
  - sudo apt install python3-pip git cmake build-essential libtool-bin python3-dev automake flex bison libglib2.0-dev libpixman-1-dev clang python3-setuptools llvm
  - pip3 install --user https://github.com/qilingframework/qiling/archive/dev.zip
  - https://github.com/kabeor/Traning-Examples

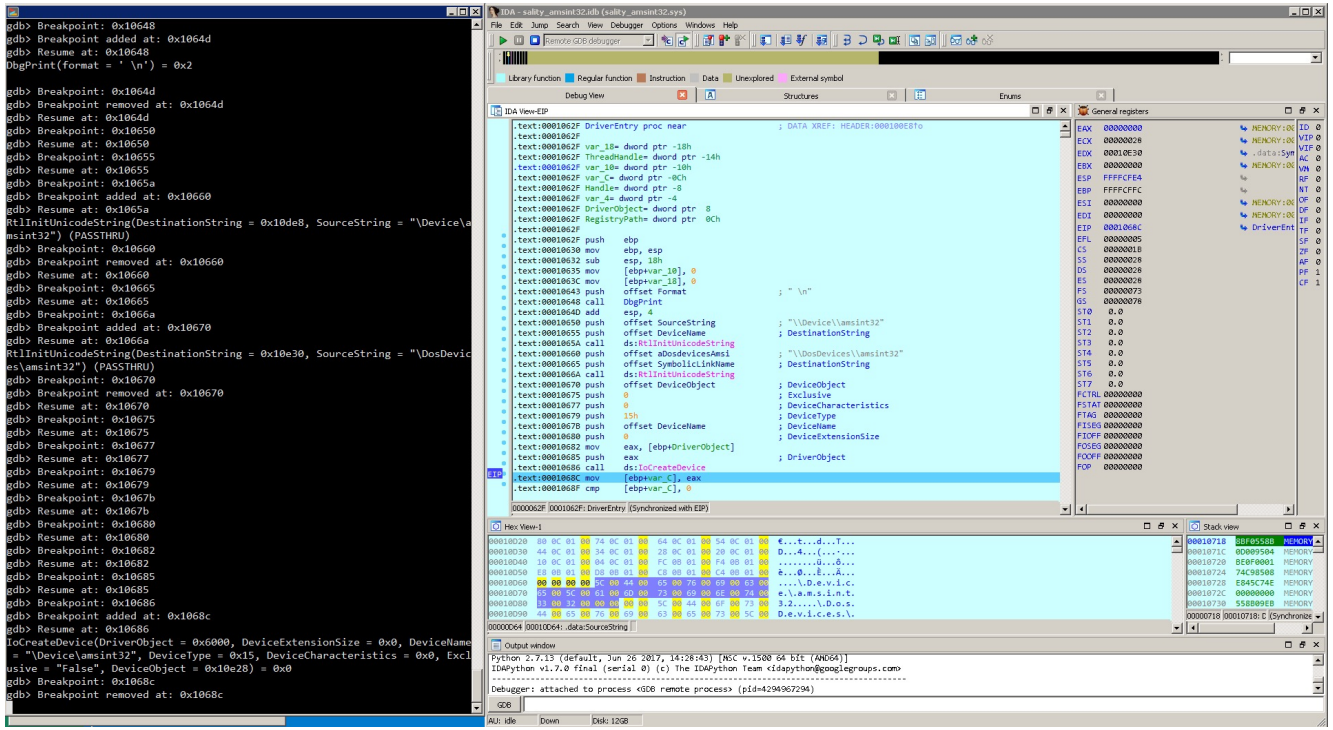
- Install AFL++
  - cd AFLplusplus
  - make
  - cd unicorn_mode
  - ./build_unicorn_support.sh

Malware & Rootkit Analysis

- Support Both Win32/64
- Support PE and System Driver
- Anti-Anti Debug
- Script able
- Cross platform support

