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

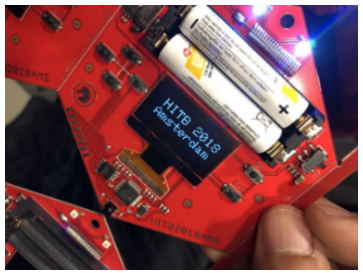
- https://qiling.io
- Lead Developer
- Founder

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 X8664
- Emulate and instrument Linux, MacOS, iphoneOS, 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, SentinalOne 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

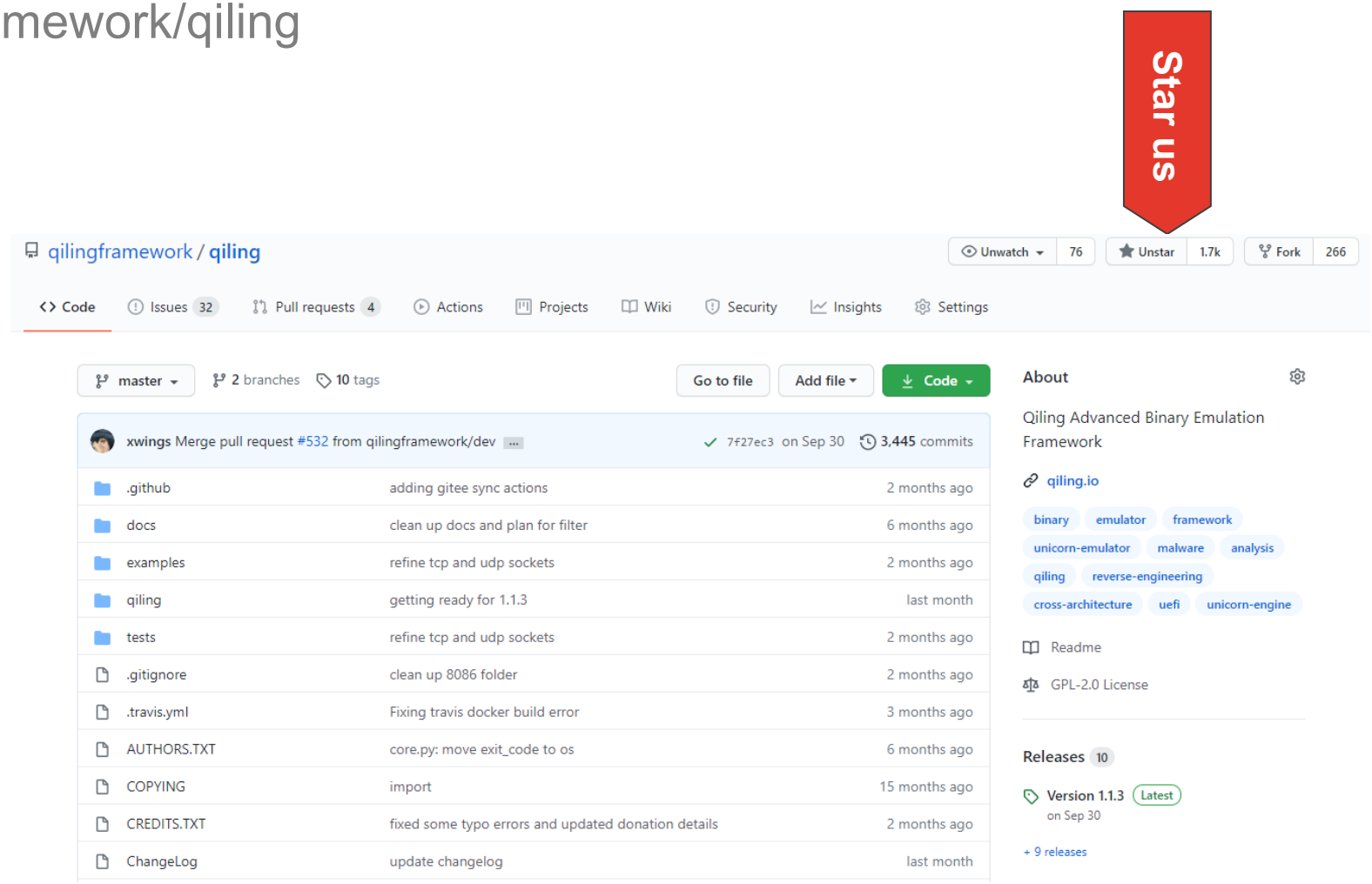
Security Engineer at The Shepherd Lab, JD Security.
Contributor of Qiling Framework
Core member of Malaysia Chapter, The Honeynet Project.
Core developer of Hex LiveCD project
SANS Advisory Board member

*Nix systems junkie
Blue team
DFIR

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

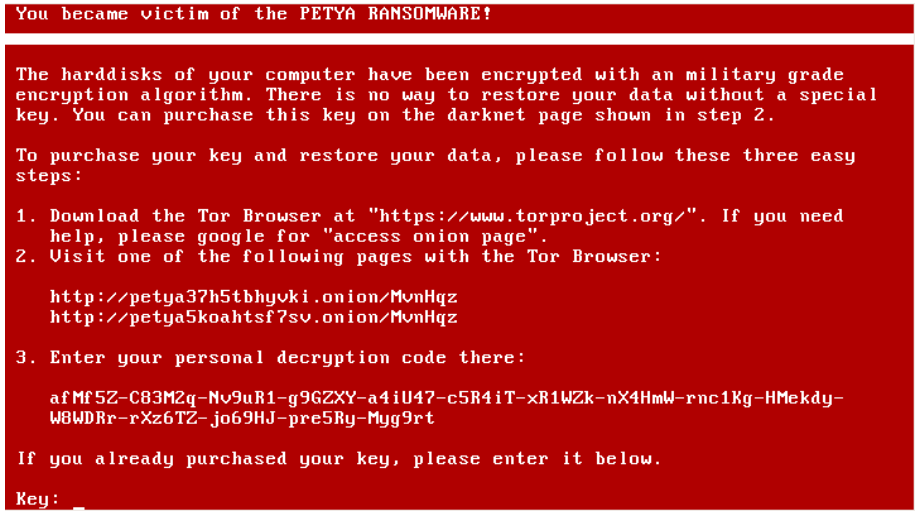
Info

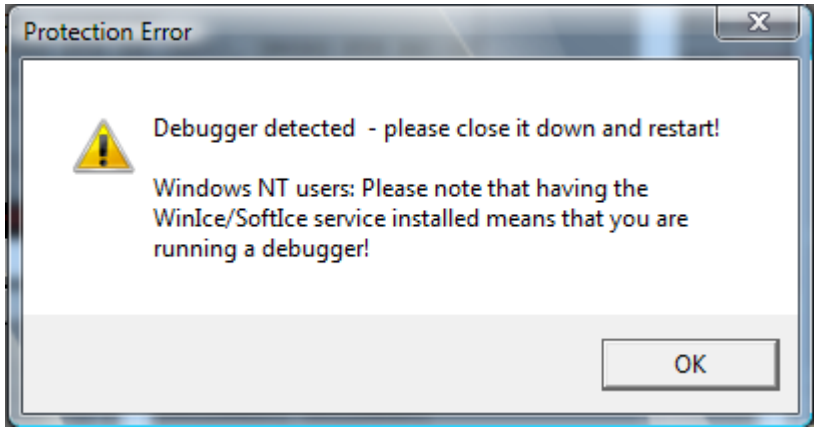
- About Qiling Framework
  - https://qiling.io
  - https://github.com/qilingframework/qiling
  - https://docs.qiling.io
  - @qiling_io

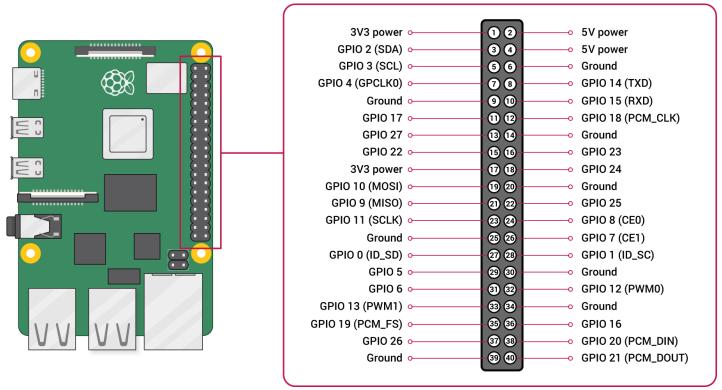


Problems

Current Emulation Limitation



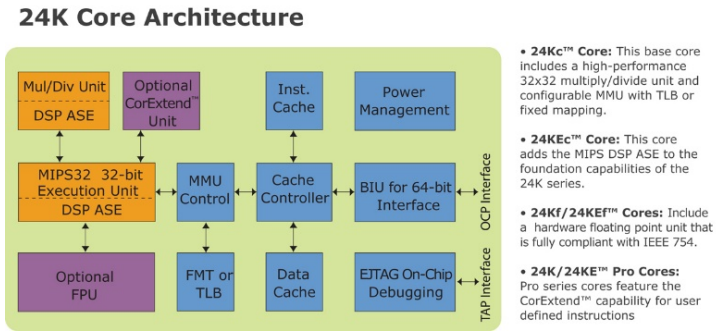


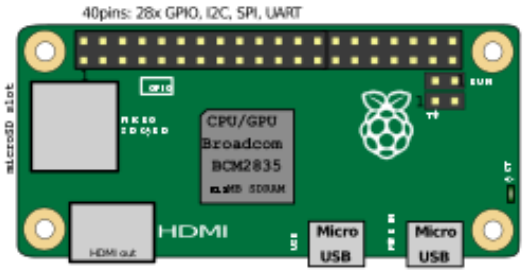


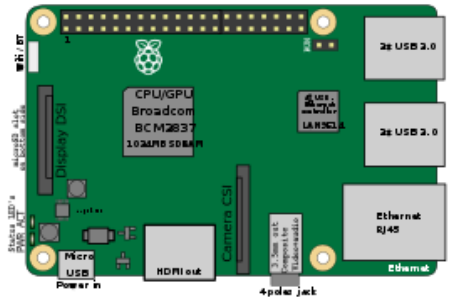
Not only for *nix and Wintel or other Operating System

