

Qiling Framework: Modern Reverse Engineering Experience

November, 2020



weibo: @qiling_io

QQ: 486812017

twitter: @qiling_io <https://qiling.io>

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

About xwings



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



HACKERSBADGE.COM

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

- > Cross platform and cross architecture binary instrumentation framework
- > Emulate and instrument ARM, ARM64, MIPS, X86 and X86_64
- > Emulate and instrument Linux, MacOS, iOS, Windows and FreeBSD
- > High-level Python API access to register, CPU and memory
- > 1,700+ Github star, more than 9,000+ pypi download, 60+ contributors worldwide
- > Contributor from Dell, Intel, SentinelOne and etc

About lazymio && kabeor && kevin

~ \$ whoami
Ziqiao Kong



~ \$ file [Lazymio](#)
The sheperd lab, JD security, Security Engineer.
CTF player, member of Lancet.
GeekPwn 2019 Hall of Fame.
HITB, BlackHat, SDC 2020

~ \$ 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: Chenxu Wu

The sheperd lab, JD security.

Security Engineer.

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_

>About Qiling Framework

- <https://qiling.io>
- <https://github.com/qilingframework/qiling>
- <https://docs.qiling.io>
- @qiling_io



The screenshot displays the GitHub repository page for 'qilingframework/qiling'. The main area shows the commit history for the 'master' branch, listing 3,445 commits from various contributors. The commits are organized into several commits per day, with dates ranging from last month to 15 months ago. The sidebar on the right provides additional information about the repository, including its purpose as a 'Qiling Advanced Binary Emulation Framework', tags like 'binary', 'emulator', and 'framework', and links to 'Readme' and 'GPL-2.0 License'.

Date	Commit Message	Author
Sep 30	adding gitee sync actions	xwings
6 months ago	clean up docs and plan for filter	
2 months ago	refine tcp and udp sockets	
last month	getting ready for 1.1.3	
2 months ago	refine tcp and udp sockets	
2 months ago	clean up 8086 folder	
3 months ago	Fixing travis docker build error	
6 months ago	core.py: move exit_code to os	
15 months ago	import	
2 months ago	fixed some typo errors and updated donation details	
last month	update changelog	

Problems

Current Emulation Limitation

You became victim of the PETYA RANSOMWARE!

The harddisks of your computer have been encrypted with an military grade encryption algorithm. There is no way to restore your data without a special key. You can purchase this key on the darknet page shown in step 2.

To purchase your key and restore your data, please follow these three easy steps:

1. Download the Tor Browser at "<https://www.torproject.org/>". If you need help, please google for "access onion page".
 2. Visit one of the following pages with the Tor Browser:

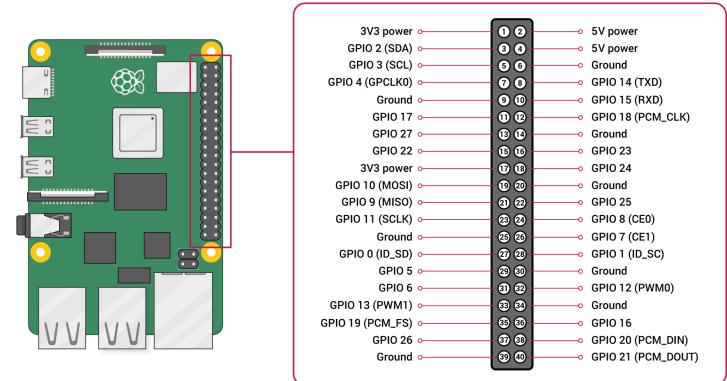
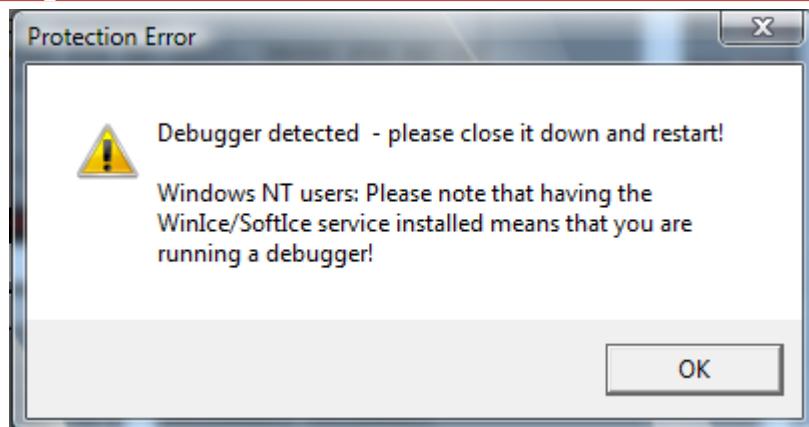
<http://petya37h5tbhyvki.onion/MvnHqz>
<http://petya5koahsf7sv.onion/MvnHqz>

3. Enter your personal decryption code there:

afMf5Z-C83M2q-Nv9uR1-g9GZXy-a4iU47-c5R4iT-xR1WZk-nX4HmW-rnc1Kg-HMekdy
W8WDrr-rXz6TZ-jo69HJ-pre5Ry-Mug9rt

If you already purchased your key, please enter it below

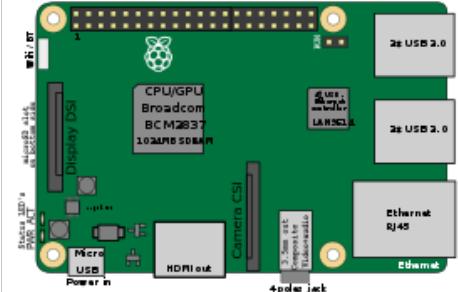
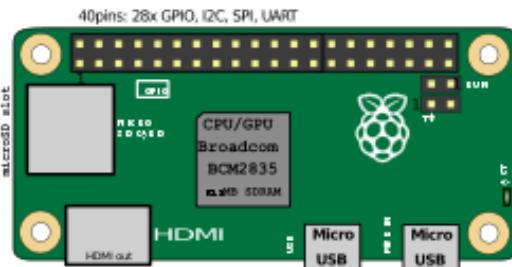
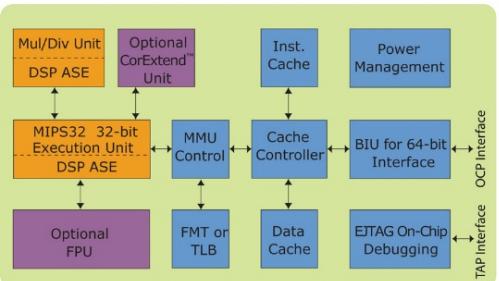
Key:



Not only for *nix and Wintel or other Operating System
More platform is either limited or non emulation or analysis tools

Current RE Issues

24K Core Architecture



Limited Architecture

- > Current emulation and analysis tools is not helping on cross arch and platform
- > IoT products always from few limited suppliers and comes with outdated and insure SDK
- > Outdated development facilities required for building new tools
 - > Outdated Compiler
 - > Outdated Debugger
 - > Almost none analysis tools

Limited access

- > Partially close source Firmware and binary
- > Existing guided fuzzers rely on source code available
 - > Source code is needed for branch instrumentation to feedback fuzzing progress
 - > Emulation such as QEMU mode support in AFL is slow & limited in capability (hardware support)
 - > Same issue for other tools based on Dynamic Binary Instrumentation

Close sourced binary

- > Most fuzzers are built for X86 only
 - > Embedded systems based on Arm, Arm64, Mips, PPC
- > Existing DBIs are poor for non-X86 CPU
 - > Pin: Intel only
 - > DynamoRio: experimental support for Arm

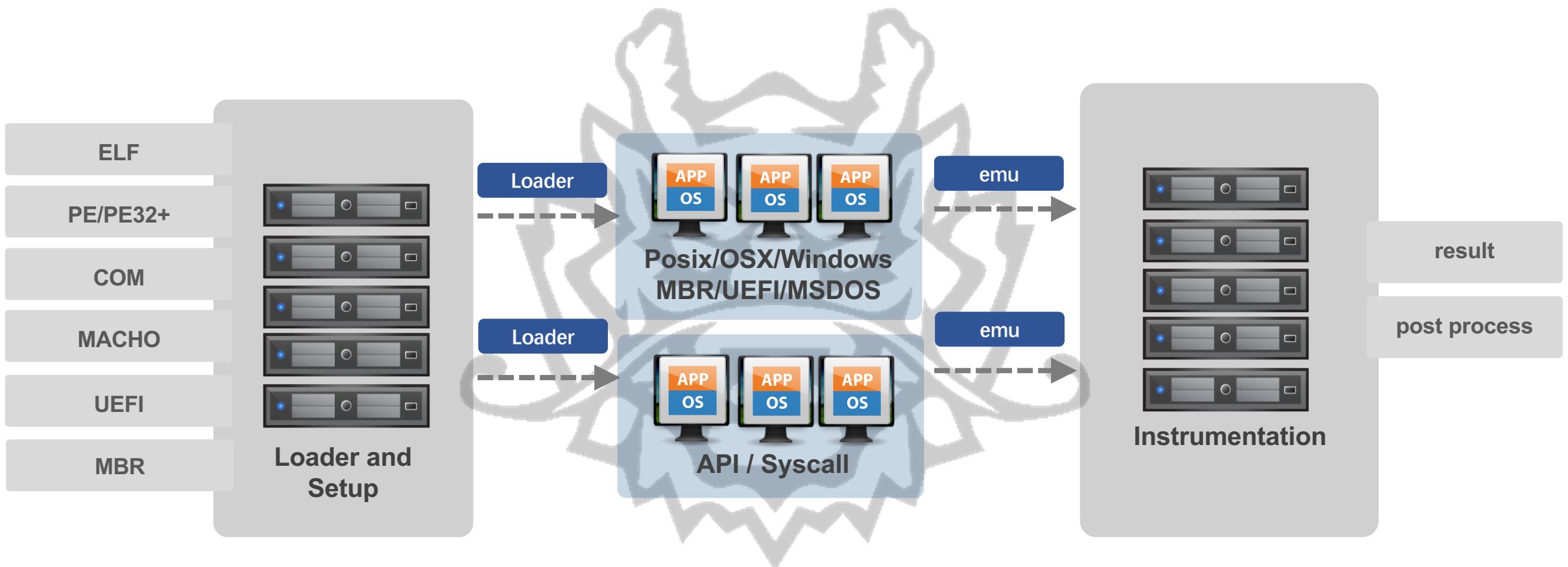
Overview

Features

- Cross platform: Windows, MacOS, Linux, BSD, UEFI, MBR
- Cross architecture: X86, X86_64, Arm, Arm64, MIPS, 8086
- Multiple file formats: PE, MachO, ELF, UEFI(PE), 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)	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	-	<input type="checkbox"/>	-
Linux (ELF)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
MacOS (MachO)	-	<input type="checkbox"/>	<input checked="" type="checkbox"/>	-	<input type="checkbox"/>	-
BSD (ELF)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
UEFI	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	-	-	-
DOS (COM)	<input checked="" type="checkbox"/>	-	-	-	-	-
MBR	<input checked="" type="checkbox"/>	-	-	-	-	-