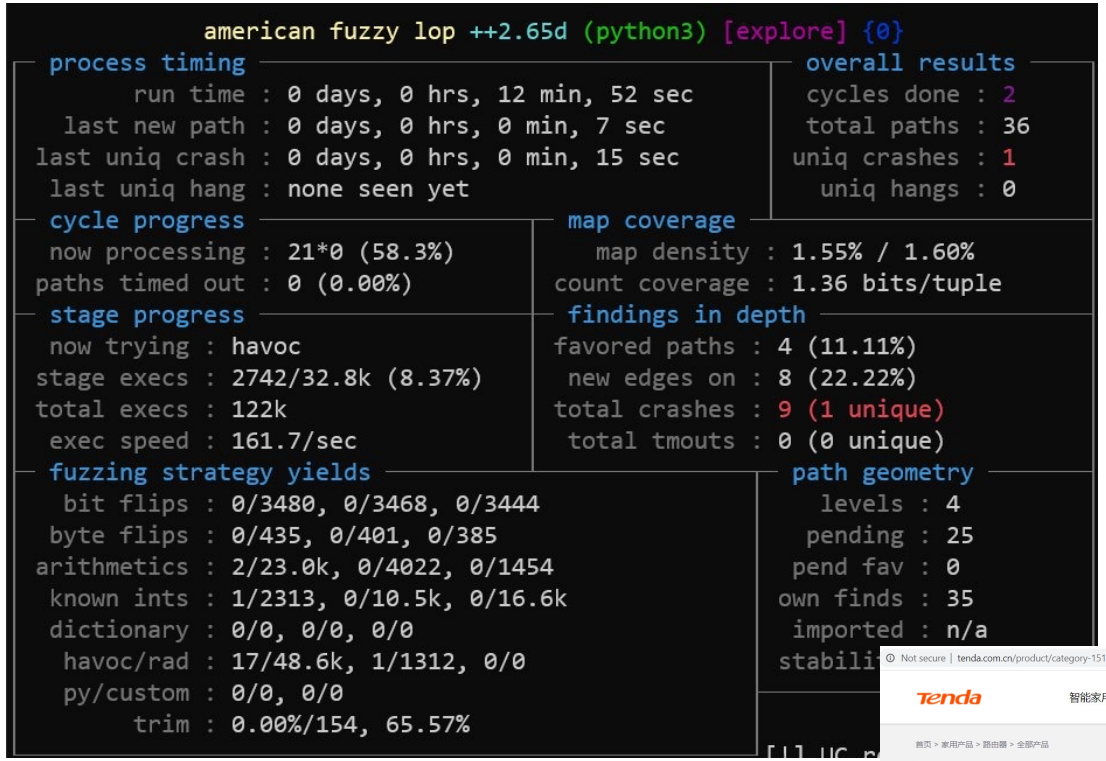
```
root@7fd8b304fcec:/qilingdev# python3 examples/one.py samples_anyrun/al-khaser.bin 2>&1
[al-khaser version 0.67]
------------------------[Debugger Detection]-------------------------
[*] Checking IsDebuggerPresent API () [=] GOOD
[*] Checking PEB.BeingDebugged [=] GOOD
[*] Checking CheckRemoteDebuggerPresentAPI () [=] GOOD
[*] Checking PEB.NtGlobalFlag [=] GOOD
[*] Checking ProcessHeap.Flags [=] GOOD
[*] Checking ProcessHeap.ForceFlags [=] GOOD
[*] Checking NtQueryInformationProcess with ProcessDebugPort [=] GOOD
[*] Checking NtQueryInformationProcess with ProcessDebugFlags [=] GOOD
[*] Checking NtQueryInformationProcess with ProcessDebugObject [=] GOOD
[*] Checking NtSetInformationThread with ThreadHideFromDebugger [=] BAD
[*] Checking CloseHandle with an invalide handle [=] GOOD
[*] Checking UnhandledExcepFilterTest [=] GOOD
[*] Checking OutputDebugString [=] GOOD
[*] Checking Hardware Breakpoints [=] GOOD
[*] Checking Software Breakpoints [=] GOOD
[*] Checking Interupt 0x2d [=] BAD
We are finally done
root@7fd8b304fcec:/qilingdev#
```