More platform is either limited or non emulation or analysis tools

Current RE Issues







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

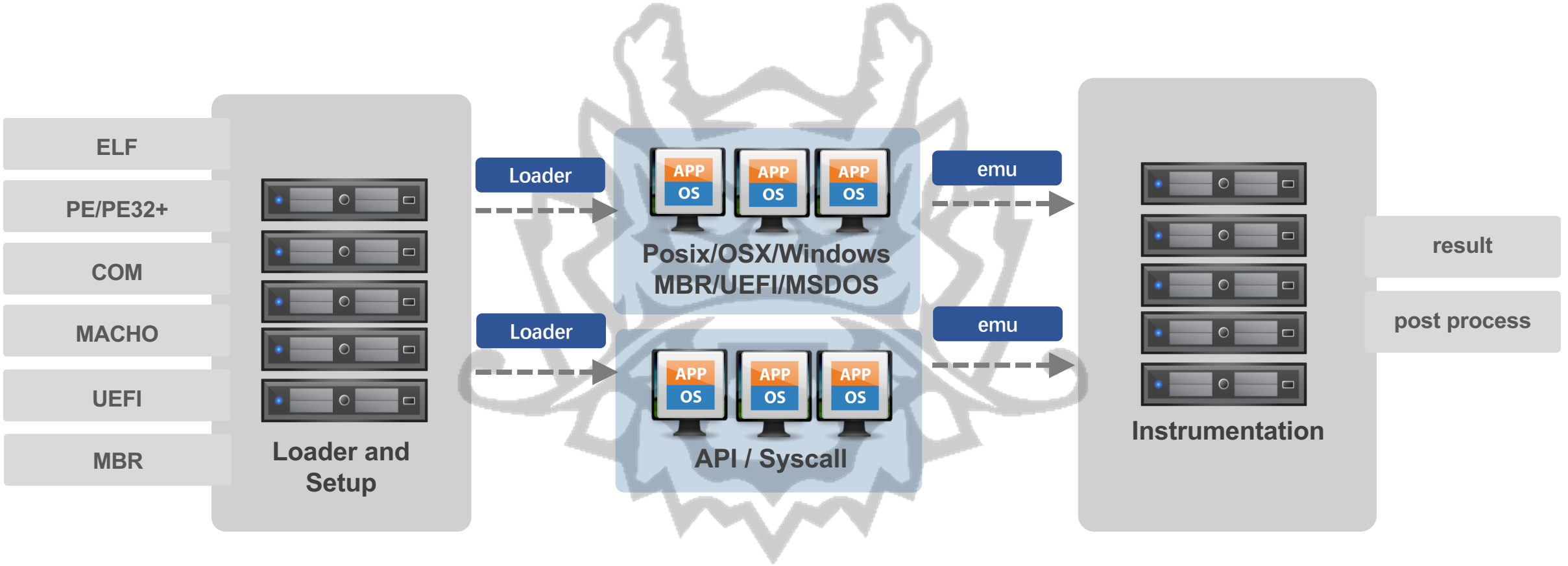
Overview

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)	-	☑	☑	-	☐	-
Linux (ELF)	☐	☑	☑	☑	☑	☑
MacOS (MachO)	-	☐	☑	-	☐	-
BSD (ELF)	☐	☐	☑	☐	☐	☐
UEFI	-	☑	☑	-	-	-
DOS (COM)	☑	-	-	-	-	-
MBR	☑	-	-	-	-	-