How Does It Work



Base OS can be Windows/Linux/BSD or OSX
And not limited to ARCH

Design

Framework Architecture

Qiling Framework

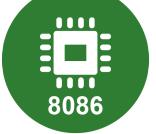
CPU
Architecture

Loader

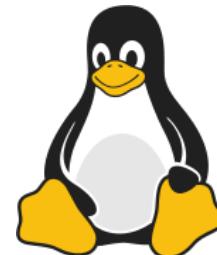
OS

Debugger

Extensions



Posix



```
qdb
gp: 0xe047fe000 at: 0x00000000
sp: 0x0000cf38 tl: 0x00000000
r0: 0x00000000 t1: 0x00000000
t4: 0x00000000 t9: 0x00000000
t8: 0x00000000 s1: 0x00000000
r1: 0x00000000 s2: 0x00000000
k0: 0x00000000 k1: 0x00000000
```



coverage

report

windows
sdk



sanitizers



Instrumentation

What are we

Framework, NOT Tools

EFI Fuzzer

A screenshot of the GitHub repository for Sentinel-One/efi_fuzz. The repository has 15 commits, 2 branches, and 0 tags. The most recent commit by liba2k is "Adding docker support. (#5)" from 24 days ago. Other commits include "NotMyUefiFault" and "Initial public commit". The repository includes files like README.md, efi_fuzz.py, requirements.txt, and sanitizer.py.

Decoder

A screenshot of the GitHub repository for nmantani/FileInsight-plugins. The repository has 214 commits, 1 branch, and 16 tags. The most recent commit by nmantani is "Use "py.exe --list" instead of hard-coded paths to check Python 3 ins..." from 2 days ago. A screenshot of the McAfee FileInsight hex editor interface is shown, displaying a file named "malicious.pdf" with various hex and ASCII data.

VAC3 Emulator

A screenshot of the GitHub repository for ioncodes/vacation3-emu. The repository has 3 commits, 1 branch, and 0 tags. The most recent commit by ioncodes is "added link" from Sep 29. The repository includes files like README.md, images, .gitignore, README.md, emu.py, and typedefs.h.

IoT Emulator **MacOS Emulator**
IOS Emulator **Binary Decrypt**

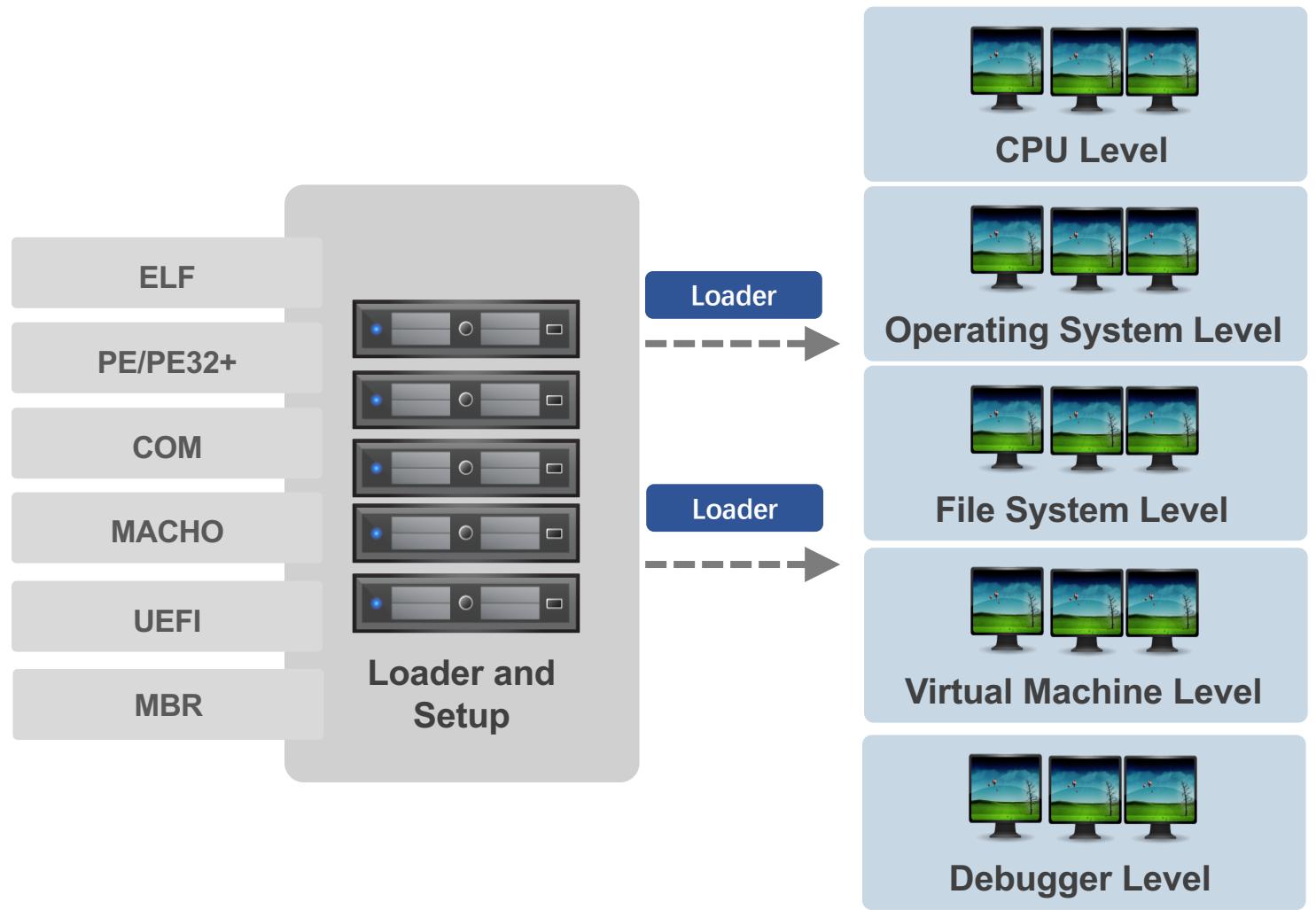
Qiling Framework



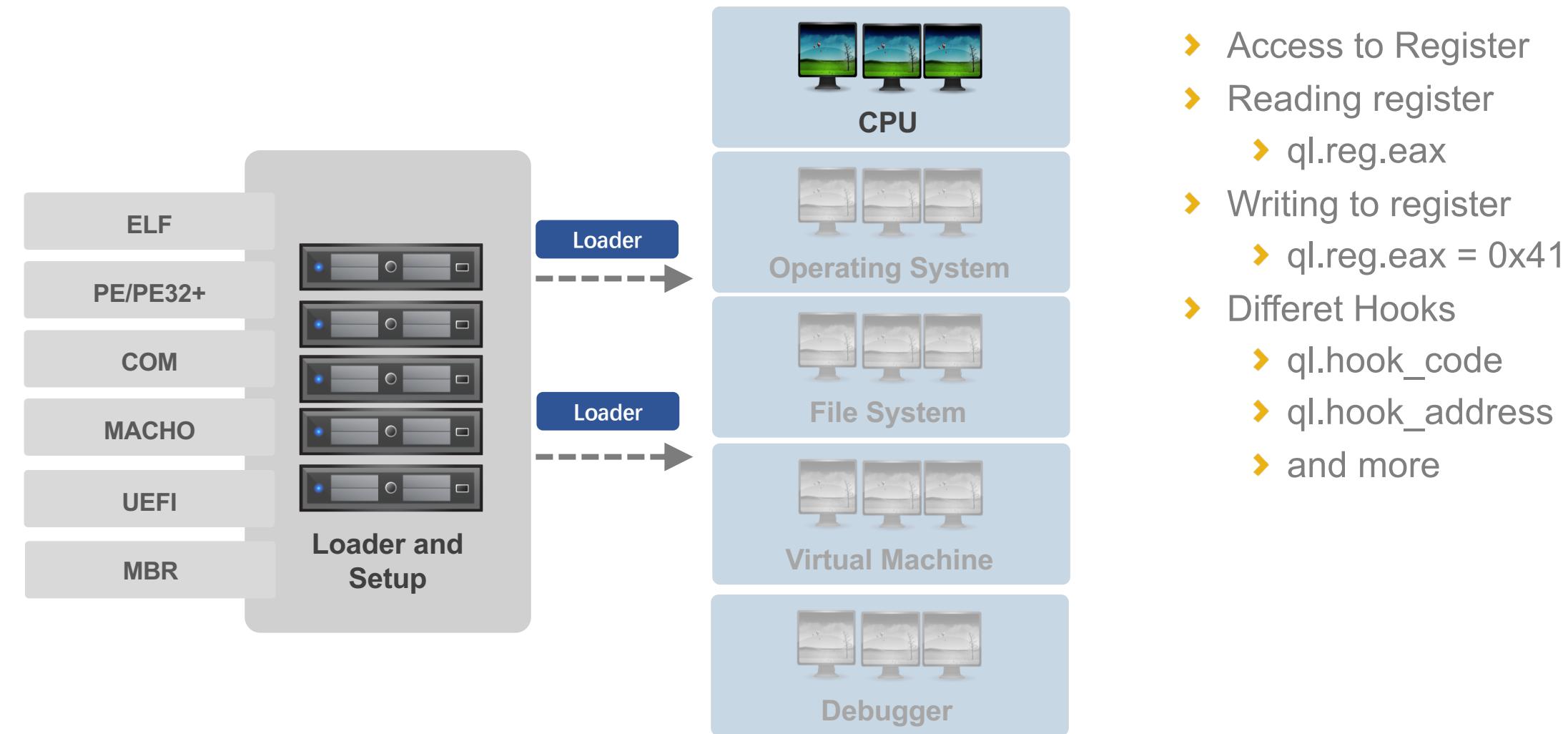
Instrumentation (Qiling's API)