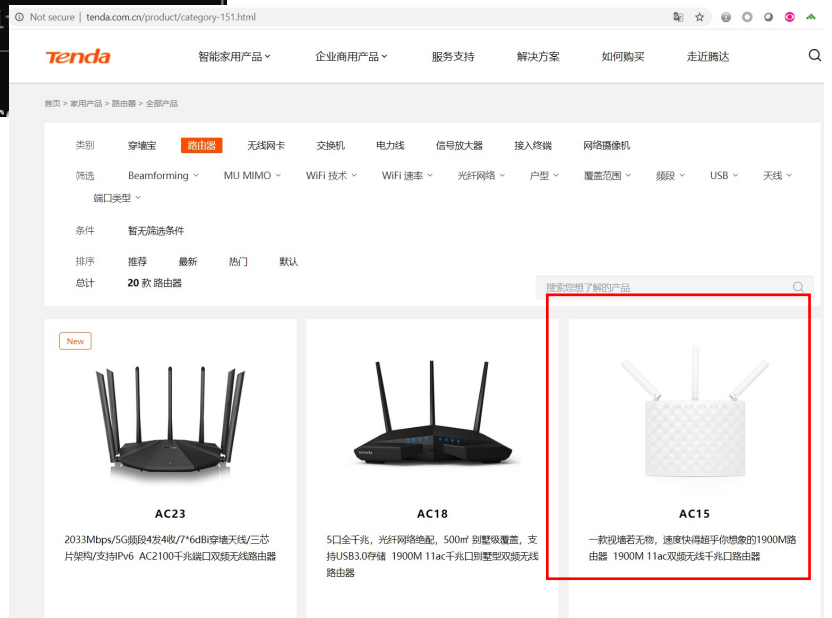
```
(22:43:04):xwings@bespin:<~/projects/qiling>
(15)$ python3 qltool run -f examples/rootfs/x86_windows/bin/al-khaser.bin --rootfs j
xamples/rootfs/x86_windows
[+] Loading examples/rootfs/x86_windows/bin/al-khaser.bin to 0x400000
[+] PE entry point at 0x403d6a
[+] Initiate stack address at 0xfffdd000
[+] TEB addr is 0x6000
[+] PEB addr is 0x6044
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/kernel32.dll to 0x10000000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/kernel32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/user32.dll to 0x100d4000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/user32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/advapi32.dll to 0x1019d000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/advapi32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/ole32.dll to 0x1023e000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/ole32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/oleaut32.dll to 0x1039a000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/oleaut32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/shlwapi.dll to 0x1042c000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/shlwapi.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/setupapi.dll to 0x10470000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/setupapi.dll
[+] Done with loading examples/rootfs/x86_windows/bin/al-khaser.bin
GetSystemTimeAsFileTime(lpSystemTimeAsFileTime = 0xffffcfec)
GetCurrentThreadId() = 0x0
GetCurrentProcessId() = 0x2005
QueryPerformanceCounter(lpPerformanceCount = 0xffffcfe4) = 0x0
IsProcessorFeaturePresent(ProcessorFeature = 0xa) = 0x1
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/api-ms-win-core-synch-l1-2
0.dll to 0x108b9000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/api-ms-win-core-
ynch-l1-2-0.dll
LoadLibraryExW(lpLibFileName = "api-ms-win-core-synch-l1-2-0", hFile = 0x0, dwFlags
 0x800) = 0x108b9000
GetProcAddress(hModule = 0x108b9000, lpProcName = "InitializeCriticalSectionEx") = 0
0
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x424d64, dwSpinCount = 0x
a0) = 0x1
[+] Loading jexamples/rootfs/x86_windows/Windows/SysWOW64/api-ms-win-core-fibers-l1-
-1.dll to 0x108bc000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SysWOW64/api-ms-win-core-
ibers-l1-1-1.dll
LoadLibraryExW(lpLibFileName = "api-ms-win-core-fibers-l1-1-1", hFile = 0x0, dwFlags
 = 0x800) = 0x108bc000
GetProcAddress(hModule = 0x108bc000, lpProcName = "FlsAlloc") = 0x0
TlsAlloc() = 0x0
GetProcAddress(hModule = 0x108bc000, lpProcName = "FlsSetValue") = 0x0
TlsSetValue(dwTlsIndex = 0x0, lpTlsValue = 0x424d3c) = 0x1
LoadLibraryExW(lpLibFileName = "api-ms-win-core-synch-l1-2-0", hFile = 0x0, dwFlags
 0x800) = 0x108b9000
GetProcAddress(hModule = 0x108b9000, lpProcName = "InitializeCriticalSectionEx") = 0
0
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x425380, dwSpinCount = 0x
a0) = 0x1
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x425398, dwSpinCount = 0x
a0) = 0x1
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x4253b0, dwSpinCount = 0x
a0) = 0x1
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x4253c8, dwSpinCount = 0x
a0) = 0x1
```