How Does It Work



Base OS can be Windows/Linux/BSD or OSX

And not limited to ARCH

Design

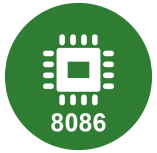
Framework Architecture

Qiling Framework

CPU Architecture | Loader | OS | Debugger | Extensions



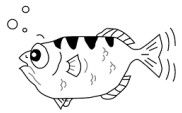


Posix








qdb


GHIDRA


coverage | idaplugin | report | sanitizers | windows sdk

Instrumentation

What are we

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

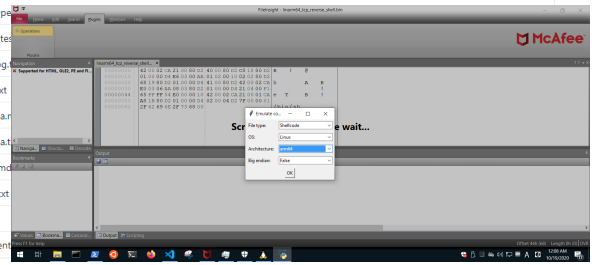
nmantani / FileInsight-plugins

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**

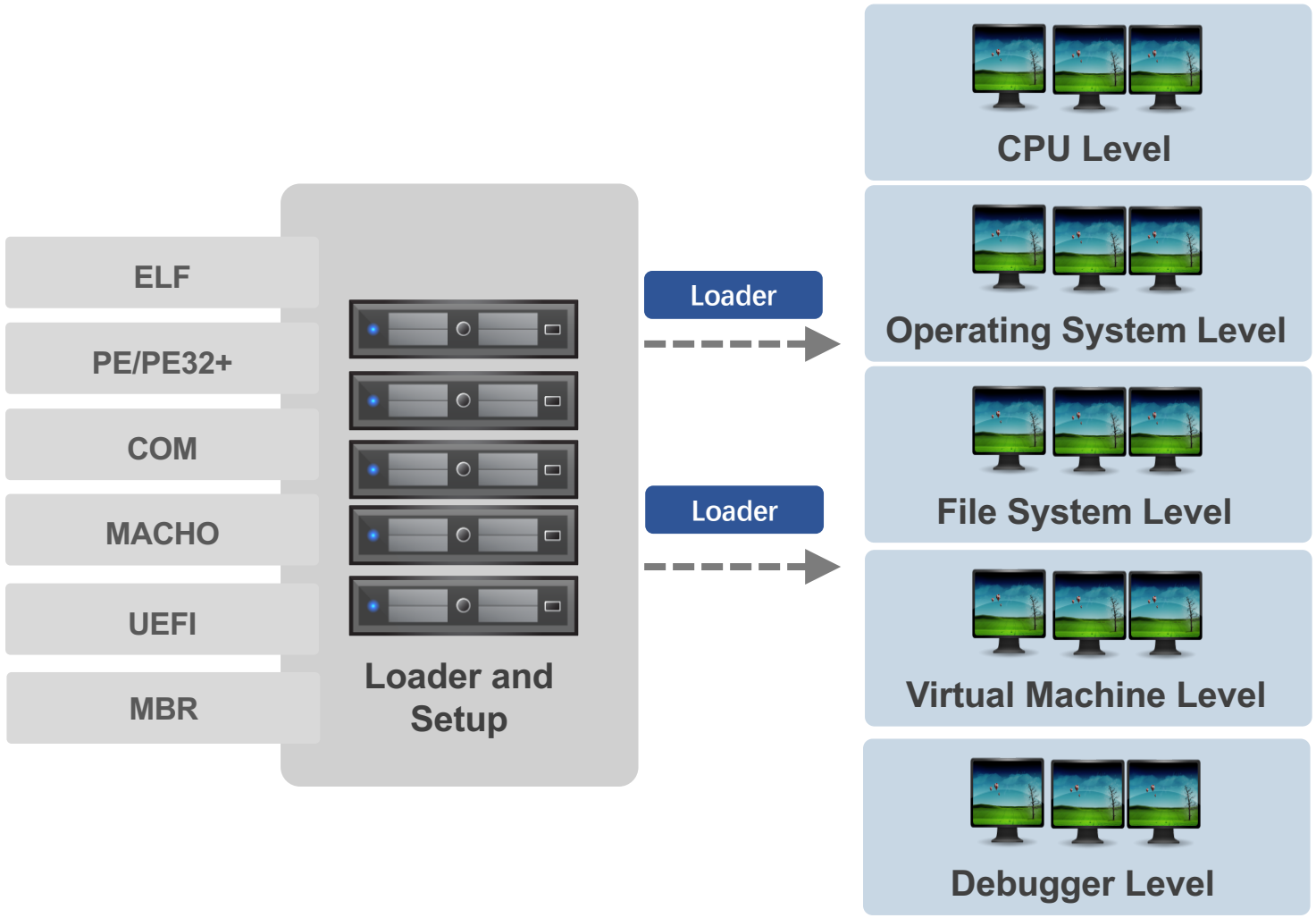
Qiling Framework

CPU Architecture | Loader | OS | Debugger | Extensions

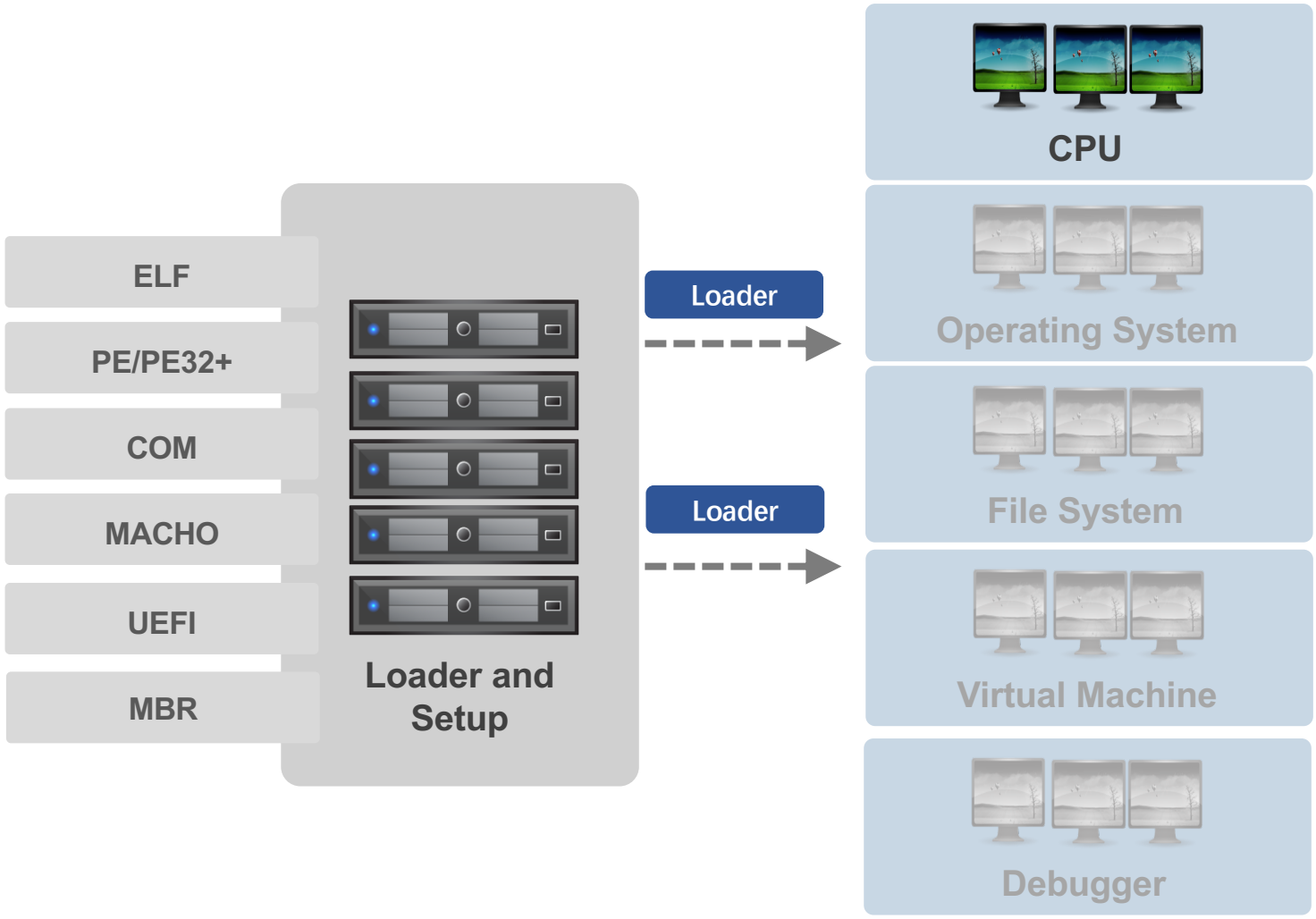
Instrumentation (Qiling's API)

Qiling and APIs

Qiling Framework and Its Interface



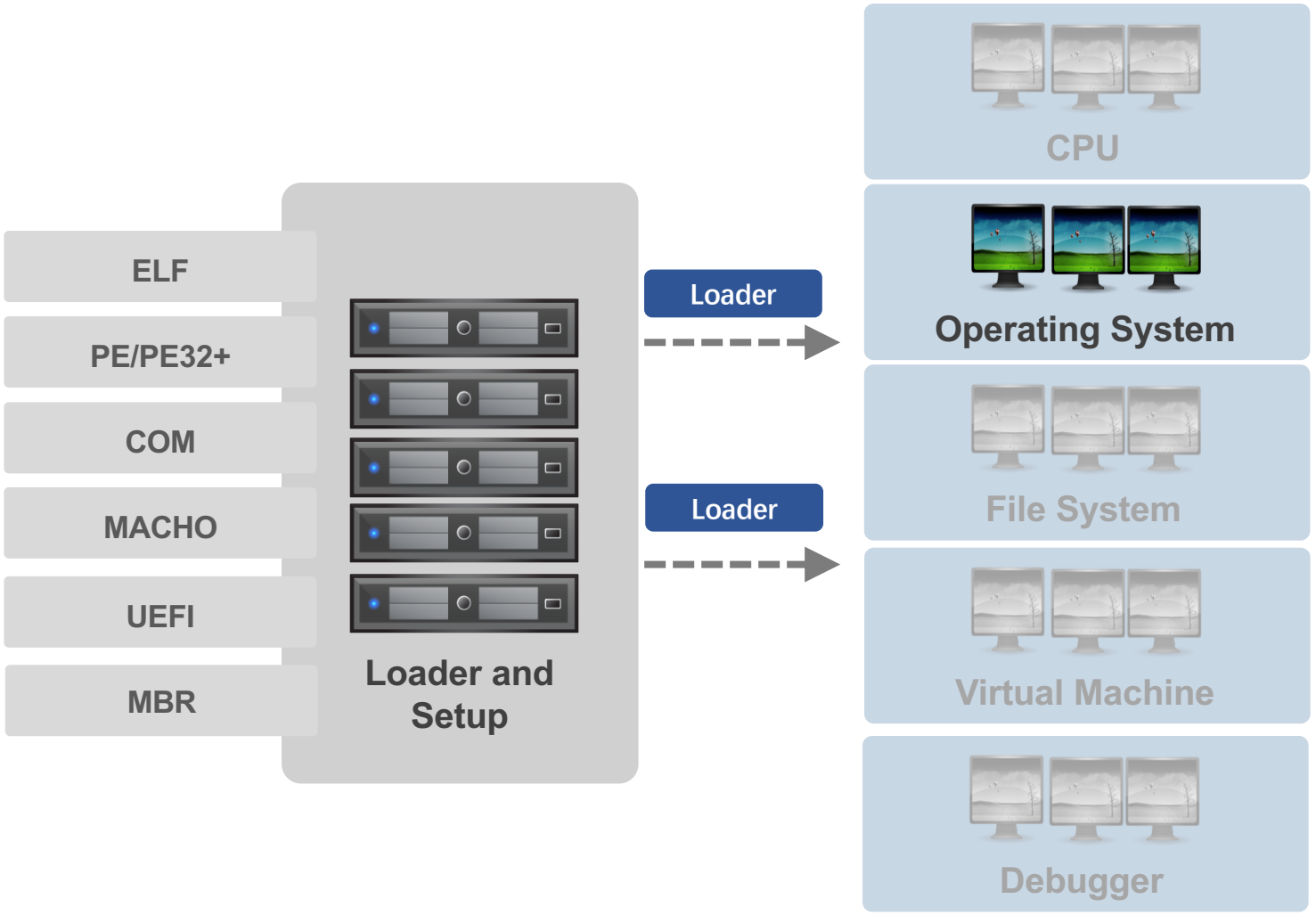
Qiling Framework: CPU



- 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

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

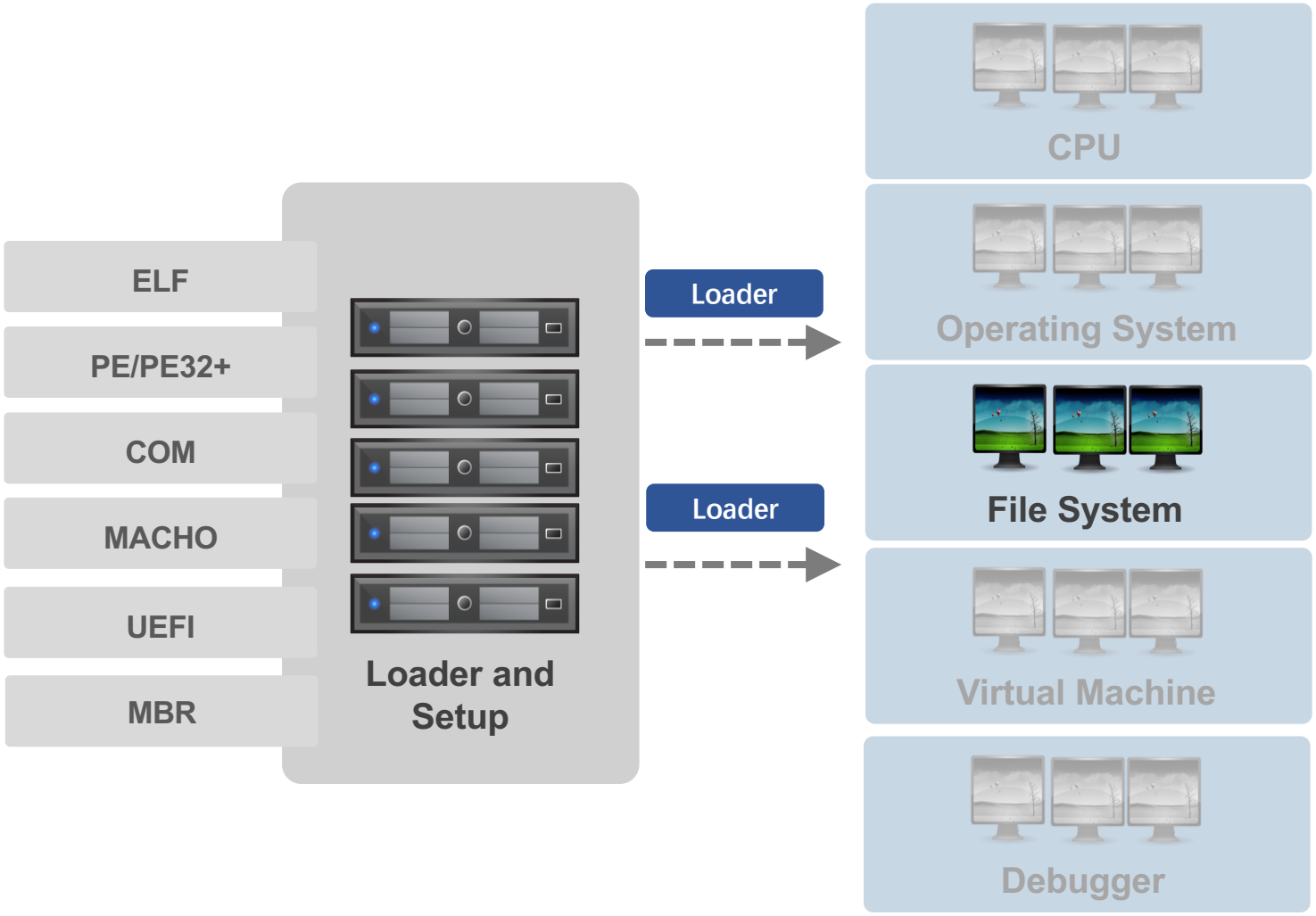
Qiling Framework: Operating System



- 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()