Qiling and APIs

Qiling Framework and Its Interface

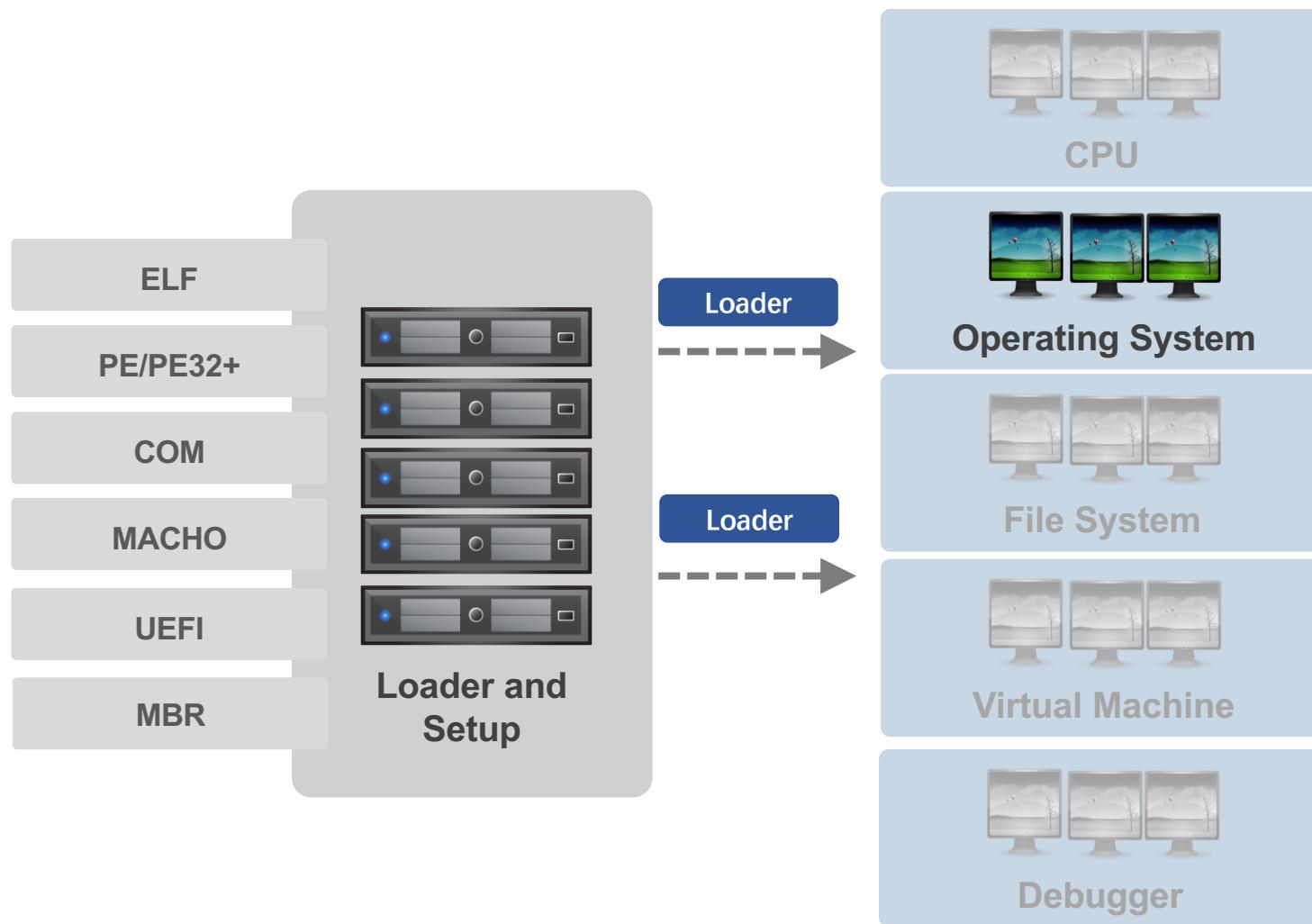


Qiling Framework: CPU



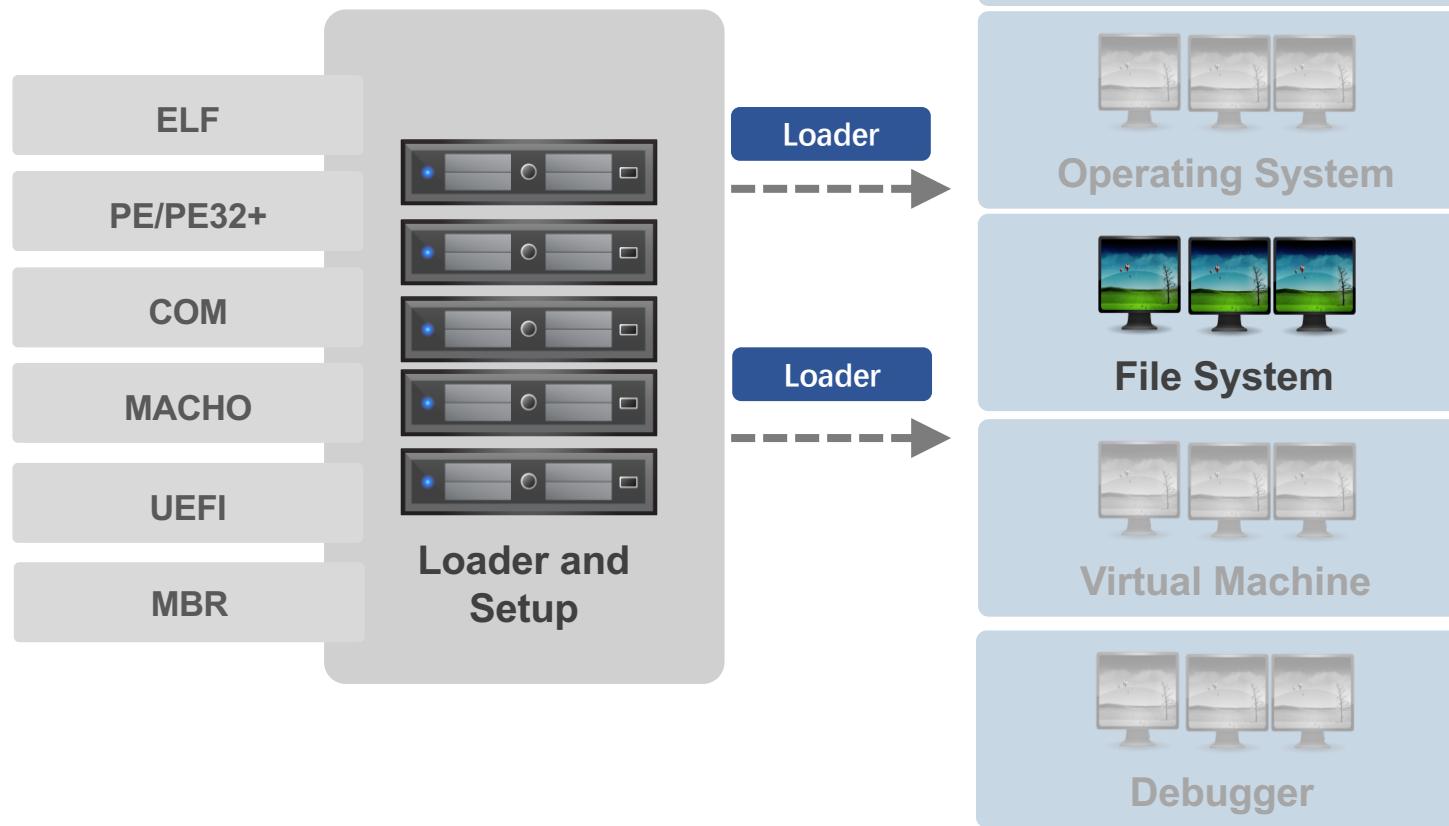
More APIs: <https://docs.qiling.io>

Qiling Framework: Operating System



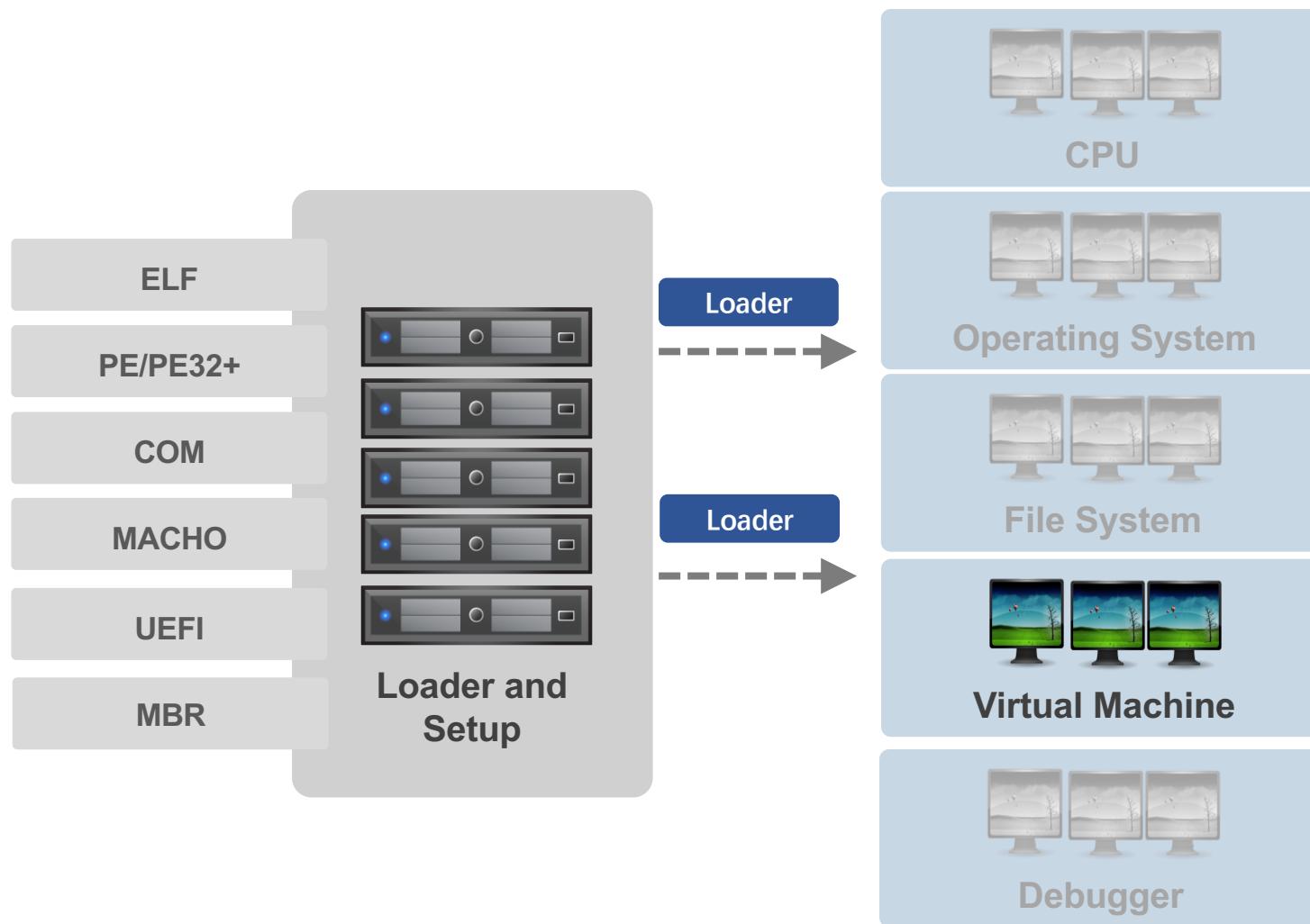
- Access to memory
 - `ql.mem.read()`
 - `ql.mem.write()`
- Search pattern from memory
 - `ql.mem.search()`
- Stack related operation