Fuzzer

- Required Firmware
  - AC15
- Run Tenda AC15
  - start_tendaac15_httpd.py
  - Test with browser
- Check crash point
  - addressNat_overflow.sh
- How to find and save snapshot
  - saver_tendaac15_httpd.py
- How to build and run fuzzer
  - fuzz_tendaac15_httpd.py

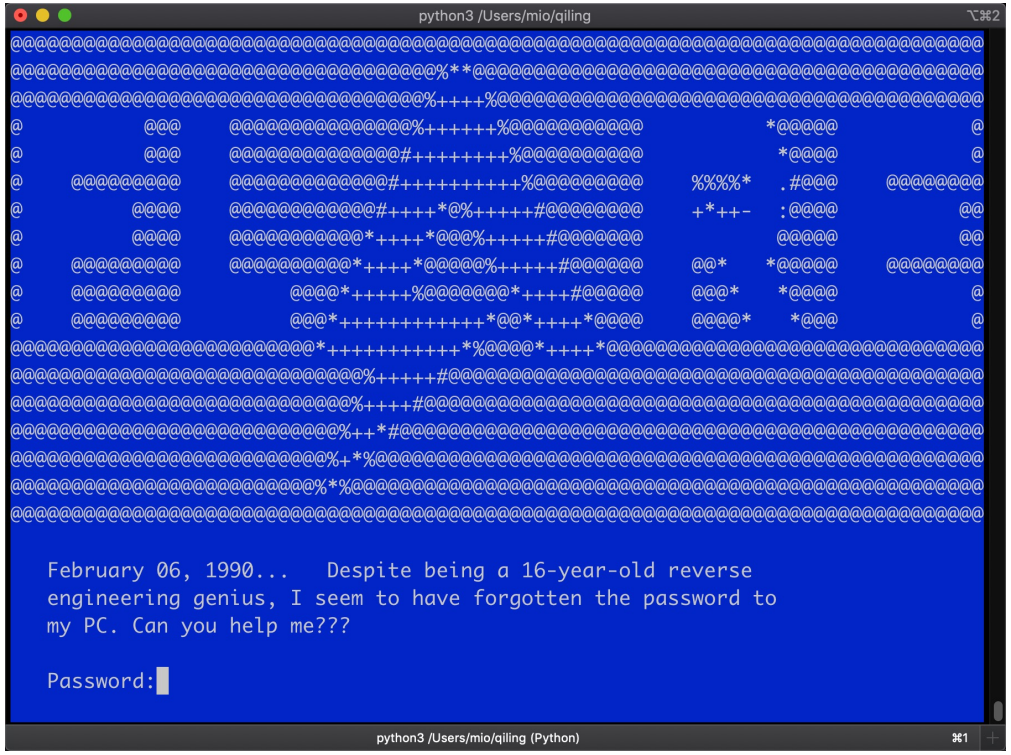


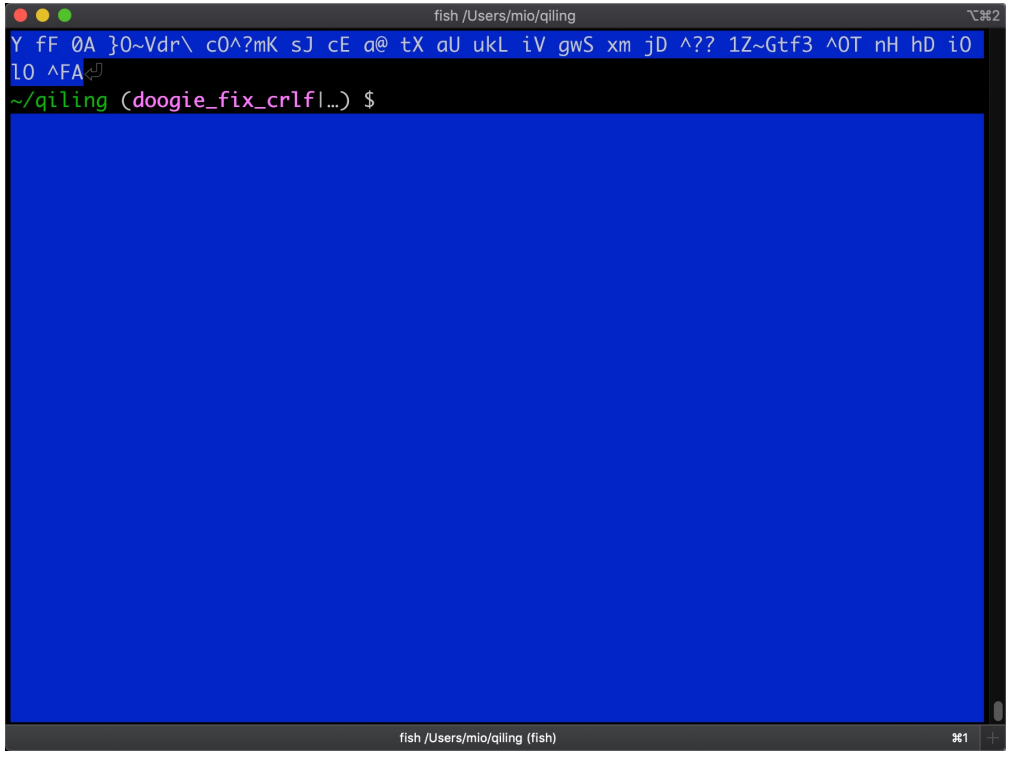


MBR Analysis

- Sample:
  - Flare-On 5 (2018) Challenge 8 - doogie
- MBR file
- Quick look by qltool.
  - python3 qltool run -f examples/rootfs/8086/doogie/doogie.bin --rootfs examples/rootfs/8086/ --console False
- Try some inputs, but only get gibberish.
- Tips: Feburary 06, 1990.

```
~/q/e/r/8/doogie (doogie|…) $ file doogie.bin
doogie.bin: DOS/MBR boot sector; partition 1 : ID=0x7, activ
e, start-CHS (0x0,32,33), end-CHS (0x3ff,254,63), startsecto
r 2048, 41938944 sectors
~/q/e/r/8/doogie (doogie|…) $
```