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

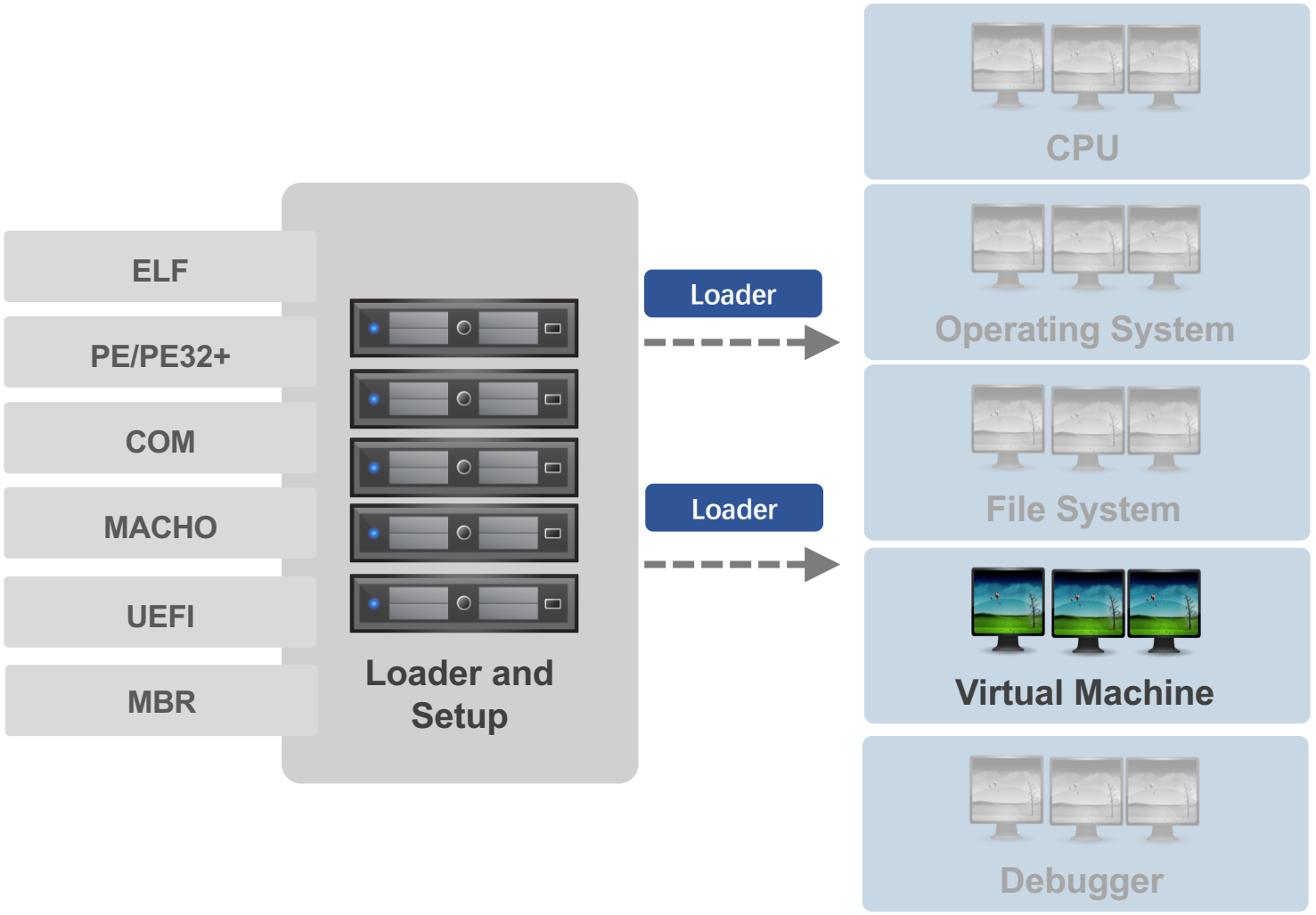
Qiling Framework: File System



- 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

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

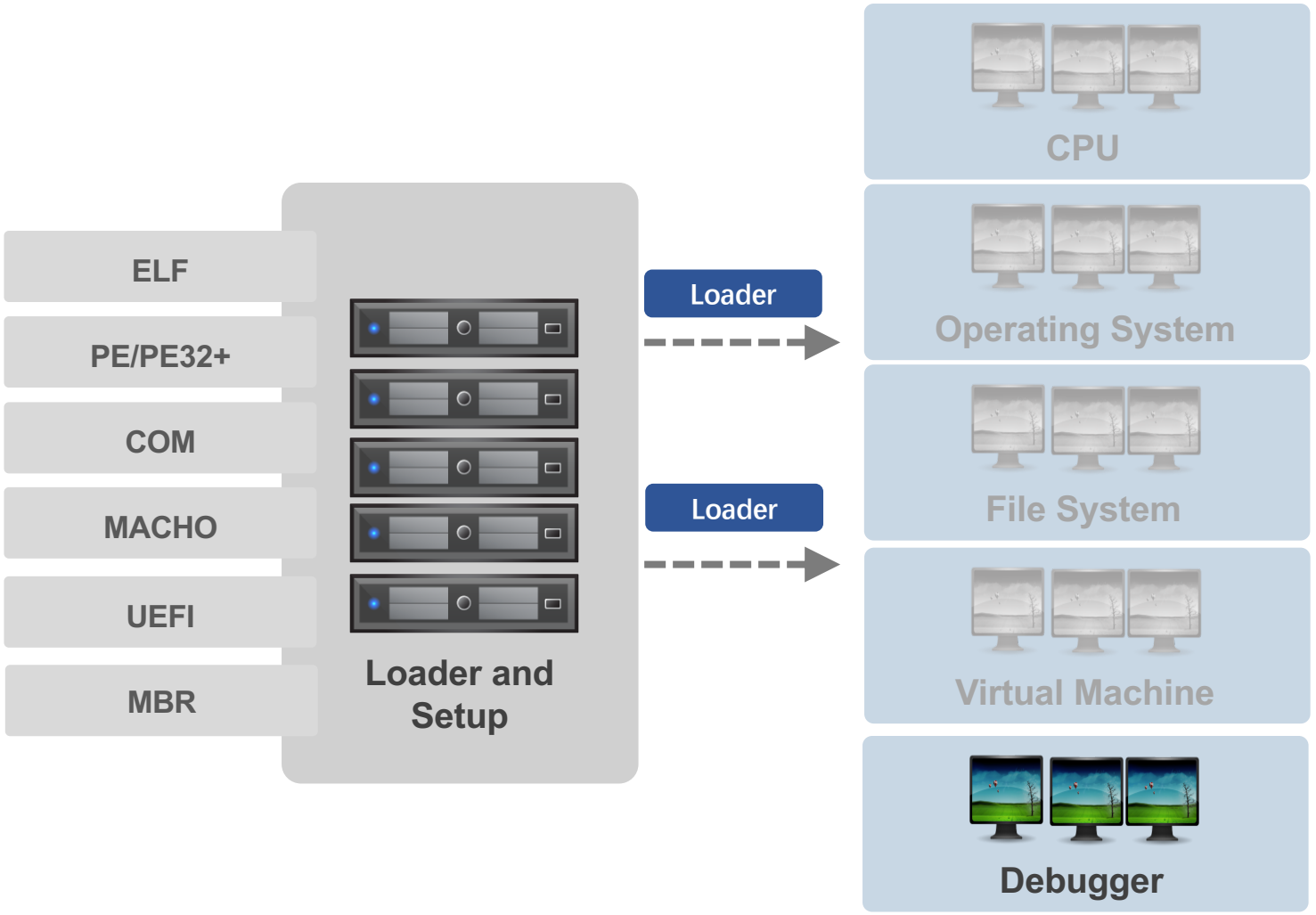
Qiling Framework: Virtual Machine



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

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

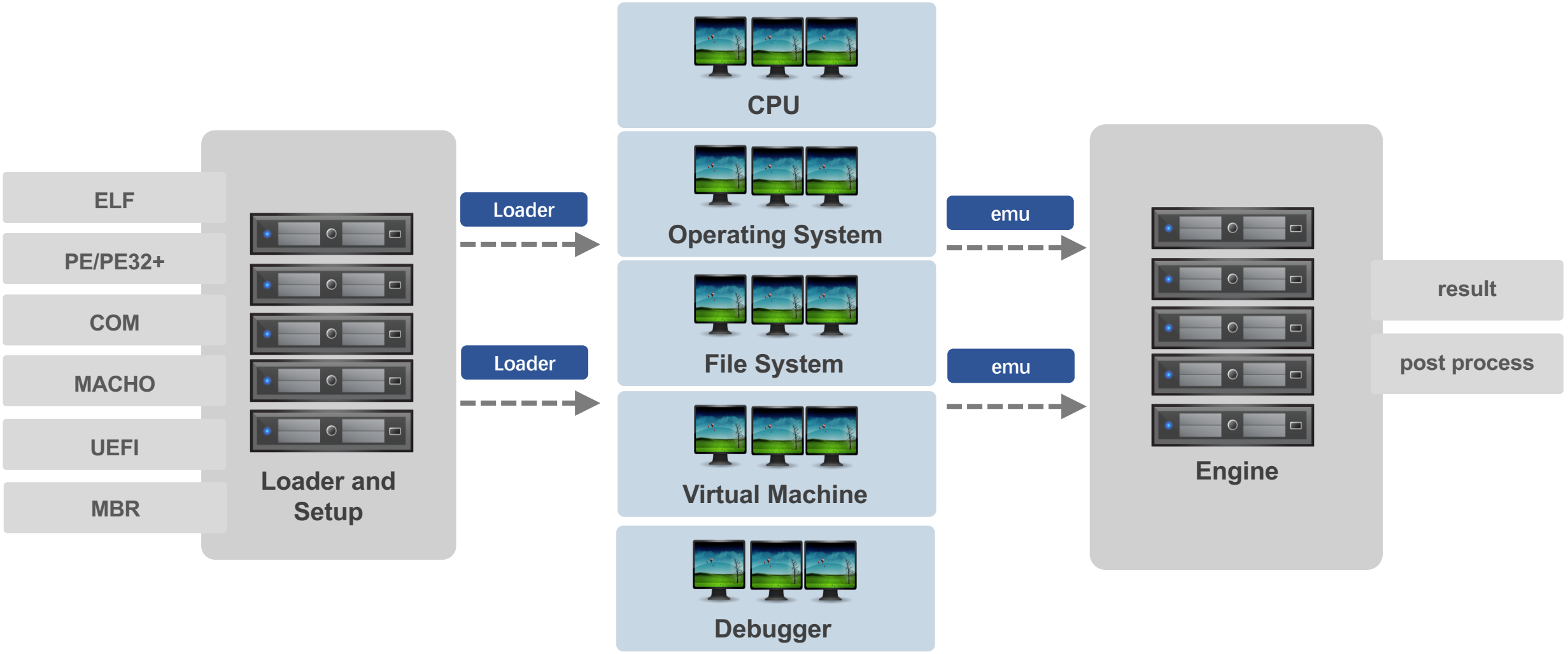
Qiling Framework: Debugger



