Qiling Framework: Blackhat Asia 2021

May 2021

twitter: @qiling_io https://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

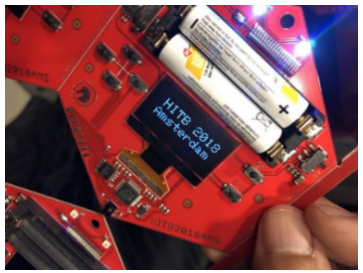
- https://qiling.io
- Lead Developer
- Founder

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, Windows and FreeBSD
- High-level Python API access to register, CPU and memory
- 2,200+ Github star, more than 13,000+ pypi download, 70+ contributors worldwide

About lazymio && kabeor

~ $ 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

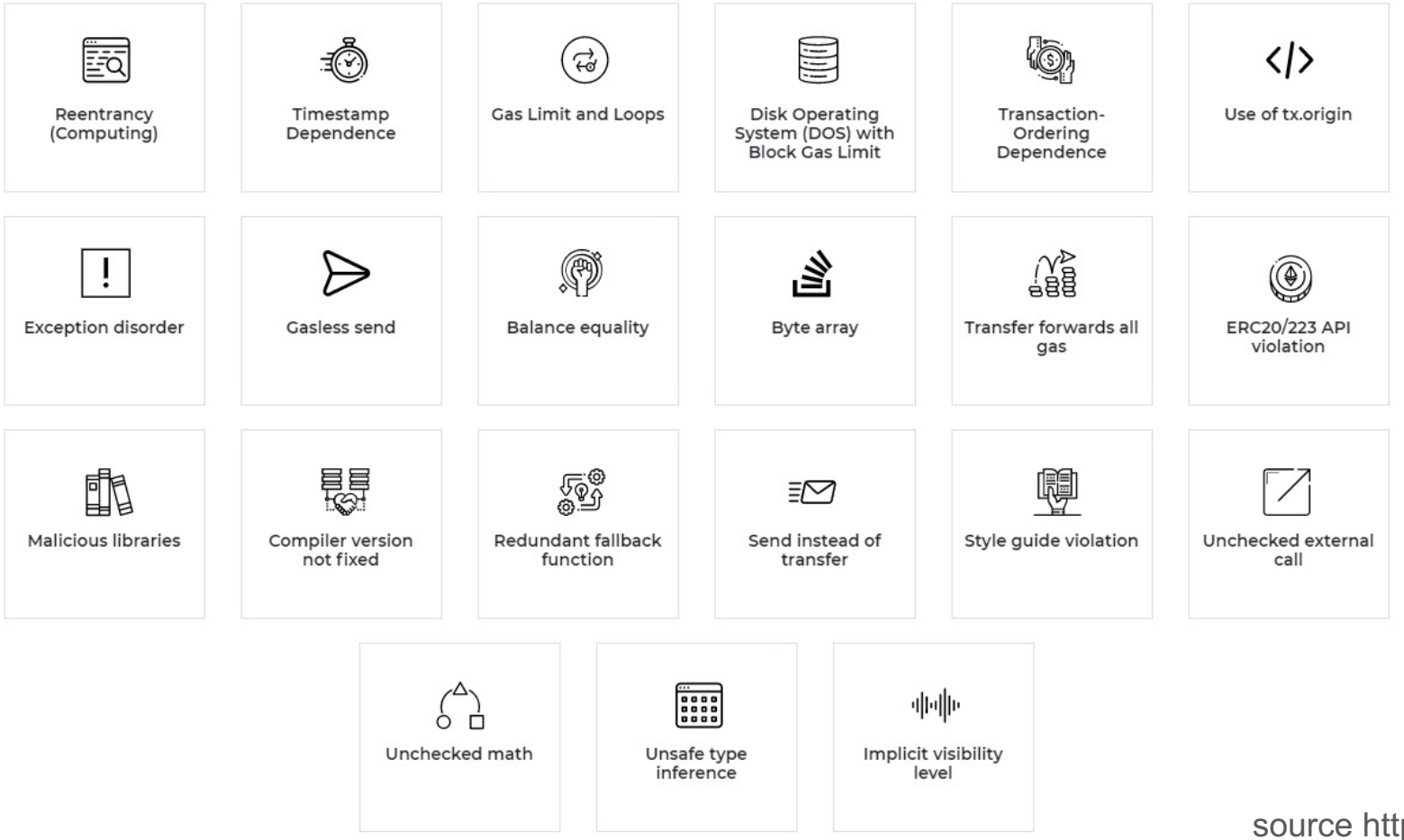