IDA Plugin

Setup
Reload User Scripts

Execute Till
Execute Selection
Continue
Set PC
Step ^⇧F9
Edit Register

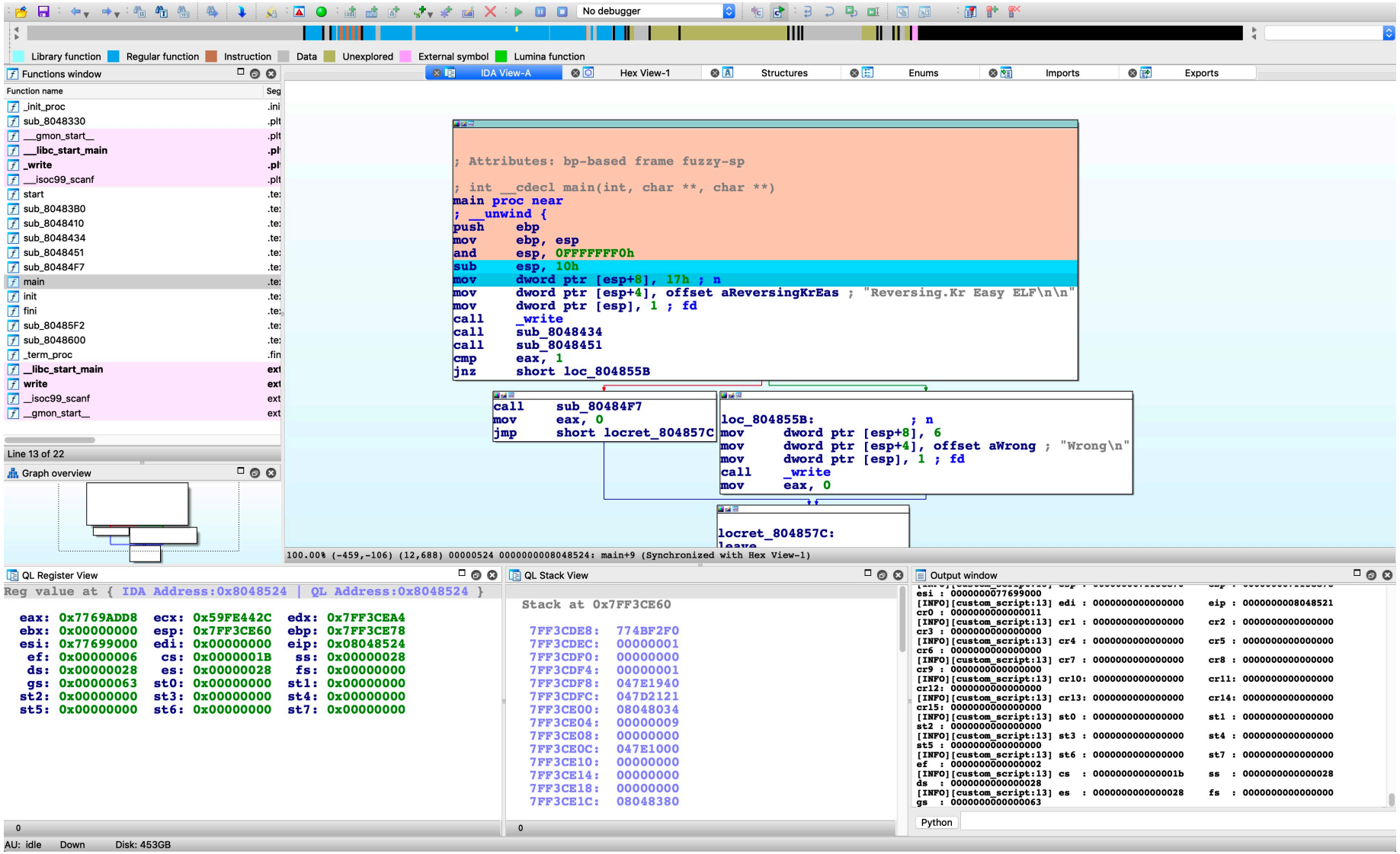
Restart

View Register
View Stack
View Memory

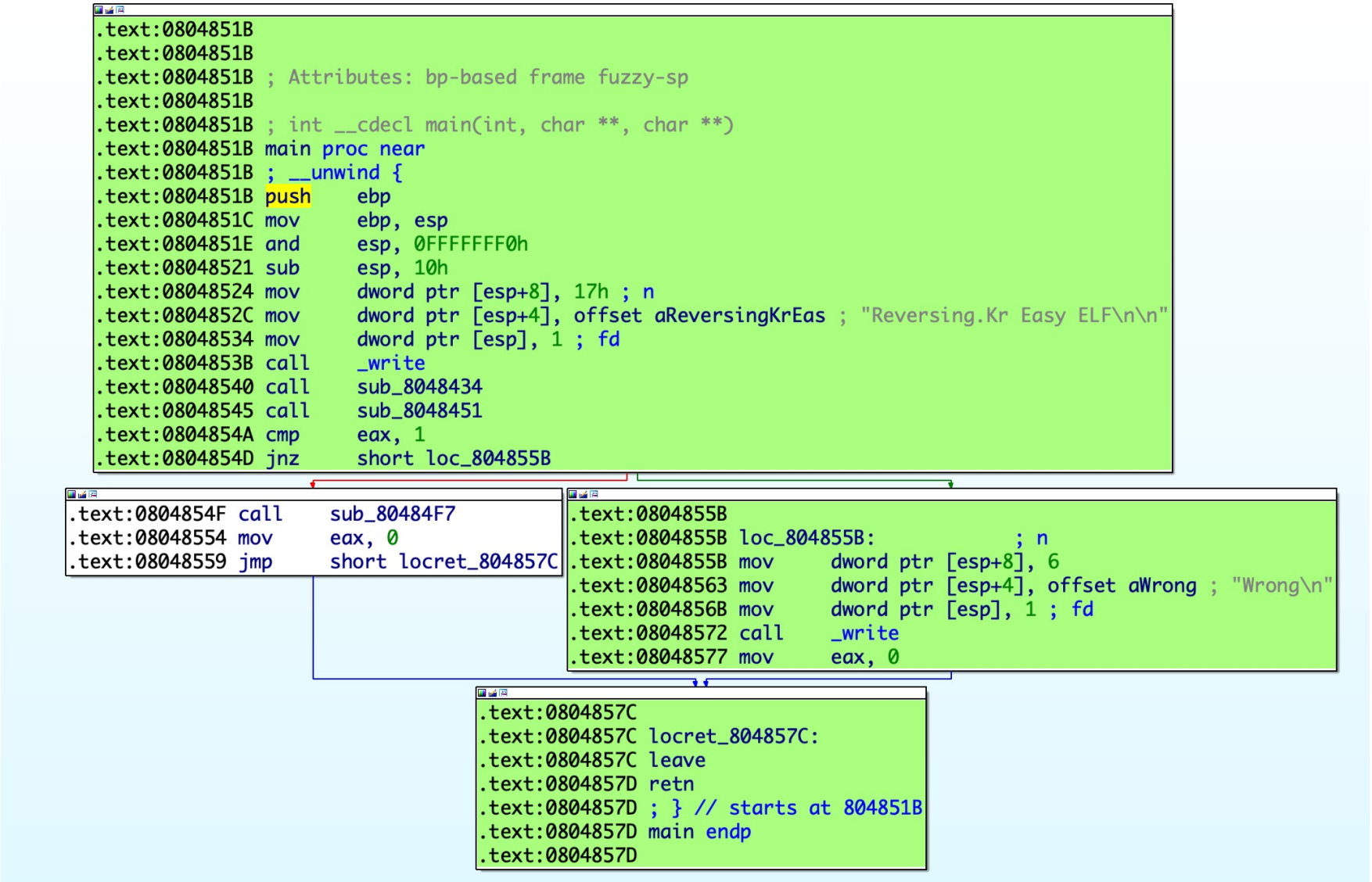
Save Snapshot
Load Snapshot