 - `ql.stack_pop`
 - `ql.stack_push`
- Syscall replacement
 - `ql.set_syscall()`
 - `ql.set_api()`
- Replace library call with
 - `ql.set_api()`

Qiling Framework: File System



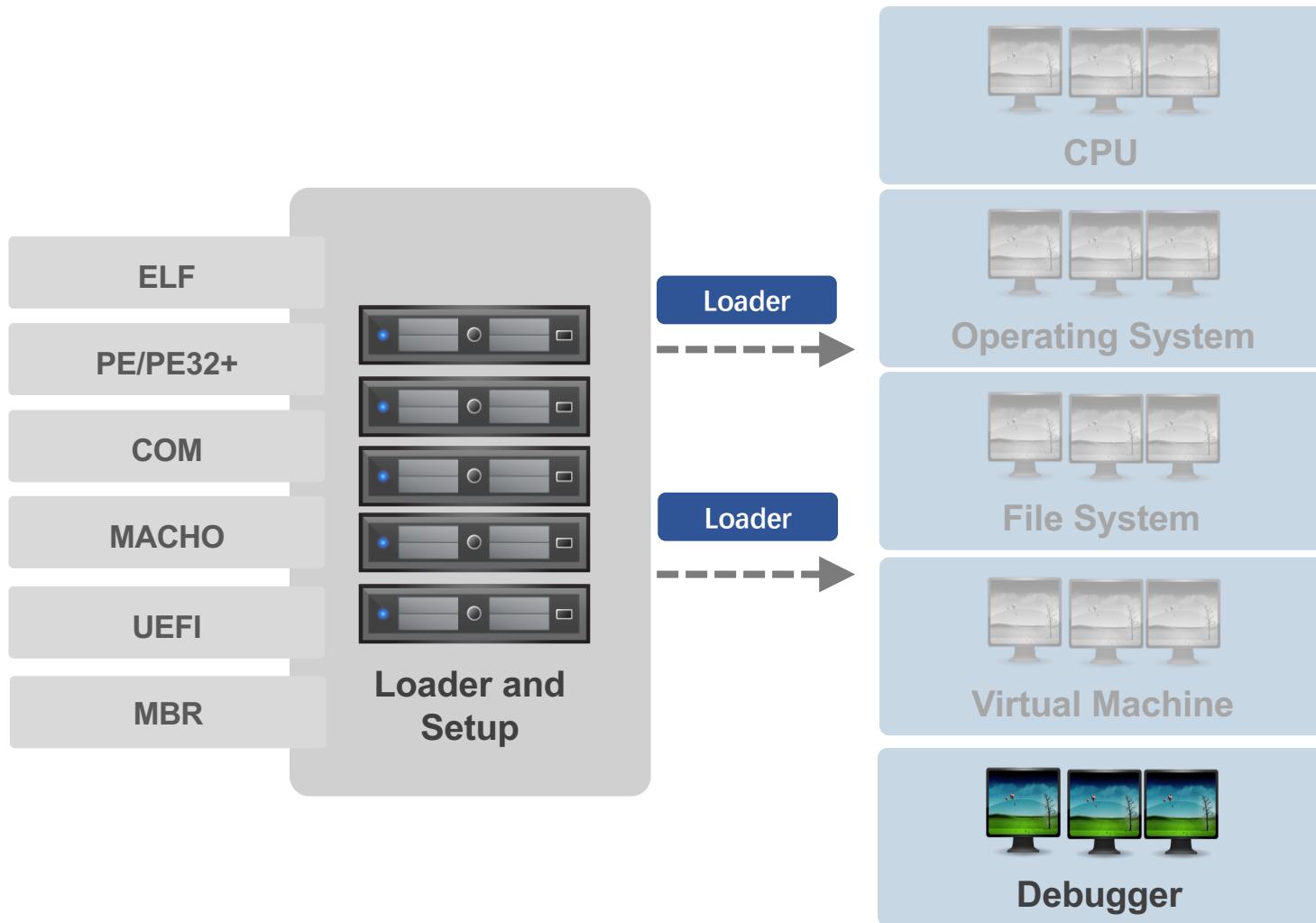
- Map host file
 - `ql.fs_mapper()`
- Hijack accessed file
 - `ql.fs_mapper(hijack_func)`
- Stdio replacement
 - `stdin`
 - `stdout`
 - `Stderr`
- Patch file's memory before execution
 - `ql.patch`