Github: https://github.com/kabeor

Blog: https://kabeor.cn

Twitter: https://twitter.com/Angrz3_K

Make Smart Contract Analysis Smarter

Greater Functions Comes With Greater Bugs



source https://www.developcoins.com/

> Various types of vulnerabilities

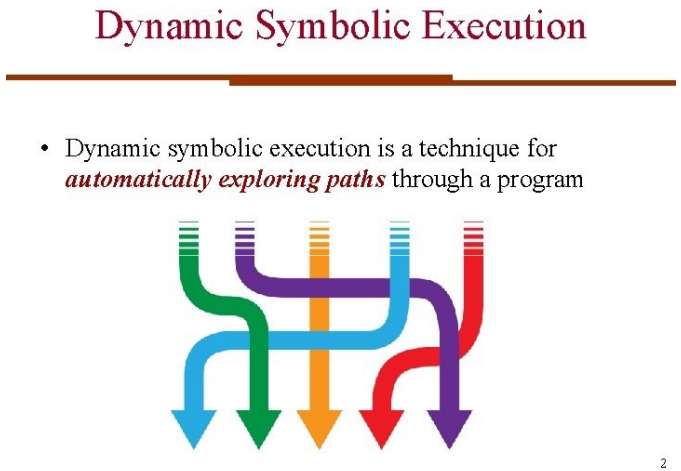
> More complicated after DeFi

> 109B DeFi Market Cap, as of April 2021

> 22B USD thief in 2019/2020

Today's Smart Contract Analysis Problems

- Binary Only Contracts
- Complex Symbolic Execution
- High False Positive
- Require Human Analysis