Auto Analysis For Deflat
Mark as Real Block
Mark as Fake Block
Mark as Return Block
Deflat

Remove Junk Code by Patterns
Nop Items without Color



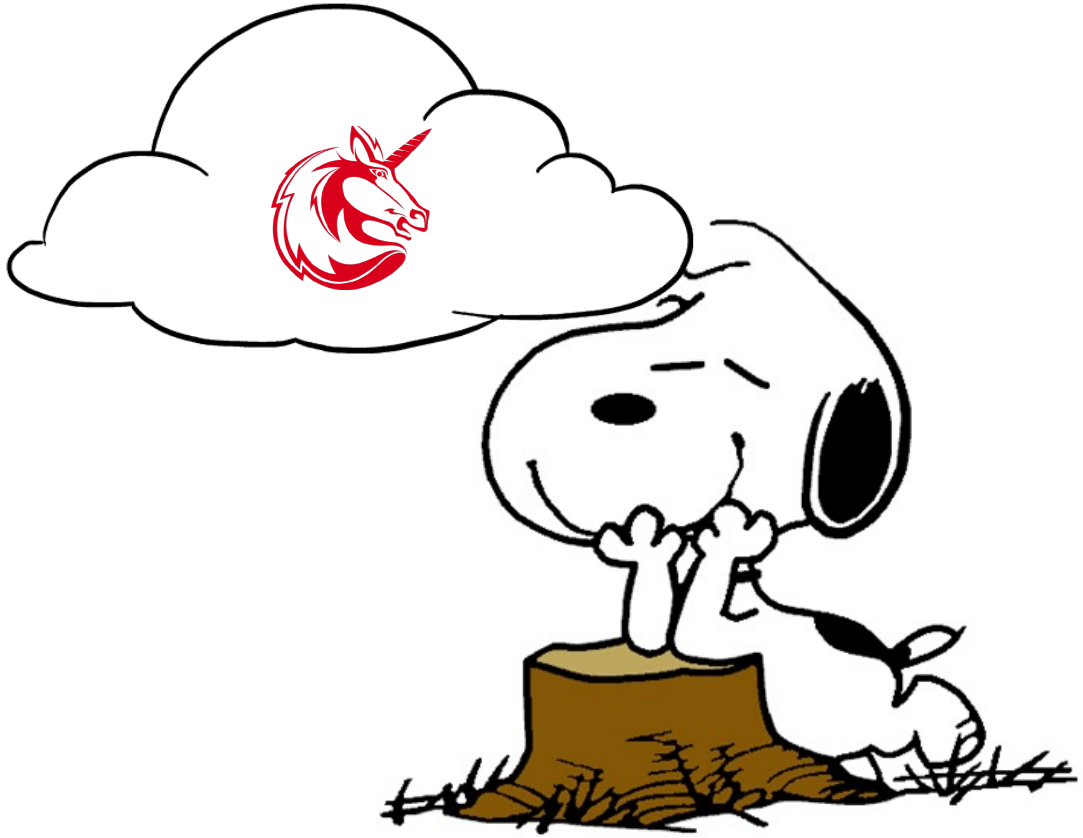
Execution path drawing



Next Step

Roadmap

- Force Unicorn Engine sync with QEMU 5
  - More architectures, more CPU instructions set
- Android Java bytecode layer instrumentation
- IOS emulation support
- More robust Windows emulation
  - Introduce wine && Cygwin or something
- Smart Contract emulation (EVM, WASM)
- MCU emulation