- 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



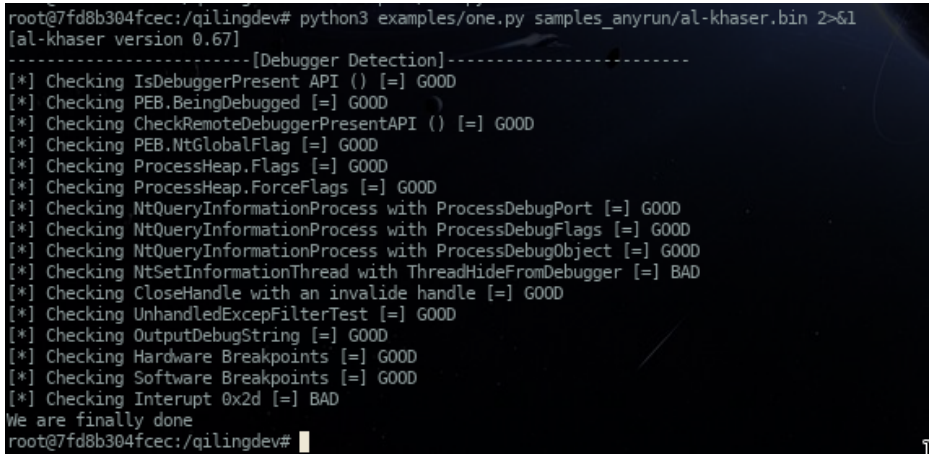
Base OS can be Windows/Linux/BSD or OSX

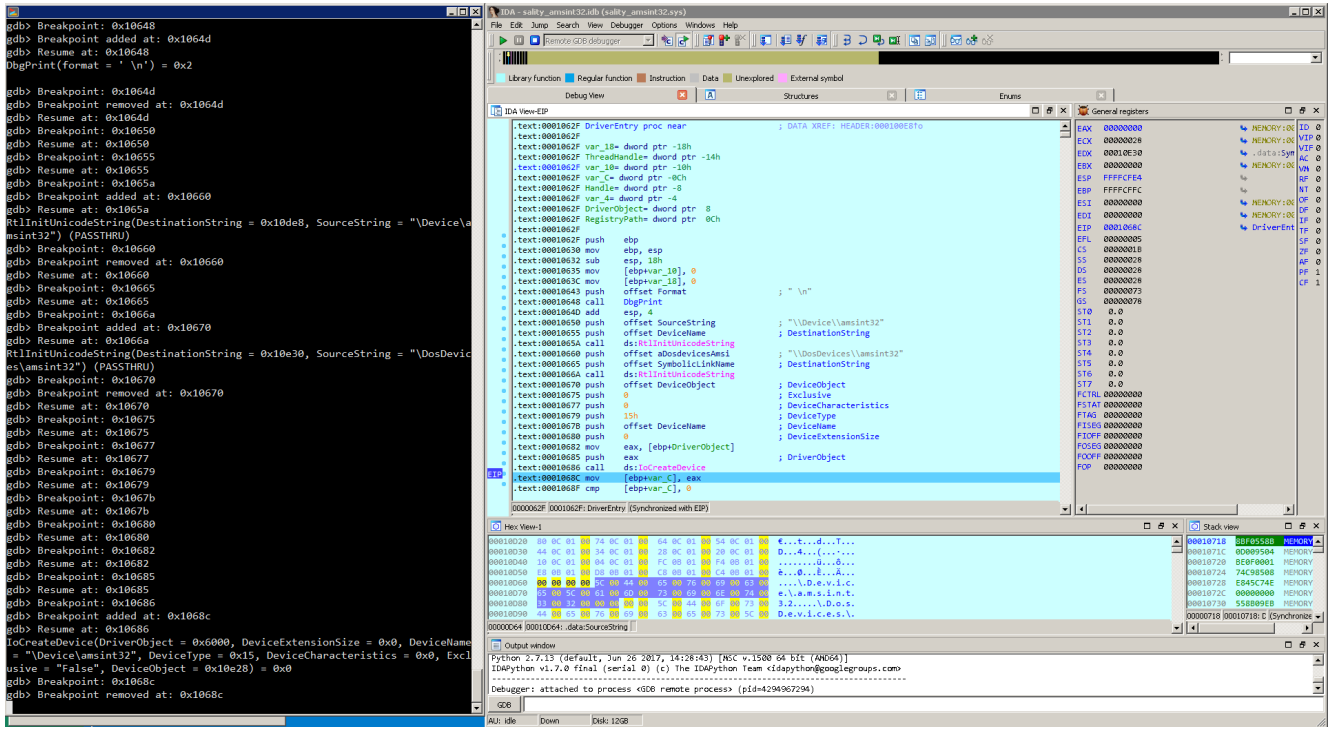
And not limited to ARCH

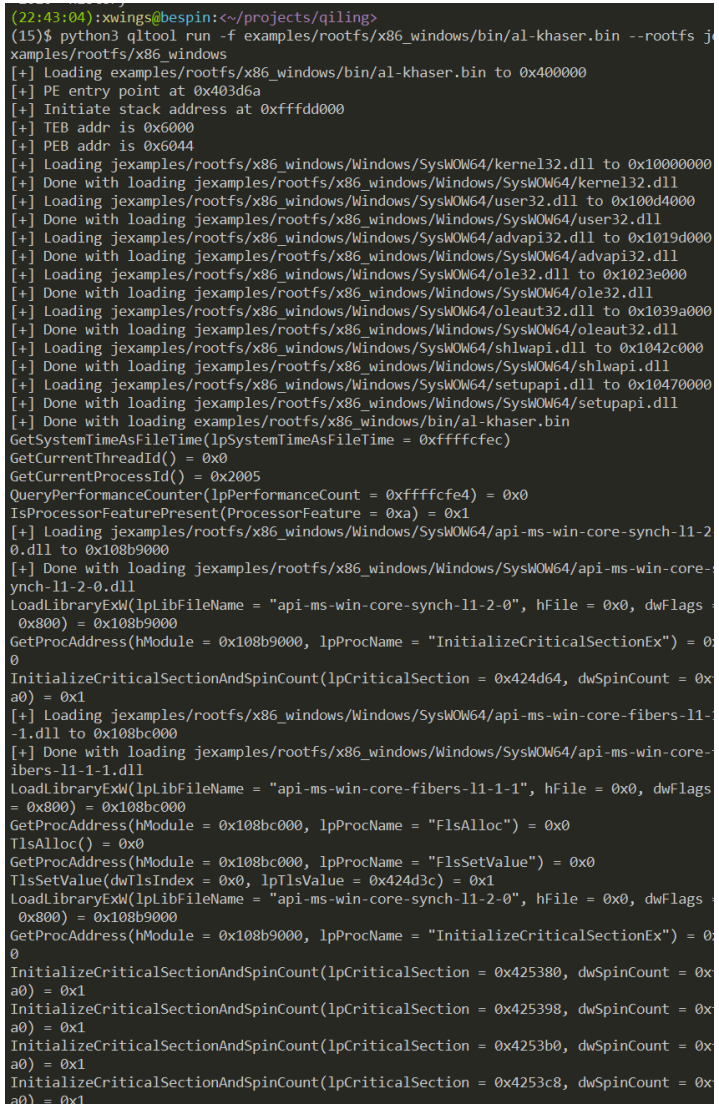
Demo

Malware & Rootkit Analysis

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



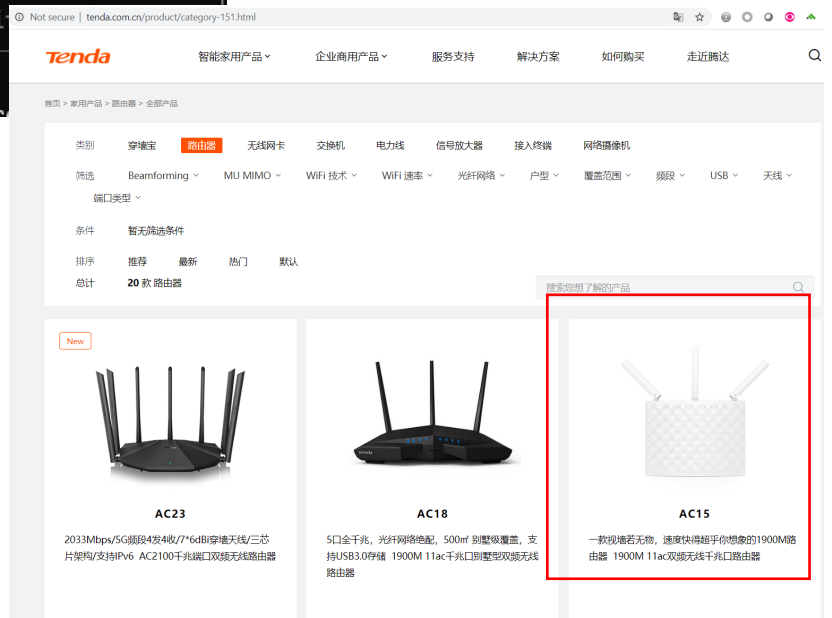




Fuzzer

- 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

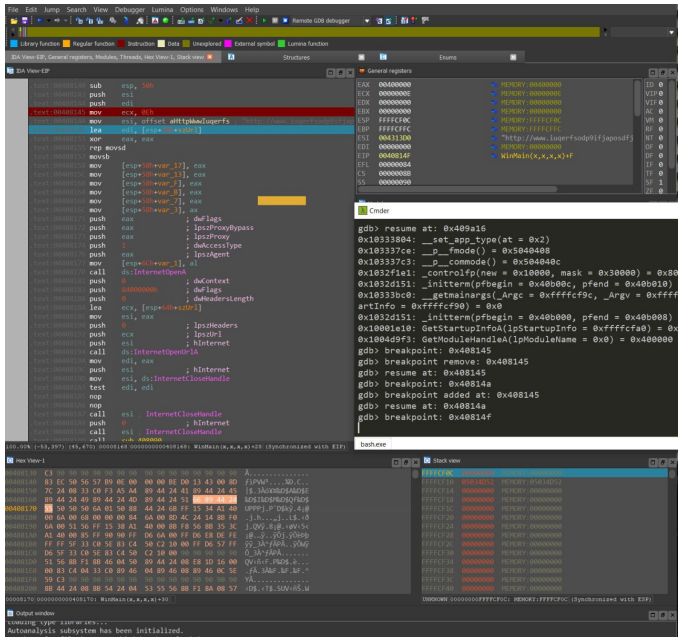
```
american fuzzy lop ++2.65d (python3) [explore] {0}