Dynamic cross contract emulation and debug is almost impossible

Not to mention close source smart contract

Wait, There are Official Emulator

Current Emulator, Symbolic Execution Limitation

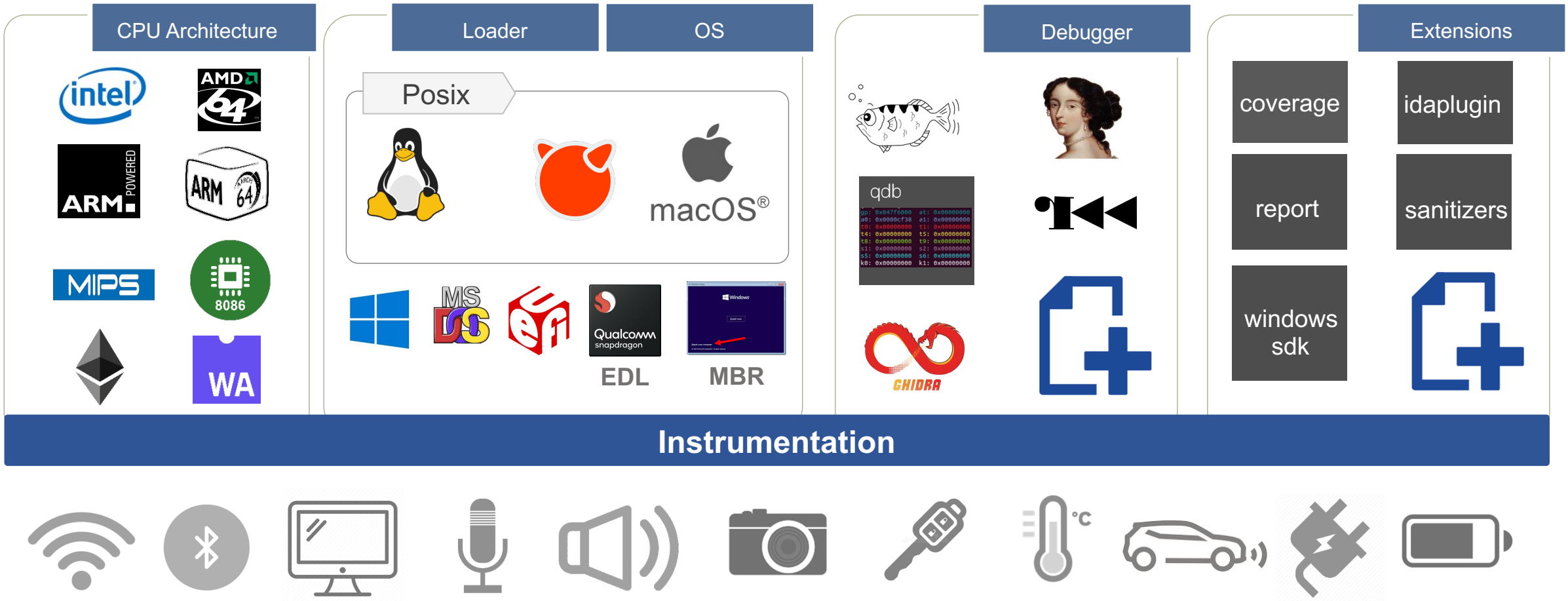
What Is Missing

- Dynamic Execution Hook
- Conditional Execution
- Contract Only Fuzzing
- Pattern Execution
- Live Debugging
- Real Instrumentation
- Not a Framework

99% of the smart contract enabled block chain are EVM/WASM

Qiling Framework

Overview


CPU Architecture
Loader
OS
Debugger
Extensions
Posix
macOS®
qdb
EDL
MBR
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

CARDANO

ETH 2.0

	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

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

Smart Contract SDK

- Design for programmer
- Lack of debugging tools
- Not multi chain friendly
- Development mindset

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

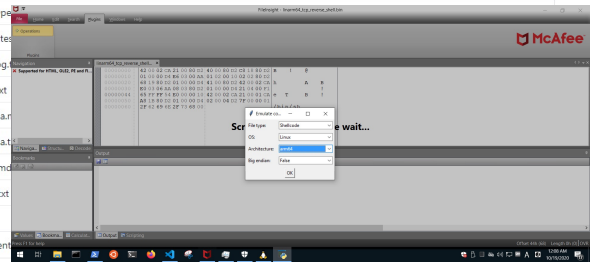
nmantani / FileInsight-plugins

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

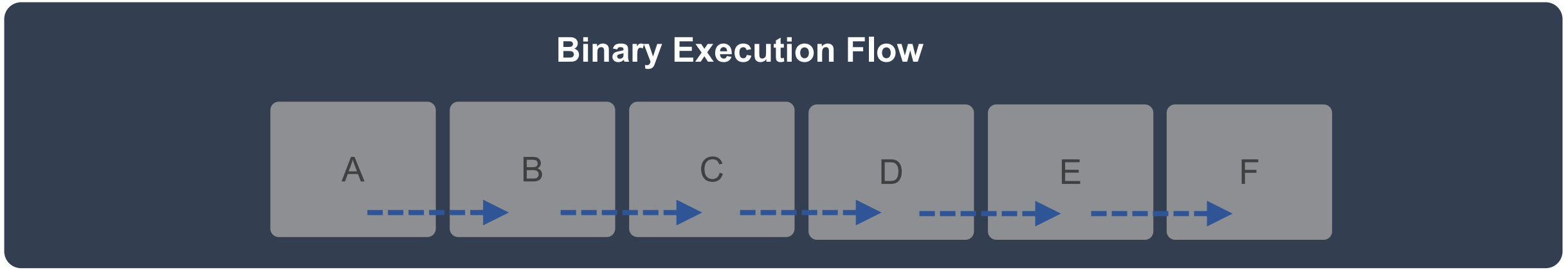
Qiling Framework

CPU Architecture | Loader | OS | Debugger | Extensions

Instrumentation (Qiling's API)