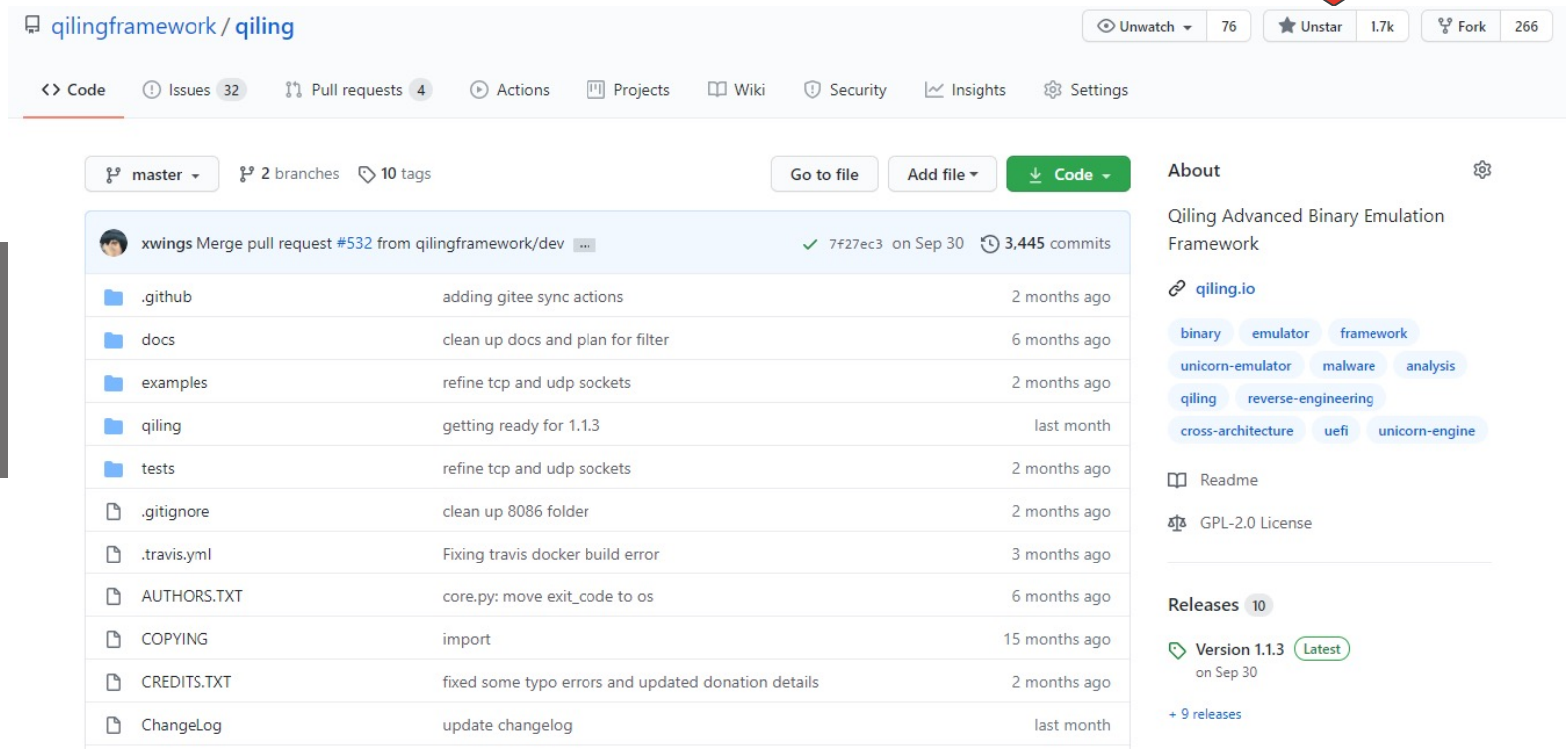


Join Us and Make Pull Request !!!

Everything Else

- About Qiling Framework
  - https://qiling.io
  - https://github.com/qilingframework/qiling
  - https://docs.qiling.io
  - http://t.me/qilingframework
  - @qiling_io

Star us



Questions