Qiling Framework: Virtual Machine



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

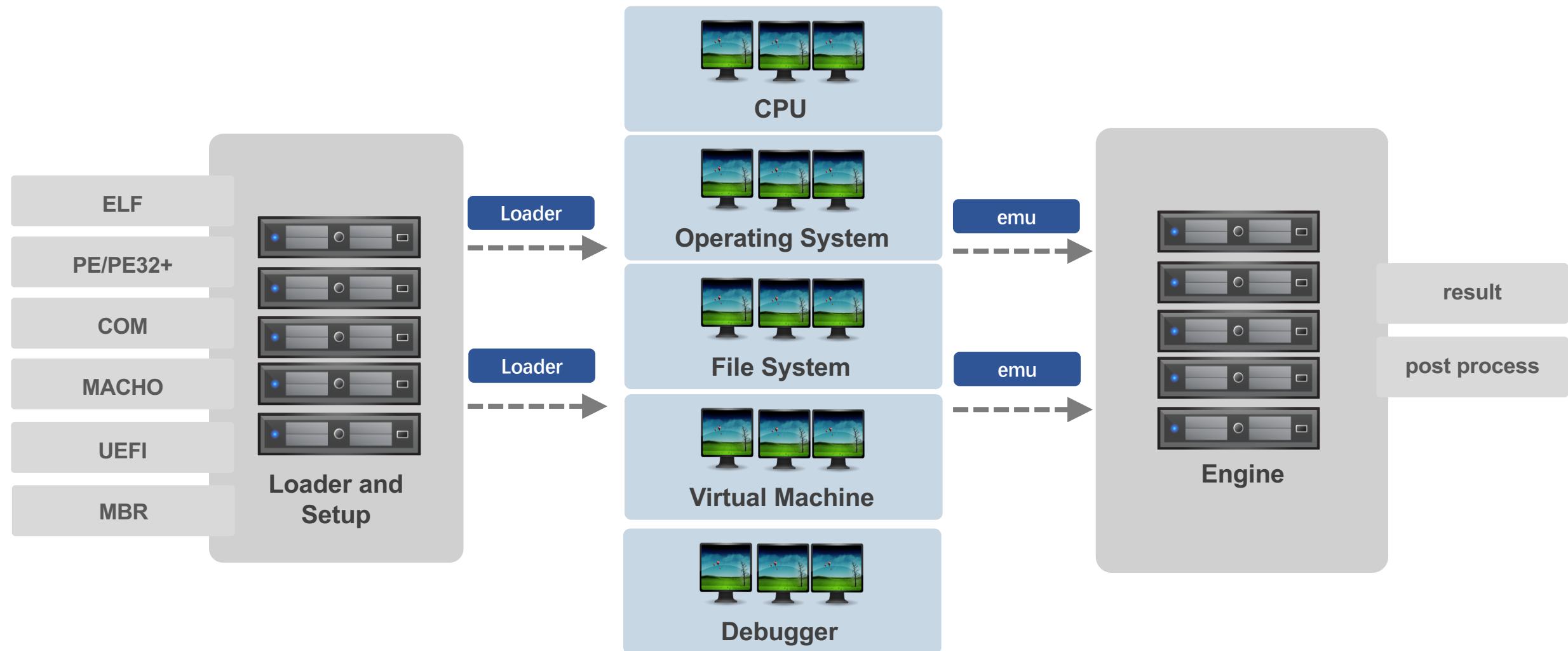
Qiling Framework: Debugger



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

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

Qiling Framework: In a Nutshell



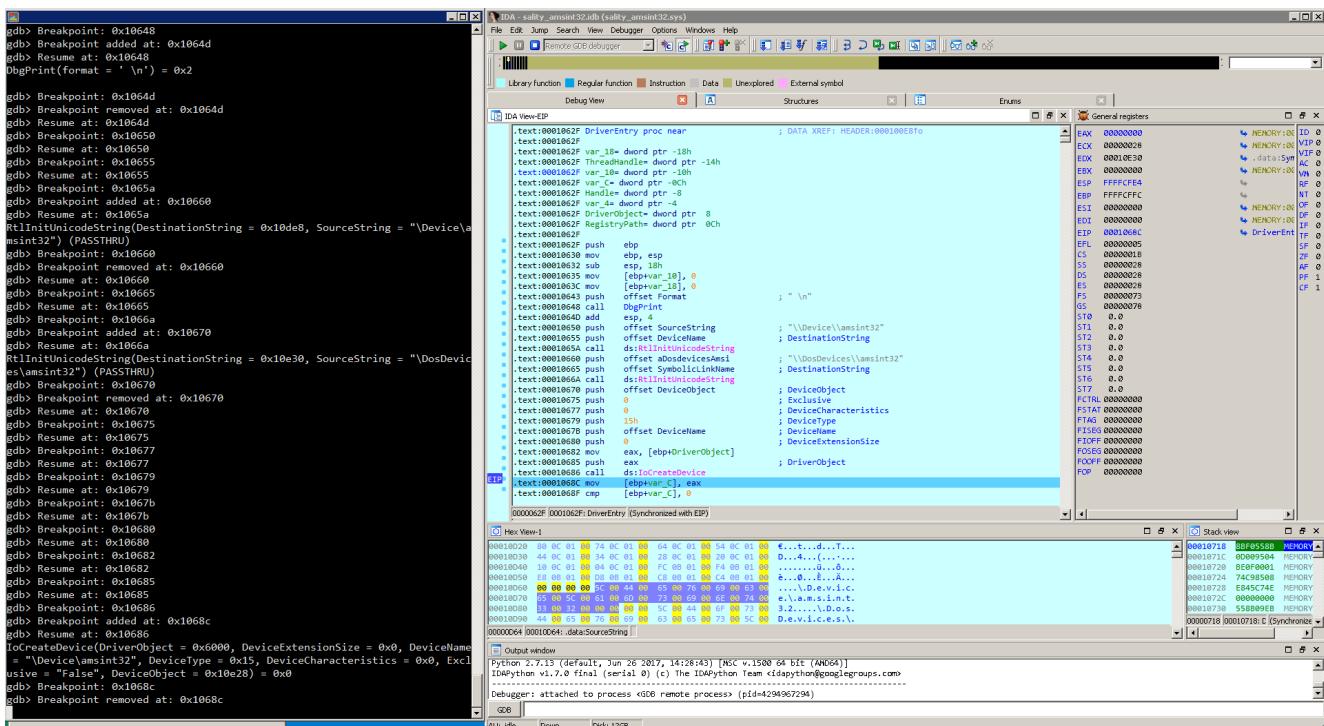
Base OS can be Windows/Linux/BSD or OSX
And not limited to ARCH

Demo

Malware & Rootkit Analysis

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

```
[root@7fd8b304fcecc:/qilingdev# python3 examples/one.py samples_anycrun/al-khaser.bin 2>&1  
[al-khaser version 0.67]  
-----  
[*] 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 invalid handle [=] GOOD  
[*] Checking UnhandledExcepFilterTest [=] GOOD  
[*] Checking OutputDebugString [=] GOOD  
[*] Checking Hardware Breakpoints [=] GOOD  
[*] Checking Software Breakpoints [=] GOOD  
[*] Checking Interrupt 0x2d [=] BAD  
We are finally done  
root@7fd8b304fcecc:/qilingdev# ]
```



```
(22:43:04):xwings@bespin:<~/projects/qiling>
(15)$ python3 qltool run -f examples/rootfs/x86_windows/bin/al-khaser.bin --rootfs xamples/rootfs/x86 windows
[+] Loading examples/rootfs/x86_windows/bin/al-khaser.bin to 0x400000
[+] PE entry point at 0x403d6a
[+] Initiate stack address at 0xfffffd000
[+] TEB addr is 0x6000
[+] PEB addr is 0x6044
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/kernel32.dll to 0x10000000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/kernel32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/user32.dll to 0x100d4000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/user32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/advapi32.dll to 0x1019d000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/advapi32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/ole32.dll to 0x1023e000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/ole32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/oleaut32.dll to 0x1039a000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/oleaut32.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/shlwapi.dll to 0x1042c000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/shlwapi.dll
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/setupapi.dll to 0x10470000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/setupapi.dll
[+] Done with loading examples/rootfs/x86_windows/bin/al-khaser.bin
GetSystemTimeAsFileTime(lpSystemTimeAsfileTime = 0xfffffcfec)
GetCurrentThreadId() = 0x0
GetCurrentProcessId() = 0x2005
QueryPerformanceCounter(lpPerformanceCount = 0xfffffcfe4) = 0x0
IsProcessorFeaturePresent(ProcessorFeature = 0xa) = 0x1
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll to 0x108b9000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll
LoadLibraryExW(lpLibFileName = "api-ms-win-core-synch-1-2-0", hFile = 0x0, dwFlags = 0x800) = 0x108b9000
GetProcAddress(hModule = 0x108b9000, lpProcName = "InitializeCriticalSectionEx") = 0x0
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x424d64, dwSpinCount = 0x0)
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-fibers-1-1-1.dll to 0x108bc000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-fibers-1-1-1.dll
LoadLibraryExW(lpLibFileName = "api-ms-win-core-fibers-1-1-1", 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-1-2-0", hFile = 0x0, dwFlags = 0x800) = 0x108b9000
GetProcAddress(hModule = 0x108b9000, lpProcName = "InitializeCriticalSectionEx") = 0x0
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x425380, dwSpinCount = 0x0)
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll to 0x108bc000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll
LoadLibraryExW(lpLibFileName = "api-ms-win-core-synch-1-2-0", hFile = 0x0, dwFlags = 0x800) = 0x108bc000
GetProcAddress(hModule = 0x108bc000, lpProcName = "InitializeCriticalSectionEx") = 0x0
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x425398, dwSpinCount = 0x0)
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll to 0x108bc000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll
LoadLibraryExW(lpLibFileName = "api-ms-win-core-synch-1-2-0", hFile = 0x0, dwFlags = 0x800) = 0x108bc000
GetProcAddress(hModule = 0x108bc000, lpProcName = "InitializeCriticalSectionEx") = 0x0
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x4253b0, dwSpinCount = 0x0)
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll to 0x108bc000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll
LoadLibraryExW(lpLibFileName = "api-ms-win-core-synch-1-2-0", hFile = 0x0, dwFlags = 0x800) = 0x108bc000
GetProcAddress(hModule = 0x108bc000, lpProcName = "InitializeCriticalSectionEx") = 0x0
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x4253c8, dwSpinCount = 0x0)
[+] Loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll to 0x108bc000
[+] Done with loading jexamples/rootfs/x86_windows/Windows/SySOw64/api-ms-win-core-synch-1-2-0.dll
```

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

```
american fuzzy lop ++2.65d (python3) [explore] {0}
process timing
    run time : 0 days, 0 hrs, 12 min, 52 sec
    last new path : 0 days, 0 hrs, 0 min, 7 sec
last uniq crash : 0 days, 0 hrs, 0 min, 15 sec
last uniq hang : none seen yet
overall results
    cycles done : 2
    total paths : 36
    uniq crashes : 1
    uniq hangs : 0
cycle progress
    now processing : 21*0 (58.3%)
    paths timed out : 0 (0.00%)
map coverage
    map density : 1.55% / 1.60%
    count coverage : 1.36 bits/tuple
stage progress
    now trying : havoc
    stage execs : 2742/32.8k (8.37%)
    total execs : 122k
    exec speed : 161.7/sec
findings in depth
    favored paths : 4 (11.11%)
    new edges on : 8 (22.22%)
    total crashes : 9 (1 unique)
    total tmouts : 0 (0 unique)
fuzzing strategy yields
    bit flips : 0/3480, 0/3468, 0/3444
    byte flips : 0/435, 0/401, 0/385
    arithmetics : 2/23.0k, 0/4022, 0/1454
    known ints : 1/2313, 0/10.5k, 0/16.6k
    dictionary : 0/0, 0/0, 0/0
    havoc/rad : 17/48.6k, 1/1312, 0/0
    py/custom : 0/0, 0/0
    trim : 0.00%/154, 65.57%
path geometry
    levels : 4
    pending : 25
    pend fav : 0
    own finds : 35
    imported : n/a
stabi
```

The screenshot shows a product listing page for Tenda routers. At the top, there's a navigation bar with links like '首页', '家用产品', '路由器', '无线网卡', etc. Below the navigation is a search bar and some filtering options. The main content area shows several routers listed, each with a thumbnail, name, and a brief description. The Tenda AC15 router is specifically highlighted with a red rectangular box around its thumbnail and name.