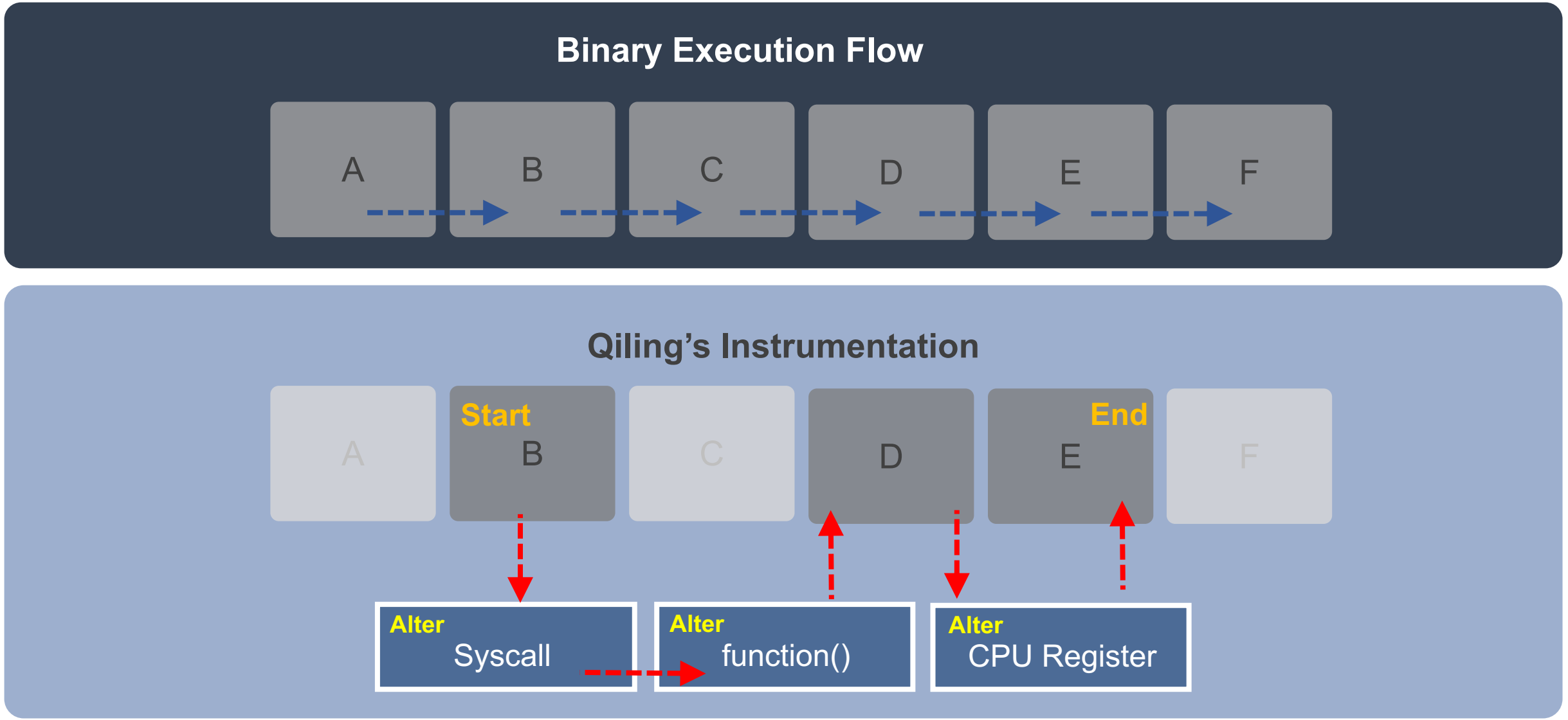
Instrumentation

What Is Instrumentation



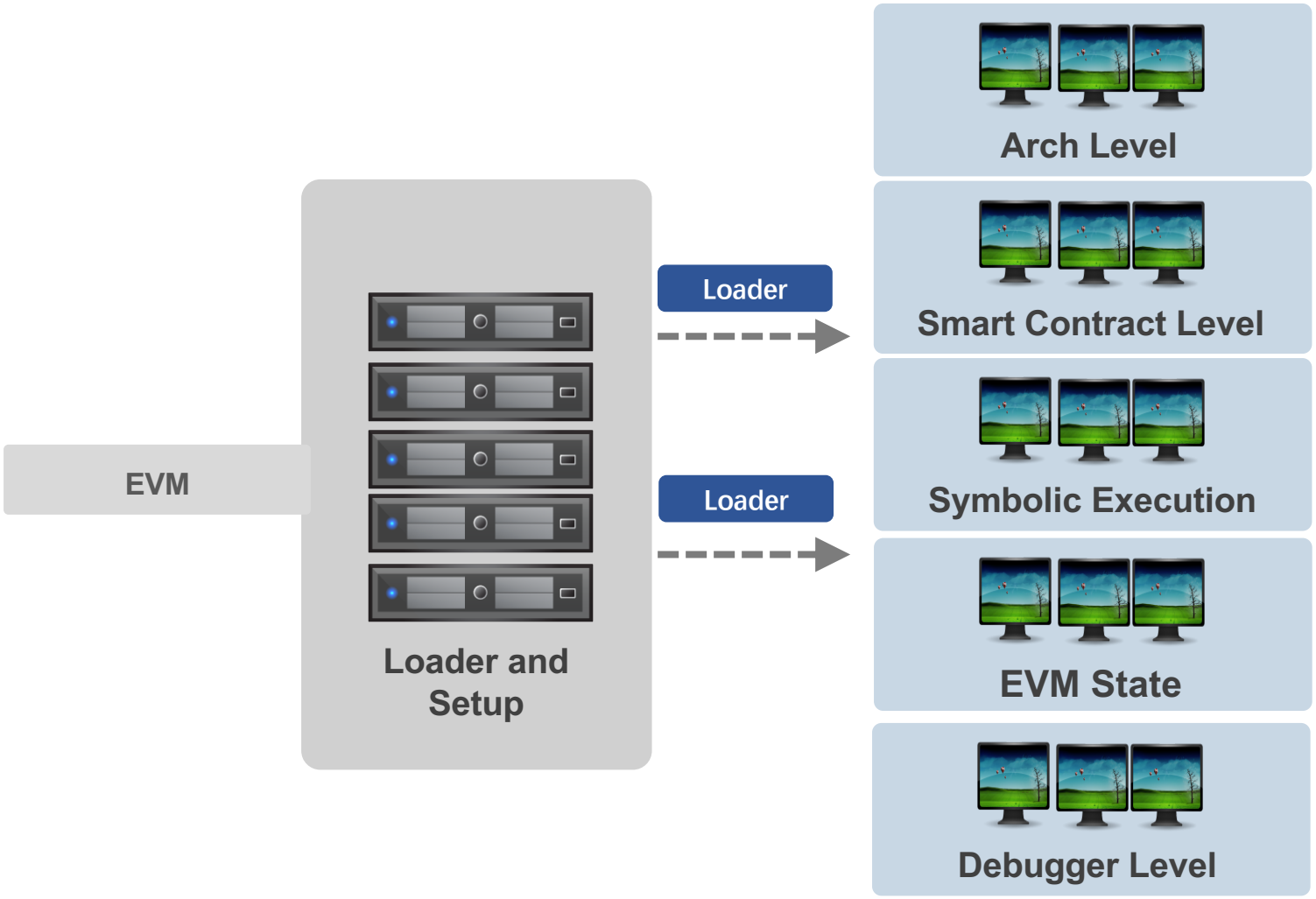