┌─ process timing ─────────────────────┬─ overall results ─┐
│        run time : 0 days, 0 hrs, 12 min, 52 sec │  cycles done : 2  │
│   last new path : 0 days, 0 hrs, 0 min, 7 sec   │  total paths : 36 │
│ last uniq crash : 0 days, 0 hrs, 0 min,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



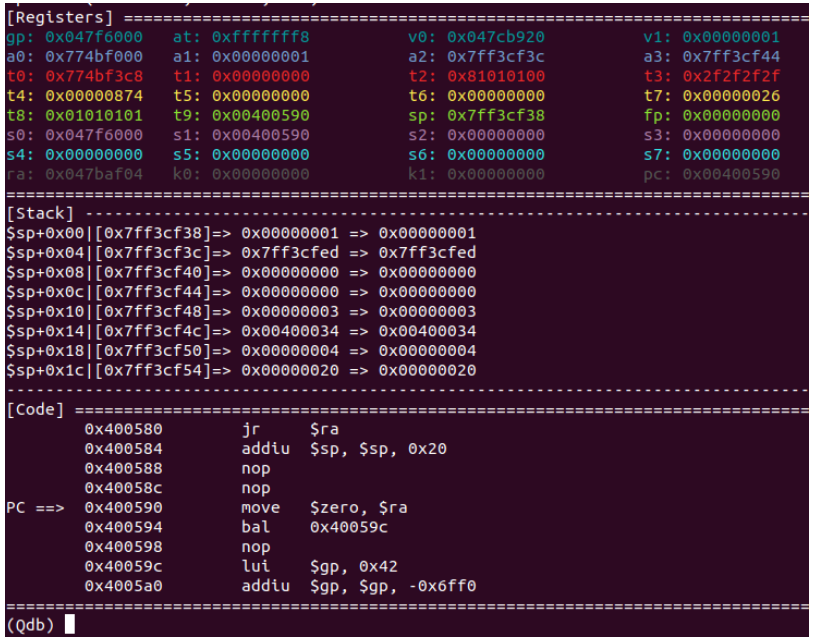
Qiling Debugger

Debugger

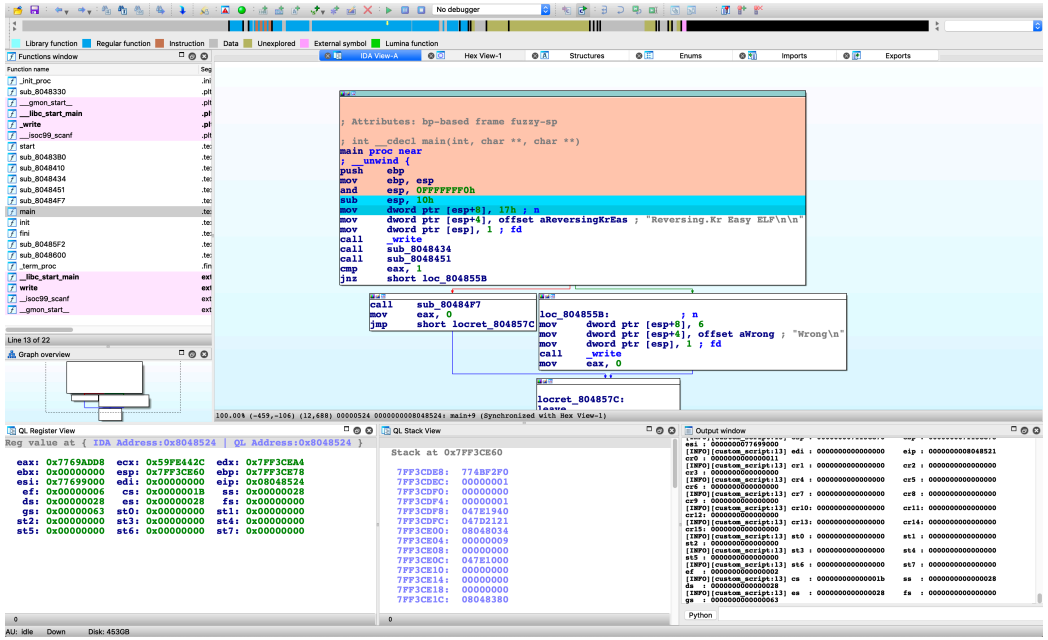
GDB Server



qdb



IDA Plugin

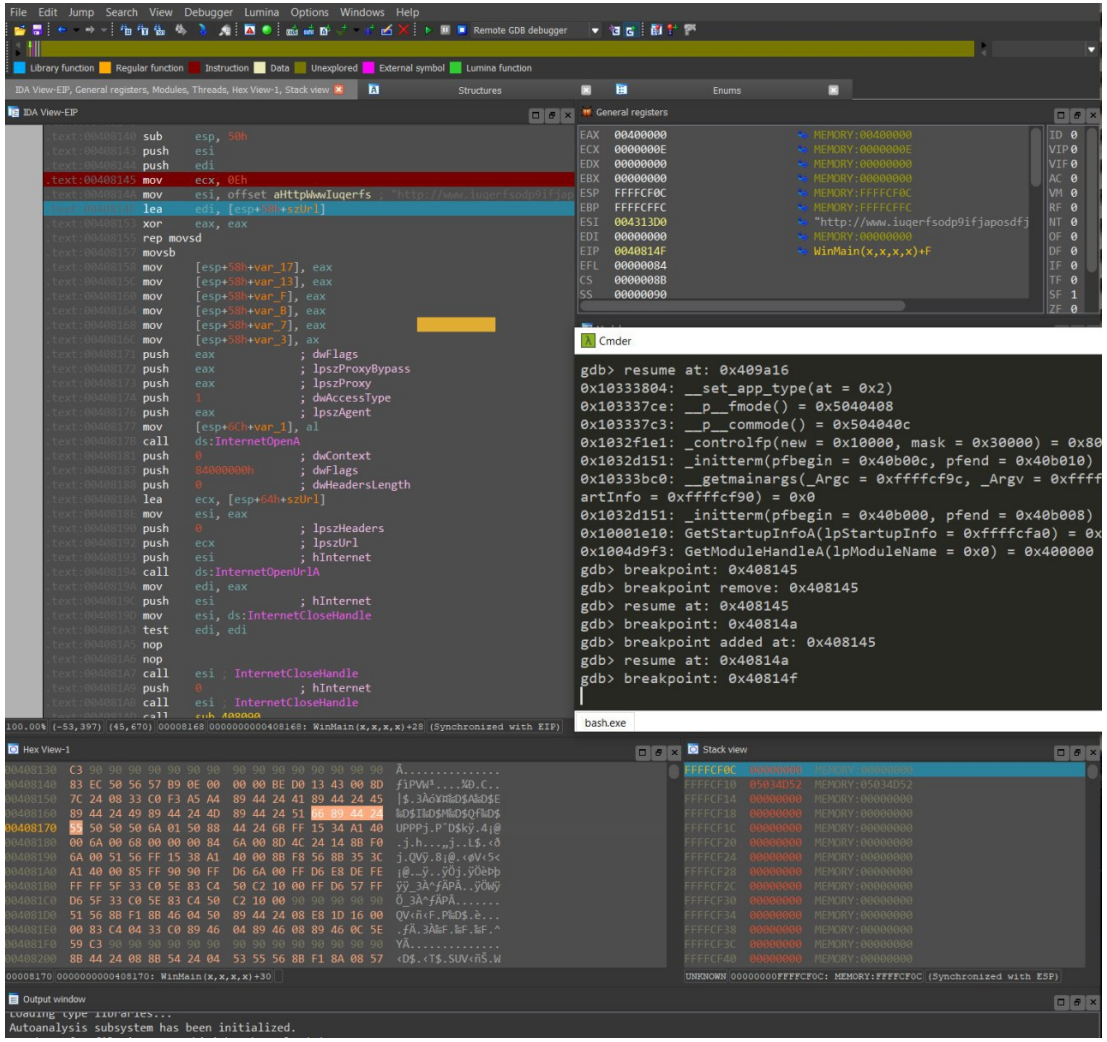


Qiling GDBServer

Terminal