Not secure | tenda.com.cn/product/category-151.html

Tenda 智能家用产品 企业商用产品 服务支持 解决方案 如何购买 走进腾达

首页 > 家用产品 > 路由器 > 全部产品

类别 穿墙宝 路由器 无线网卡 交换机 电力线 信号放大器 接入终端 网络摄像机
筛选 Beamforming MU MIMO WiFi 技术 WiFi 速率 光纤网络 户型 覆盖范围 频段 USB 天线
端口类型
条件 暂无筛选条件
排序 推荐 最新 热门 默认
总计 20 款 路由器

New

产品	描述
AC23	203Mbps/5G频段4发4收7*6dBi穿墙天线/三芯片架构/支持IPv6 AC2100千兆端口双频型双频无线路由器
AC18	5口全千兆，光纤网络绝配，500m别墅级覆盖，支持USB3.0存储 1900M 11ac千兆口别墅型双频无线路由器
AC15	一款视墙若无物，速度快得超乎你想象的1900M路由器 1900M 11ac双频无线千兆口路由器

Qiling Debugger

Debugger

GDB Server

qdb

IDA Plugin

The screenshot shows the GDB Server interface with three main sections:

- Registers:** Displays CPU registers (gp, a0-a9, t0-t9, s0-s9, f0-f7) with their addresses and values.
- Stack:** Displays the stack contents, showing memory dump and assembly dump.
- Code:** Displays assembly code at the current PC address (0x400590).

Annotations in the Registers section highlight specific register values, such as gp = 0x047f6000 and sp = 0x047ba04.

The screenshot shows the IDA Plugin interface with the following components:

- Functions window:** Shows the function structure with various symbols like _main, _lfunc_start_main, _write, _lfunc_end_main, and _gmon_start.
- Assembly view:** Displays assembly code for the main function, including instructions like mov, addiu, and jal.
- Call graph:** Shows the call graph for the main function.
- Register View:** Shows the register values for the current instruction.
- Stack View:** Shows the stack contents.
- Output window:** Displays various assembly dump and memory dump outputs.

Qiling GDBServer

Terminal

```
set.SOCK_STREAM)
gdb
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote 127.0.0.1:9999
Remote debugging using 127.0.0.1:9999
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x00409a16 in ?? ()
(gdb) break *0x00408192
Breakpoint 1 at 0x408192
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x00408192
(gdb) c
Continuing.

Breakpoint 1, 0x00408192 in ?? ()
(gdb) disas 0x0408192,0x0408198
Dump of assembler code from 0x408192 to 0x408198:
=> 0x00408192: push %ecx
 0x00408193: push %esi
 0x00408194: call *0x40a138
End of assembler dump.
(gdb) x/s $ecx
0xffffcf14: "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwerwgea.com"
(gdb)
```

GUI

The screenshot shows the IDA Pro interface with several windows open:

- IDA View-EIP:** Assembly view showing the decompiled code for the WinMain function. The assembly listing includes instructions like sub esp,50h, push esi, mov ecx,0Eh, and calls to InternetOpenA and InternetCloseHandle.
- Registers:** Registers pane showing CPU register values. For example, ECX is 00431300, EIP is 0040814F, and CS is 0000000B.
- Memory:** Hex dump of memory starting at address 00408168. It shows the URL "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwerwgea.com" being loaded into memory.
- Stack view:** Stack dump showing the current stack contents.
- Output window:** Shows messages like "Autoanalysis subsystem has been initialized".

Qiling 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

The screenshot shows the Qiling IDA Plugin integrated into the IDA Pro interface. The main window displays assembly code for the 'main' function, which includes calls to '_write' and '_isoc99_scanf'. Below the assembly view are two call graphs: one for 'sub_80484F7' and another for 'locret_804857C'. The bottom right corner shows the 'Output window' displaying various CPU register values.

Assembly code (IDA View-A):

```
; Attributes: bp-based frame fuzzy-sp
main proc near
; _unwind {
push    ebp
mov     ebp, esp
and    esp, 0FFFFFFF0h
sub    esp, 10h
mov    dword ptr [esp+8], 17h ; n
mov    dword ptr [esp+4], offset aReversingKrEas ; "Reversing.Kr Easy ELF\n"
mov    dword ptr [esp], 1 ; fd
call   _write
call   sub_8048434
call   sub_8048451
cmp    eax, 1
jnz    short loc_804855B

call   sub_80484F7
mov    eax, 0
jmp    short locret_804857C

loc_804855B:
; n
mov    dword ptr [esp+8], 6
mov    dword ptr [esp+4], offset aWrong ; "Wrong\n"
mov    dword ptr [esp], 1 ; fd
call   _write
mov    eax, 0

locret_804857C:
leave
```

QL Register View:

eax: 0x7769ADD8	ecx: 0x59FE442C	edx: 0x7FF3CEA4
ebx: 0x00000000	esp: 0x7FF3CE60	ebp: 0x7FF3CE78
esi: 0x77699000	edi: 0x00000000	eip: 0x08048524
ef: 0x00000006	cs: 0x0000001B	ss: 0x00000028
ds: 0x00000028	es: 0x00000000	fs: 0x00000000
gs: 0x00000063	st0: 0x00000000	st1: 0x00000000
st2: 0x00000000	st3: 0x00000000	st4: 0x00000000
st5: 0x00000000	st6: 0x00000000	st7: 0x00000000

QL Stack View:

Stack at 0x7FF3CE60
7FF3CDE8: 774BF2F0
7FF3CDC8: 00000001
7FF3CDF0: 00000000
7FF3CDF4: 00000001
7FF3CDF8: 047E1940
7FF3CDFA: 047D2121
7FF3CEO0: 08048034
7FF3CEO4: 00000009
7FF3CEO8: 00000000
7FF3CEOCC: 047E1000
7FF3CE10: 00000000
7FF3CE14: 00000000
7FF3CE18: 00000000
7FF3CE1C: 08048380

Output window:

```
esi : 0000000076990000 edi : 0000000000000000 eip : 0000000008048521
[INFO][custom_script13] cr0 : 0000000000000000 cr2 : 0000000000000000
[INFO][custom_script13] cr1 : 0000000000000000 cr3 : 0000000000000000
[INFO][custom_script13] cr4 : 0000000000000000 cr5 : 0000000000000000
[INFO][custom_script13] cr6 : 0000000000000000 cr7 : 0000000000000000 cr8 : 0000000000000000
[INFO][custom_script13] cr9 : 0000000000000000 cr10: 0000000000000000 cr11: 0000000000000000
[INFO][custom_script13] cr12: 0000000000000000 cr13: 0000000000000000 cr14: 0000000000000000
[INFO][custom_script13] st0 : 0000000000000000 st1 : 0000000000000000 st2 : 0000000000000000
[INFO][custom_script13] st3 : 0000000000000000 st4 : 0000000000000000 st5 : 0000000000000000 st6 : 0000000000000000
[INFO][custom_script13] ef : 0000000000000002 ss : 000000000000001b ds : 0000000000000028
[INFO][custom_script13] cs : 0000000000000001 es : 0000000000000028 fs : 0000000000000003
```

Python:

```
0
AU: idle Down Disk: 453GB
```

Qiling IDA Plugin custom script

Use all APIs provided by Qiling and IDAPython

Execute Till

Execute Selection

Continue

Set PC

Step

Edit Register

^K F9

```
from qiling import *
import logging

class QILING_IDA():
    def __init__(self):
        pass

    def custom_prepare(self, ql:Qiling):
        logging.info('Context before starting emulation:')
        self._show_context(ql)

    def custom_continue(self, ql:Qiling):
        logging.info('custom_continue hook.')
        self._show_context(ql)
        hook = []
        return hook

    def custom_step(self, ql:Qiling):
        def step_hook(ql, addr, size):
            logging.info(f"Executing: {hex(addr)}")
            self._show_context(ql)

        logging.info('custom_step hook')
        hook = []
        hook.append(ql.hook_code(step_hook))
        return hook

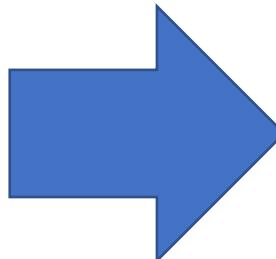
    def custom_execute_selection(self, ql:Qiling):
        logging.info('custom execute selection hook')
        hook = []
        return hook
```

OLLVM De-flatten

De-flatten: Introduction

- Reserve all original basic blocks and transform control flow.
 - Utilize a big while-switch loop to hide real control flow.
 - The example here introduces a variable b.
- With more complex code, the number of cases increases dramatically.

```
int main(int argc, char** argv) {  
    int a = atoi(argv[1]);  
    if(a == 0)  
        return 1;  
    else  
        return 10;  
    return 0;  
}
```