One Function After Another

What Is Instrumentation



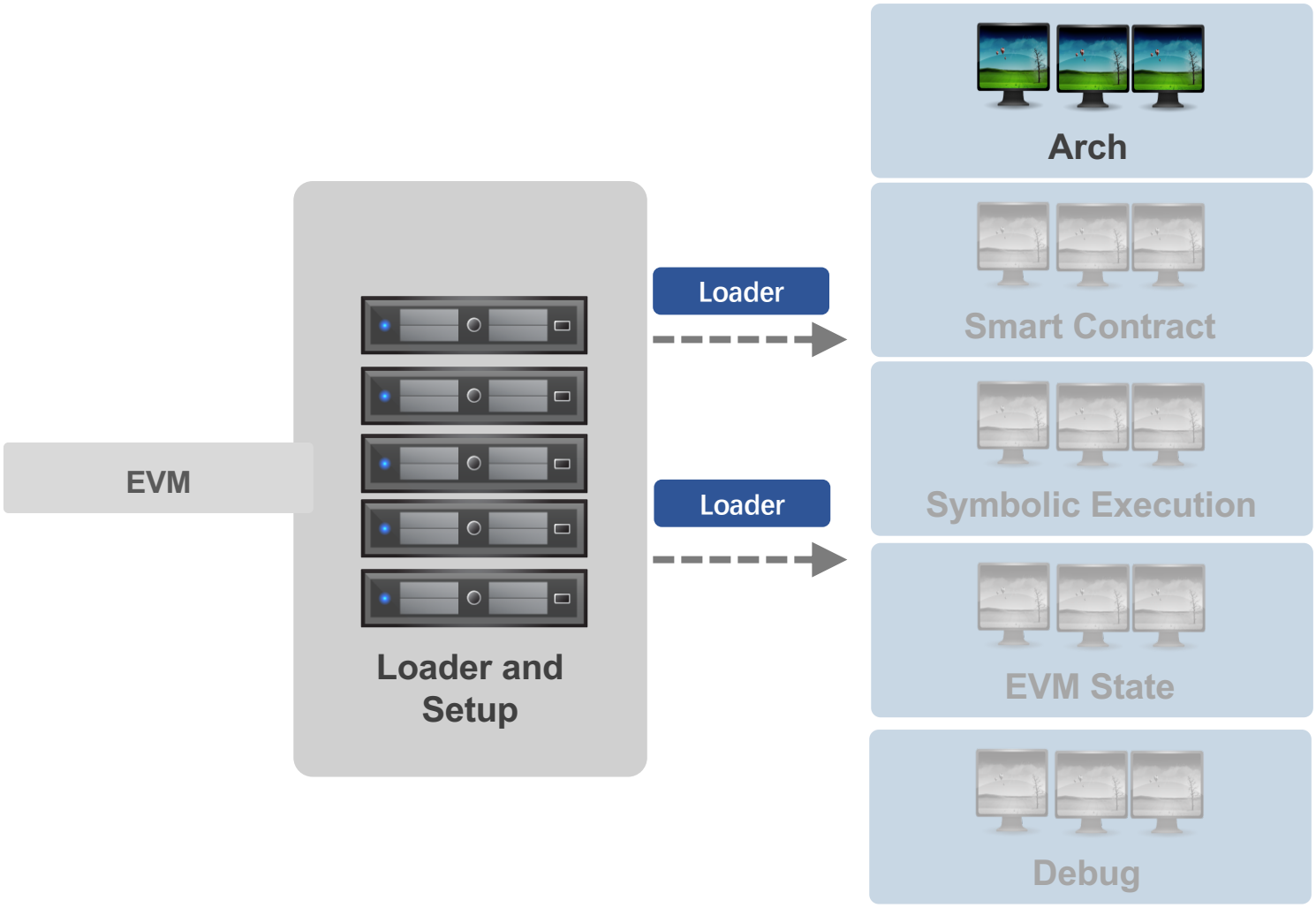
Qiling and APIs

Qiling Framework and Its Interface



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

Architecture



- create_account()
- create_message()