```
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.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com"
(gdb)
```

GUI



Qiling IDA Plugin

- Setup
- Reload User Scripts

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

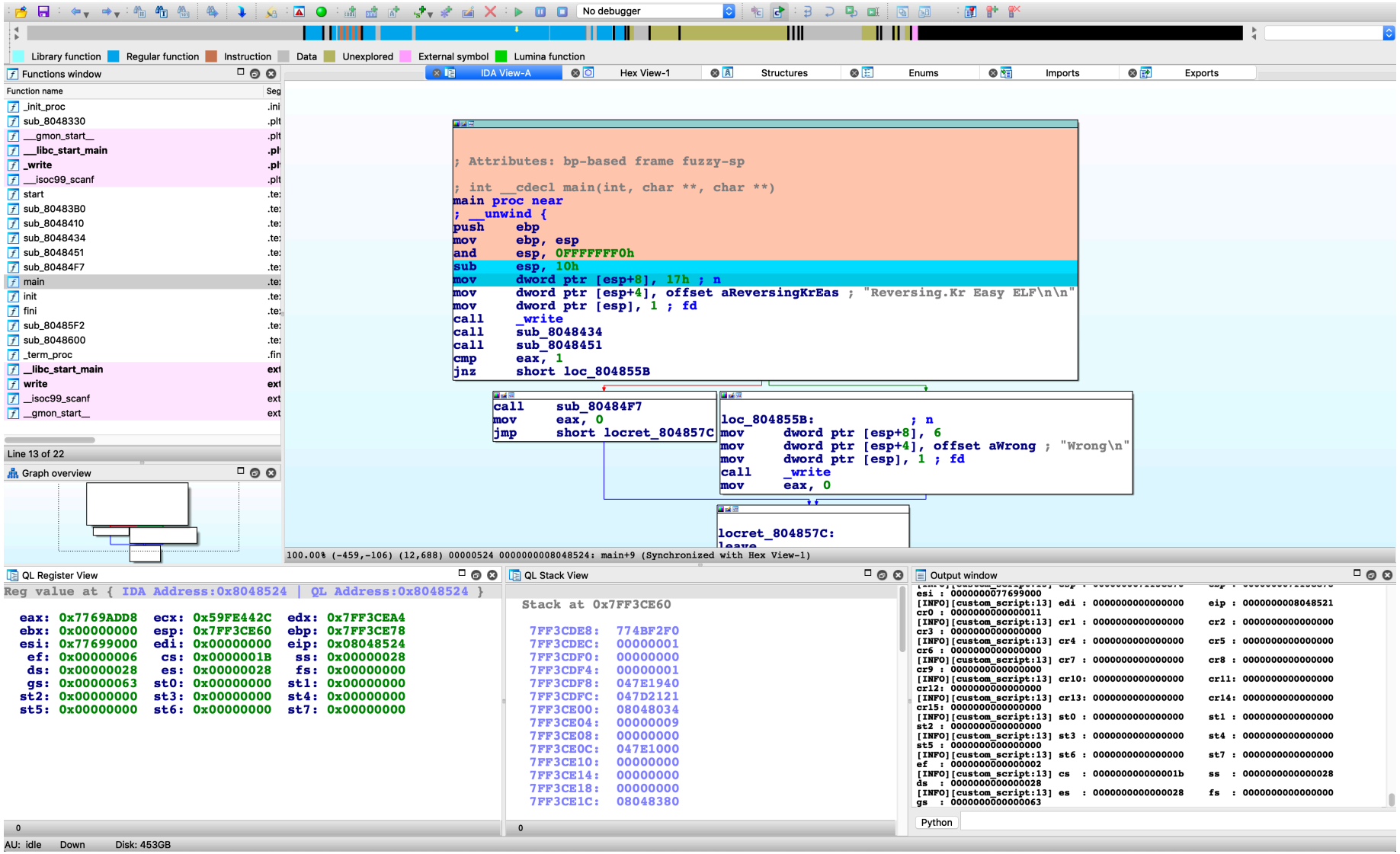
- Restart

- 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



Qiling IDA Plugin custom script

Use all APIs provided by Qiling and IDAPython

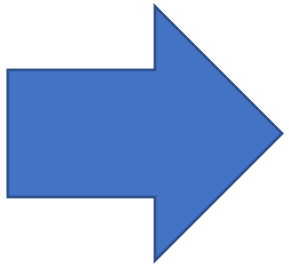

```python
from qiling import *
import logging

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;
}
```

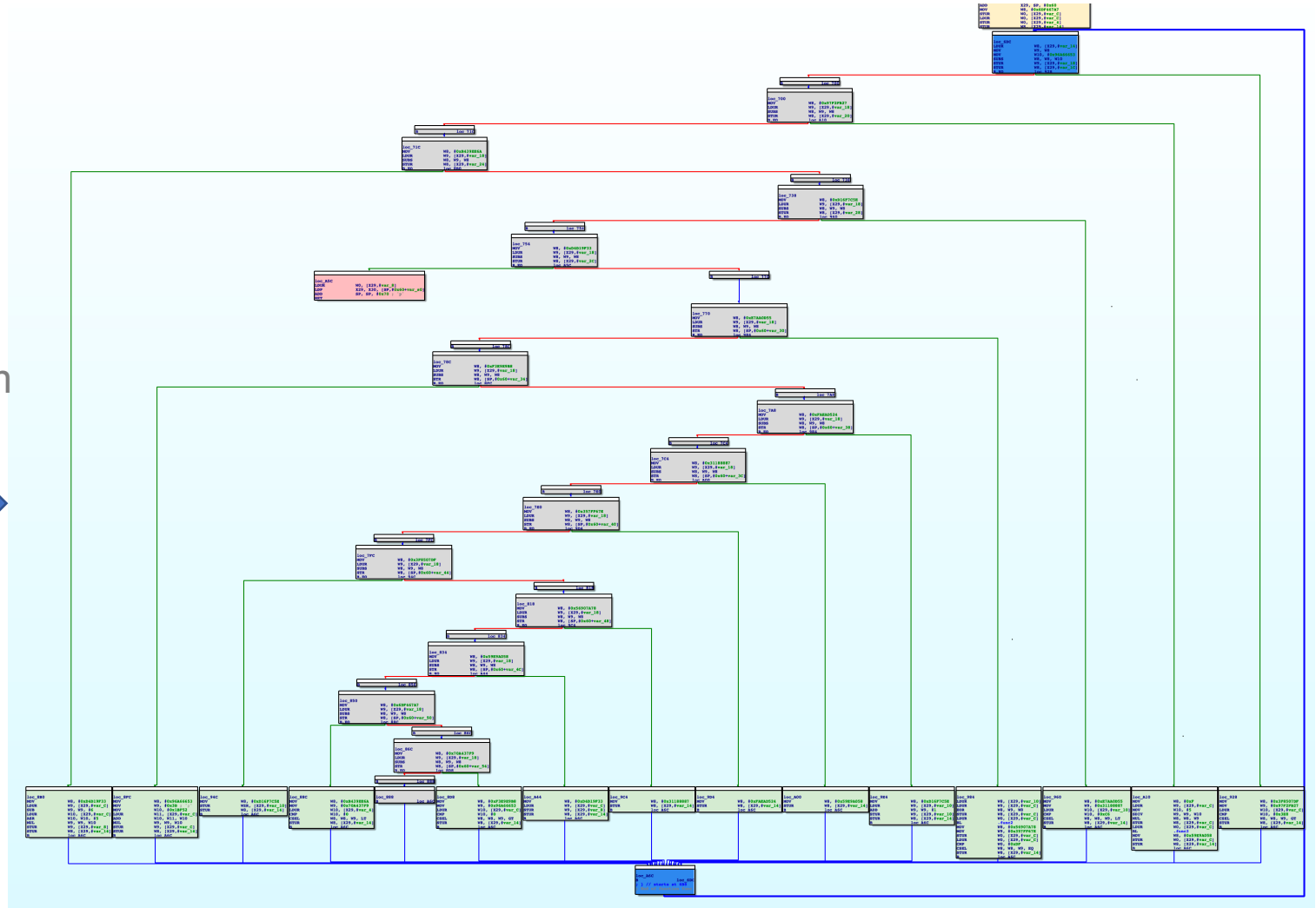


```
int main(int argc, char** argv) {
  int a = atoi(argv[1]);
  int b = 0;
  while(1) {
    switch(b) {
      case 0:
        if(a == 0)
          b = 1;
        else
          b = 2;
        break;
      case 1:
        return 1;
      case 2:
        return 10;
      default:
        break;
    }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