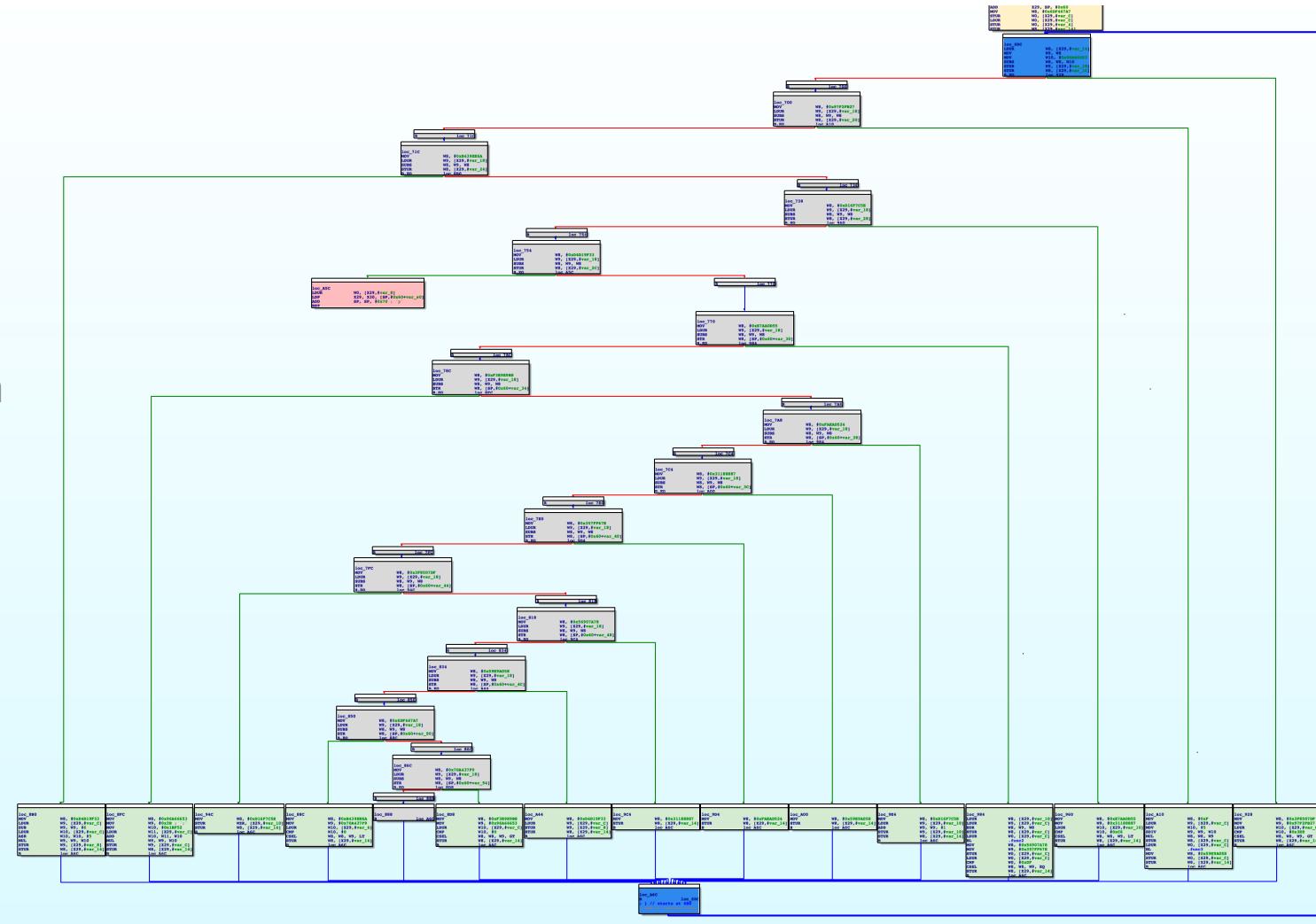
```
int main(int argc, char** argv) {  
    int a = atoi(argv[1]);  
    int b = 0;  
    while(1) {  
        switch(b) {  
            case 0:  
                if(a == 0)  
                    b = 1;  
                else  
                    b = 2;  
                break;  
            case 1:  
                return 1;  
            case 2:  
                return 10;  
            default:  
                break;  
        }  
    }  
    return 0;  
}
```

De-flatten: Sample

```
int func3(int v){  
    return v+v/2+v*v;  
}  
  
int func2(int v){  
    return v*v*4;  
}  
  
int func(int v){  
    if(v<0)  
        return (v - 6)*(v>>5);  
    if(v>0)  
        v = 59 * (v + 114514);  
    if(v > 1000){  
        for(int i =0;i<198;i++){  
            v ^= i;  
            v = func2(v);  
            if(v == 223)  
                break;  
        }  
    }else{  
        v = 15 * (v / 5);  
        v = func3(v);  
    }  
    return v;  
}  
  
int main(){  
    func(1);  
}
```