De-flatten: Sample

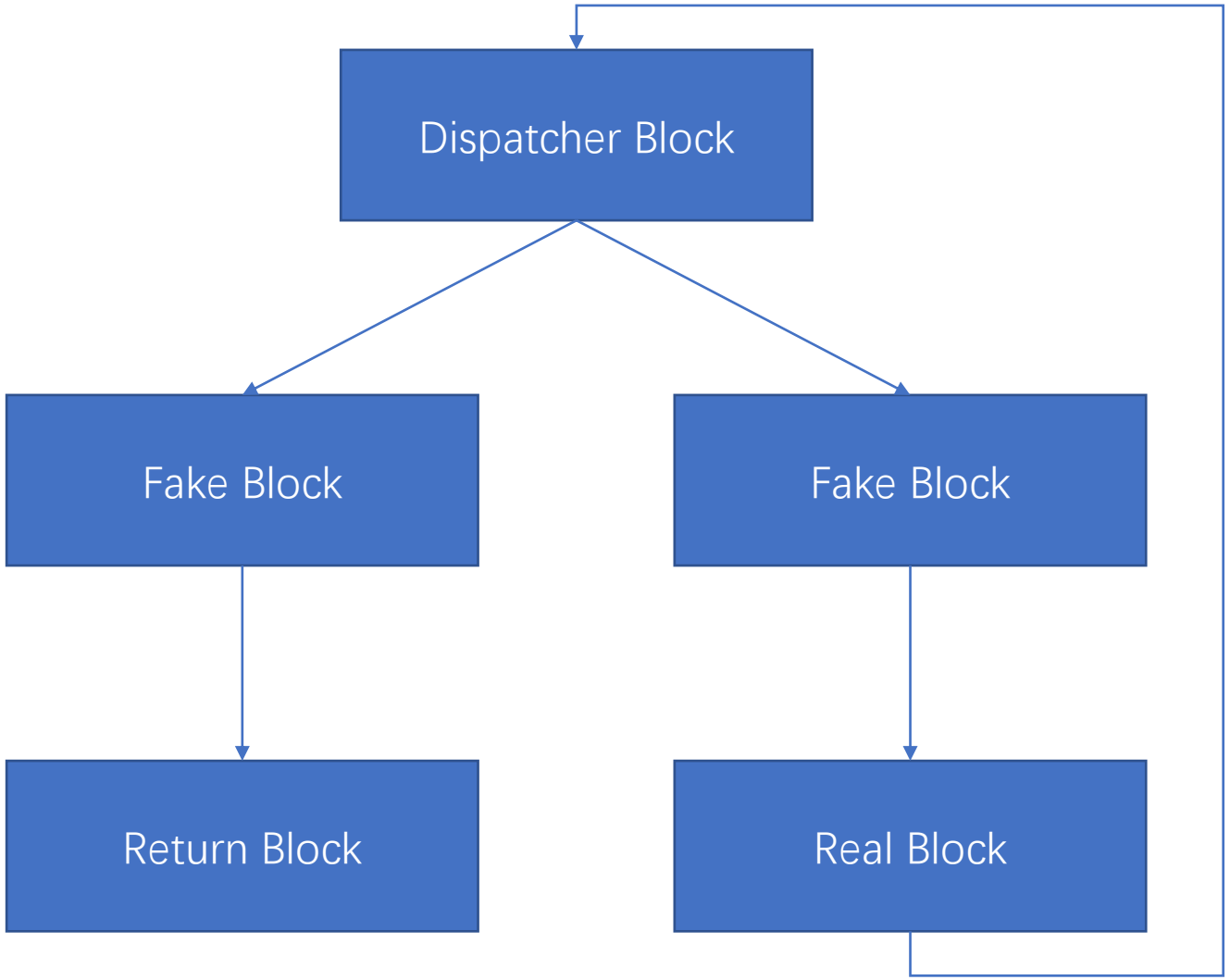
```
int func3(int v){
        return v+v/2+v*v;
}

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



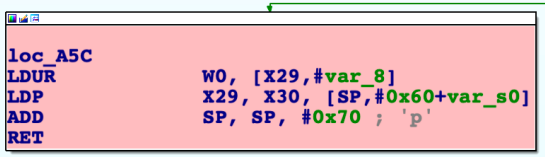
De-flatten: Identification

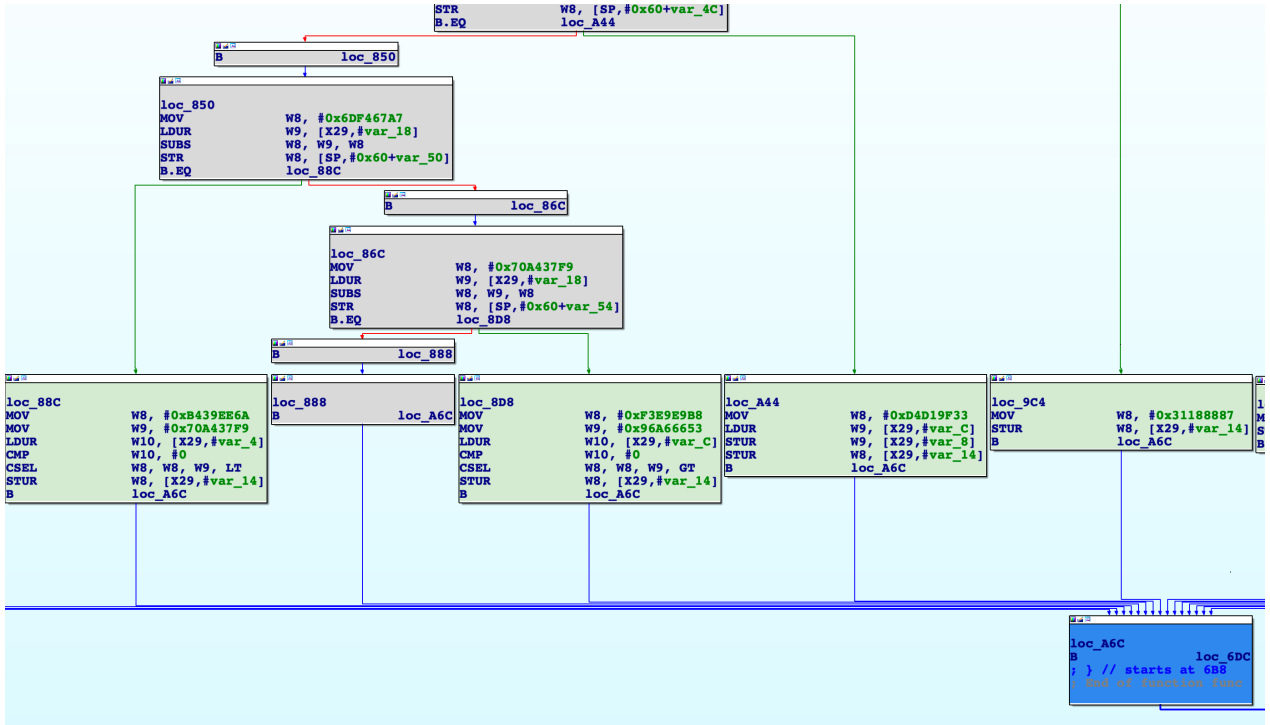
- Four types of blocks
- Real Blocks
  - Only contain original logic
  - May contain zero or only one conditional statement.
- Fake Blocks
  - Only responsible for switch-case implementation.
- Return Blocks
  - Return the function.
- Dispatcher Blocks
  - Equivalent to switch statement.
  - Decide the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



De-flatten: Identification

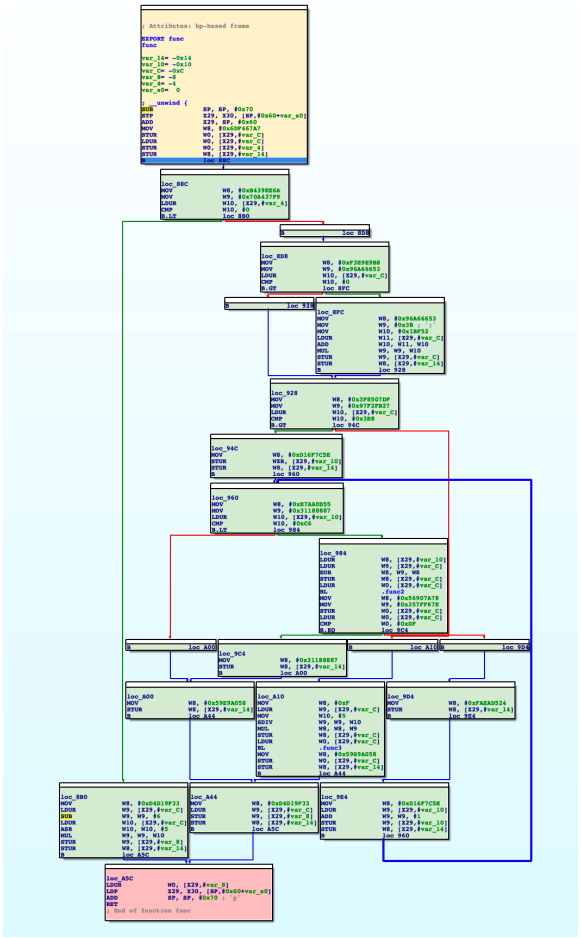
- The successor of all real blocks is the dispatcher block.
  - Locate dispatcher blocks by reference counts.
- The blocks with more than one instructions whose successor is the dispatcher can be identified as real blocks.
- The blocks which has any return statement like retn is considered as return 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 i; // [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.
  - cmov, 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                                 ; CODE XREF: func+1AC↑j
.text:000000000000088C                 MOV     W8, #0xB439EE6A
.text:0000000000000894                 MOV     W9, #0x70A437F9
.text:000000000000089C                 LDUR    W10, [X29,#var_4]
.text:00000000000008A0                 CMP     W10, #0
.text:00000000000008A4                 CSEL    W8, W8, W9, LT
.text:00000000000008A8                 STUR    W8, [X29,#var_14]
.text:00000000000008AC                 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
```

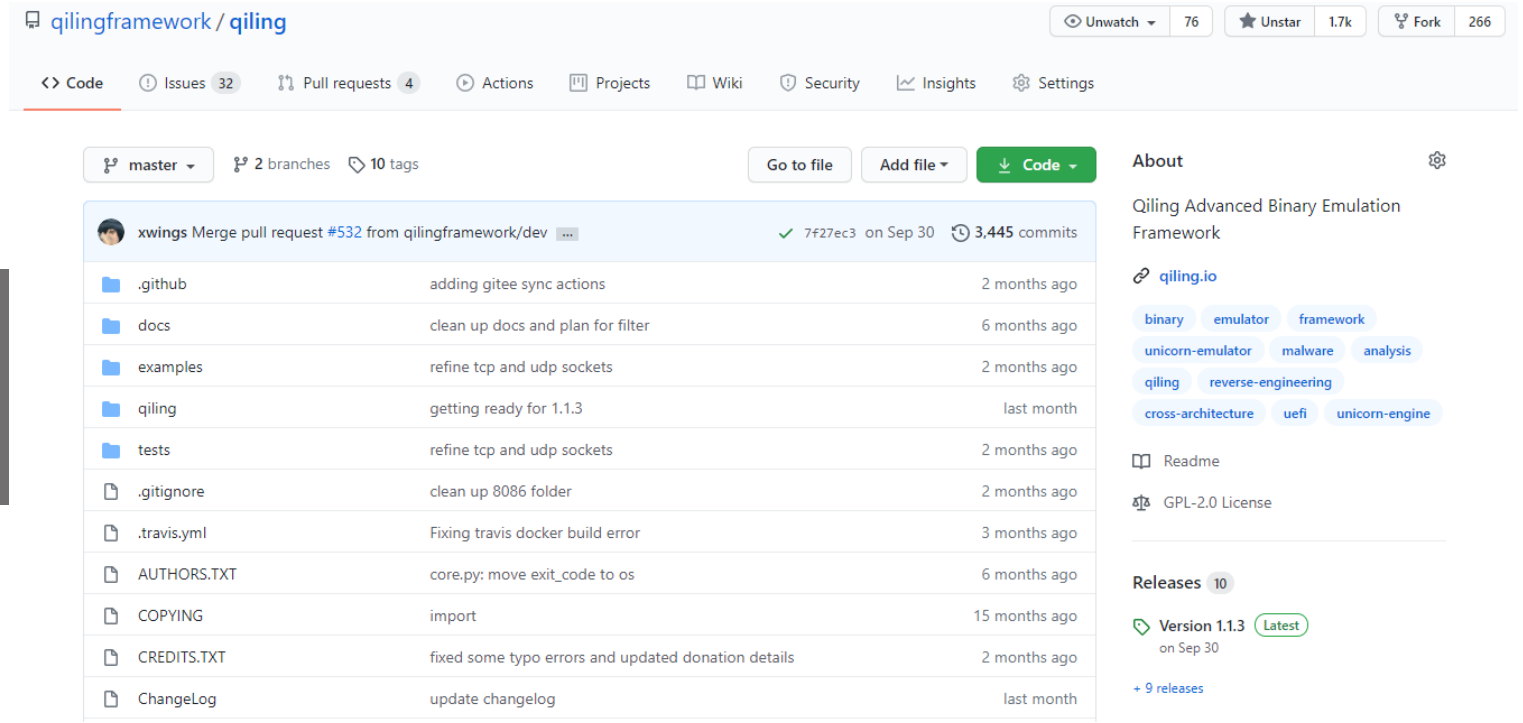
Future

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

Star us



https://docs.qiling.io