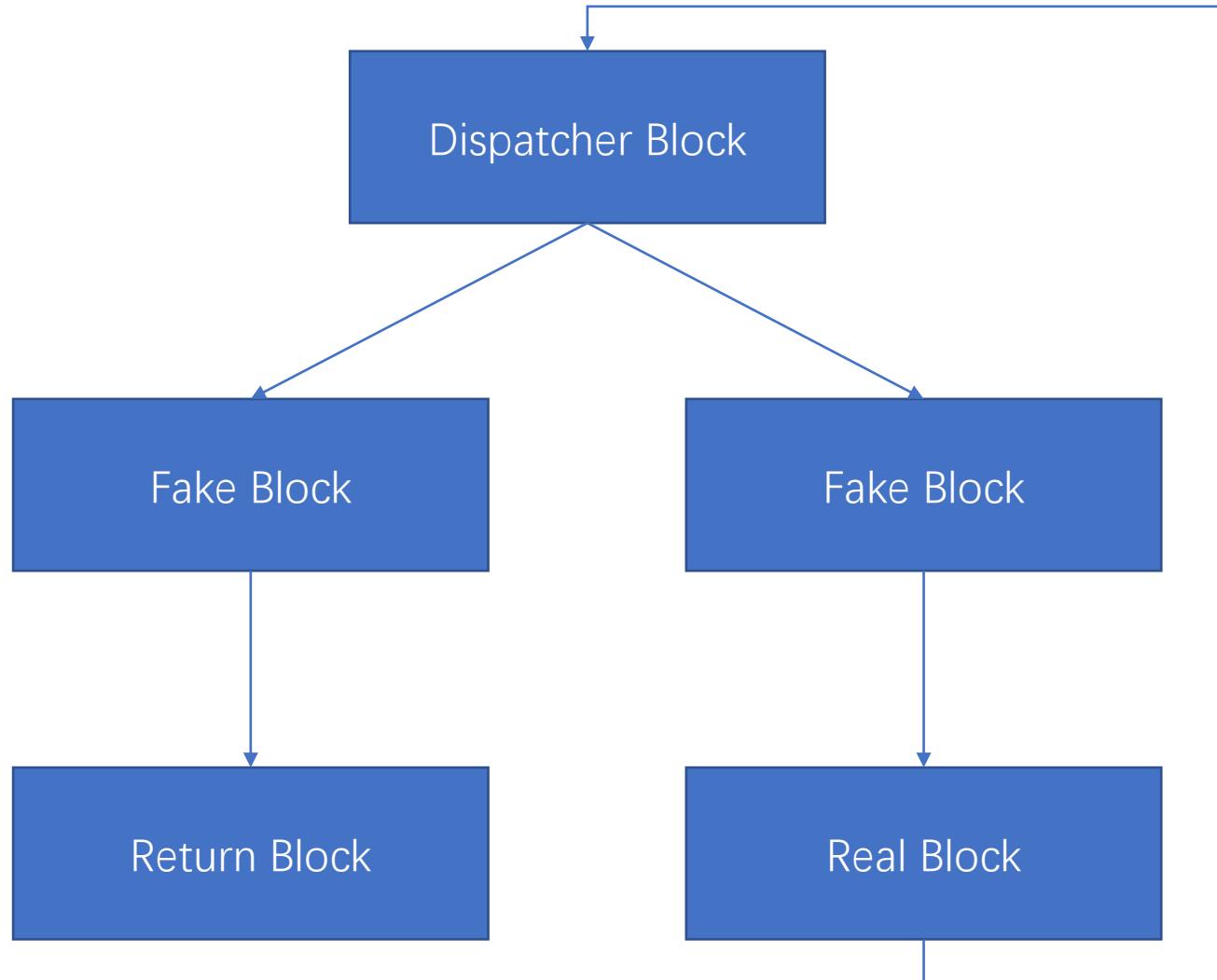


NDK + ollvm



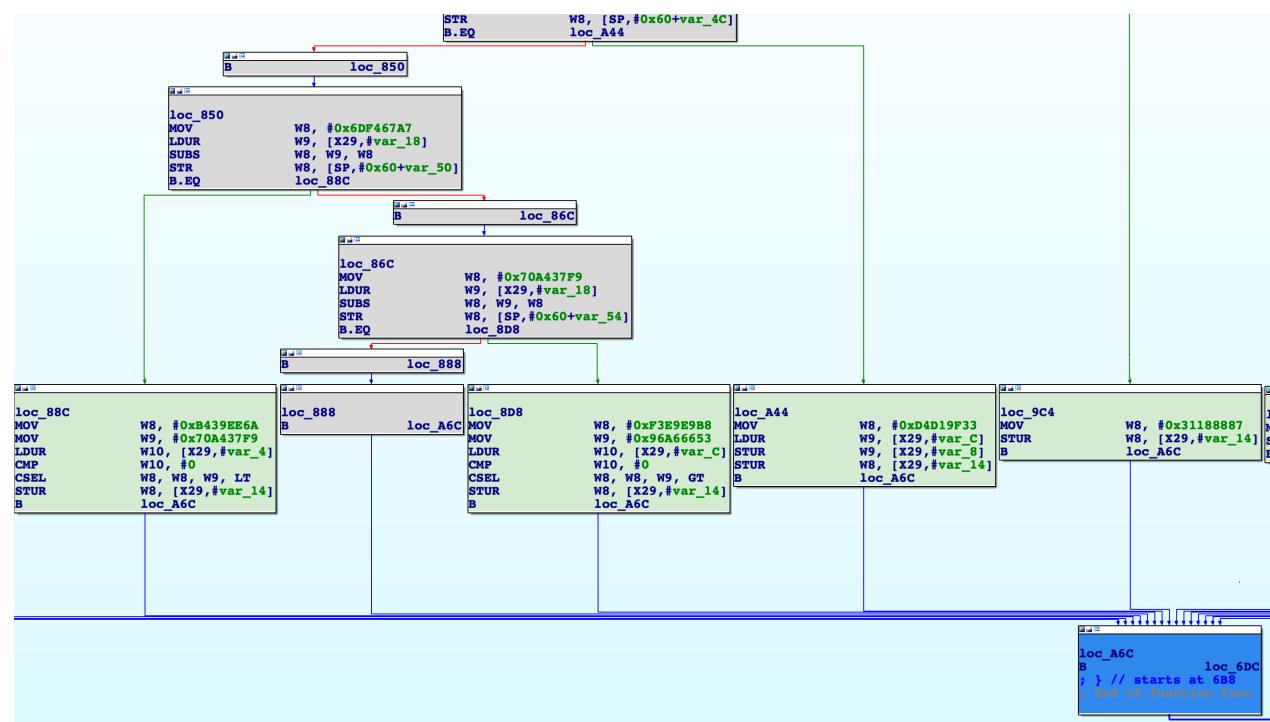
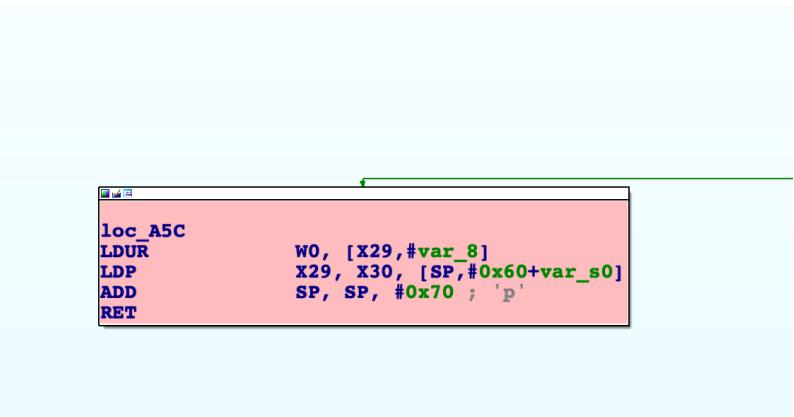
De-flatten: Identification

- Four types of blocks
- Real Blocks
 - Only contain original logic
 - May contain zero or only one conditional statement.
- Fake Blocks
 - Only responsible for switch-case implementation.
- Return Blocks
 - Return the function.
- Dispatcher Blocks
 - Equivalent to switch statement.
 - Decide the following control flow.
 - The core of the identification.



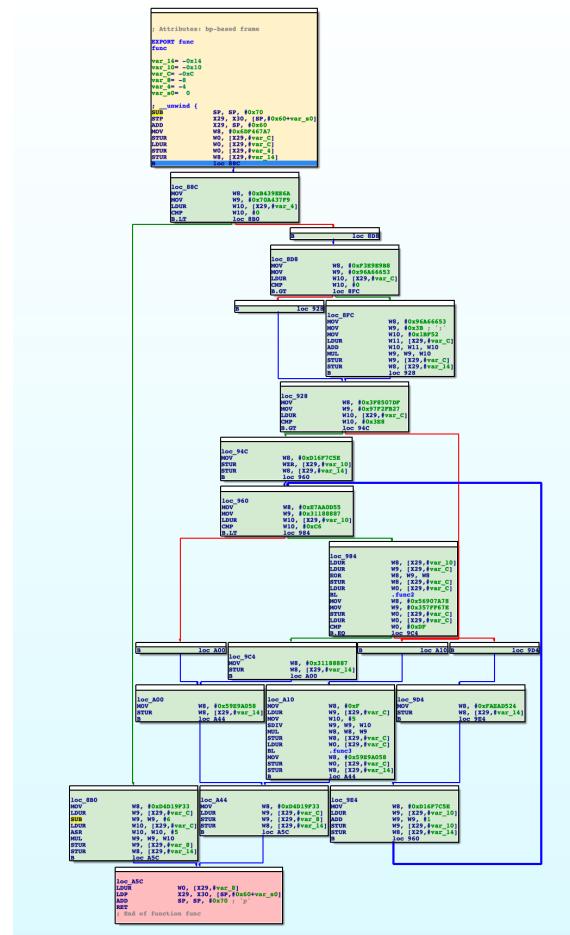
De-flatten: Identification

- The successor of all real blocks is the dispatcher block.
 - Locate dispatcher blocks by reference counts.
- The blocks with more than one instructions whose successor is the dispatcher can be identified as real blocks.
- The blocks which has any return statement like retn is considered as return Blocks.
- All other blocks are fake blocks.
- We use different colors to identify the blocks.
 - Users can also adjust the result manually.



De-flatten: Qiling Emulation

- Restore original control flow.
 - Partial Execution && Snapshot API
 - Save snapshot and start from any real block.
 - Execute each branch for conditional instruction.
 - Stop when meeting another real/return block.
 - Record control flow and restore snapshot.
 - Iterate all real blocks.
 - Patch with Keystone
 - Only real blocks and return blocks are reserved.



```
int64 __fastcall func(int a1)
{
    int v1; // [xsp+50h] [xbp-10h]
    int v3; // [xsp+54h] [xbp-Ch]
    unsigned int v4; // [xsp+58h] [xbp-8h]

    v3 = a1;
    if ( a1 >= 0 )
    {
        if ( a1 > 0 )
            v3 = 59 * (a1 + 114514);
        if ( v3 > 1000 )
        {
            for ( i = 0; i < 198; ++i )
            {
                v3 = func2(v3 ^ (unsigned int)i);
                if ( v3 == 223 )
                    break;
            }
        }
        else
        {
            v3 = func3((unsigned int)(15 * (v3 / 5)));
        }
        v4 = v3;
    }
    else
    {
        v4 = (a1 - 6) * (a1 >> 5);
    }
    return v4;
}
```

De-flatten: Intermediate Representation

Some implementation details

- Conditional instruction differs.
- cmove, csel, ittq...
- External function calls.

Solution: IR

- IDA 7.1 microcode
- Match microcode pattern and identify conditional jumps to avoid hard-code instructions.
- Identify external calls with microcode and skip such calls.

```
=====
.text:000000000000088C loc_88C
.text:000000000000088C
.text:0000000000000894
.text:000000000000089C
.text:00000000000008A0
.text:00000000000008A4 ; CODE XREF: func+1AC↑j
.text:00000000000008A8
.text:00000000000008AC

MOV          W8, #0xB439EE6A
MOV          W9, #0x70A437F9
LDUR        W10, [X29,#var_4]
CMP          W10, #0
CSEL        W8, W8, W9, LT |
STUR        W8, [X29,#var_14]
B           loc_A6C
```



```
=====0x86c===== id=16 0x8a4
0x8a4: jge    %var_4.4 {1} , #0.4
>>>>>>>>>>>>0x884<<<<<<<<<<<
=====0x8a4===== id=17 0x8a4
0x8a4: mov    #0xB439EE6A.4 , w8.4
0x8a4: goto   @19
>>>>>>>>>>>>0x8a8<<<<<<<<<<
=====0x8a4===== id=18 0x8a8
0x8a4: mov    #0x70A437F9.4 , w8.4
```

Future

Future

- Sync with Qemu5
 - More architectures, instructions
- Android Java bytecode layer instrumentation
- IOS emulation support.
- More robust Windows emulation.
 - Introduce wine&&cygwin
- Smart Contract emulation.
- Single-chip microcomputer emulation.

Questions

The GitHub repository page for `qilingframework/qiling` is displayed. The page includes a header with repository statistics: 76 watchers, 1.7k stars, and 266 forks. A prominent red arrow-shaped button on the right side encourages users to star the repository with the text "Star us". The main content area shows the `master` branch with 2 branches and 10 tags. A list of recent commits is shown, starting with a merge pull request from `xwings`. The commits are as follows:

Commit	Message	Date
7f27ec3	merge pull request #532 from qilingframework/dev ...	Sep 30
3,445 commits		
.github	adding gitee sync actions	2 months ago
docs	clean up docs and plan for filter	6 months ago
examples	refine tcp and udp sockets	2 months ago
qiling	getting ready for 1.1.3	last month
tests	refine tcp and udp sockets	2 months ago
.gitignore	clean up 8086 folder	2 months ago
.travis.yml	Fixing travis docker build error	3 months ago
AUTHORS.TXT	core.py: move exit_code to os	6 months ago
COPYING	import	15 months ago
CREDITS.TXT	fixed some typo errors and updated donation details	2 months ago
ChangeLog	update changelog	last month

On the right sidebar, there is an "About" section with a description of the Qiling Advanced Binary Emulation Framework, links to `qiling.io`, and sections for "Readme", "GPL-2.0 License", and "Releases". The latest release is Version 1.1.3.

<https://docs.qiling.io>