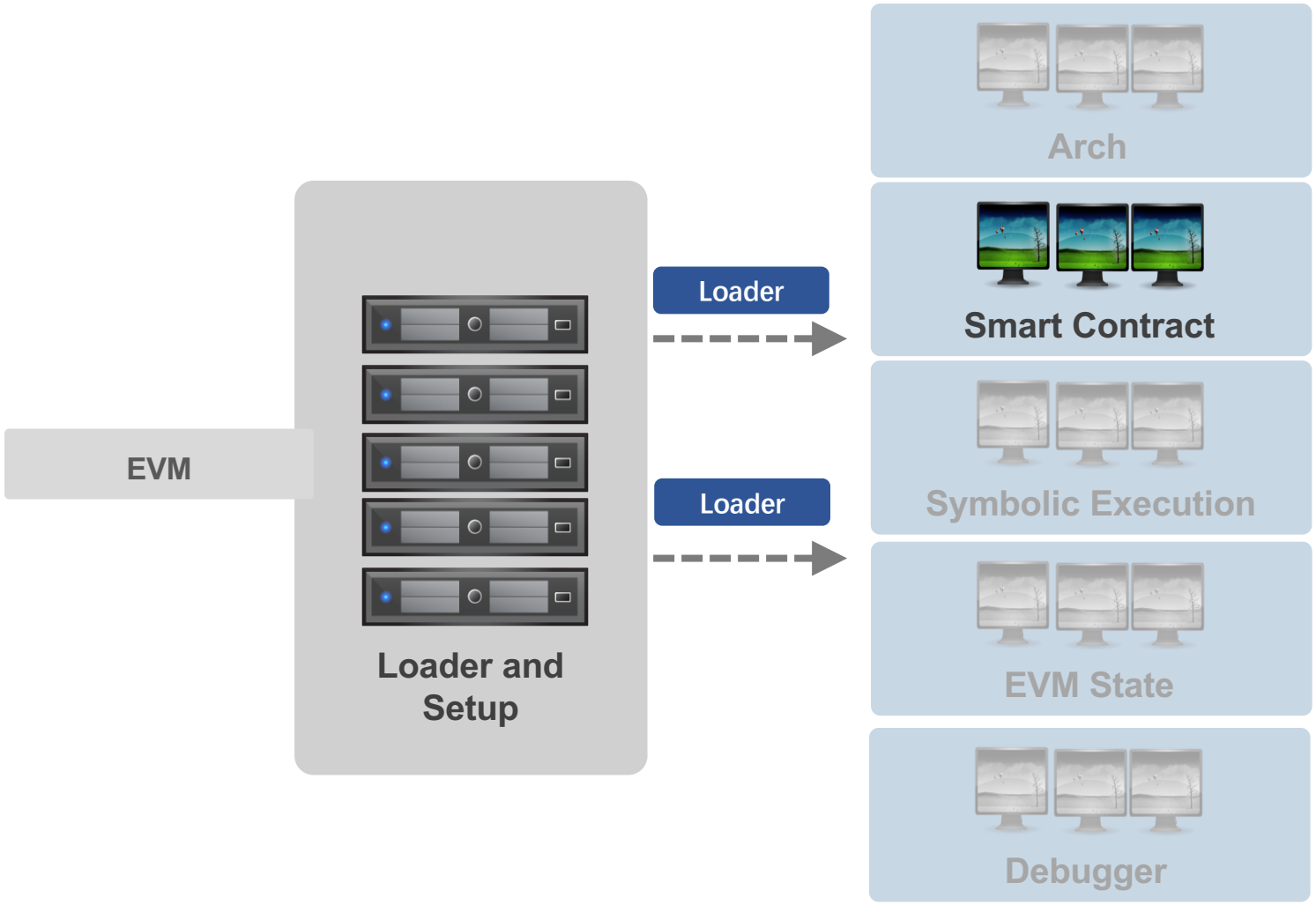
Examples: Architecture

- Initiate “Block Chain”
- Initiate “Smart Contract”

```
ql = Qiling(archtype="evm", verbose=4)
code = '0x6060604052341561000f57600080fd5b60405160208061031c8
argu = ql.arch.evm.abi.convert(['uint256'], [20])
code = code + argu

user1 = ql.arch.evm.create_account(balance=100*10**18)
user2 = ql.arch.evm.create_account(balance=100*10**18)
c1 = ql.arch.evm.create_account()
```

Initiate EVM State and Account DB

Operating System Instrumentation



- hook
  - hook_code()
  - hook_insn()
  - hook_address()
  - hook_del()
- analysis
  - analysis_func_sign()
  - disasm()

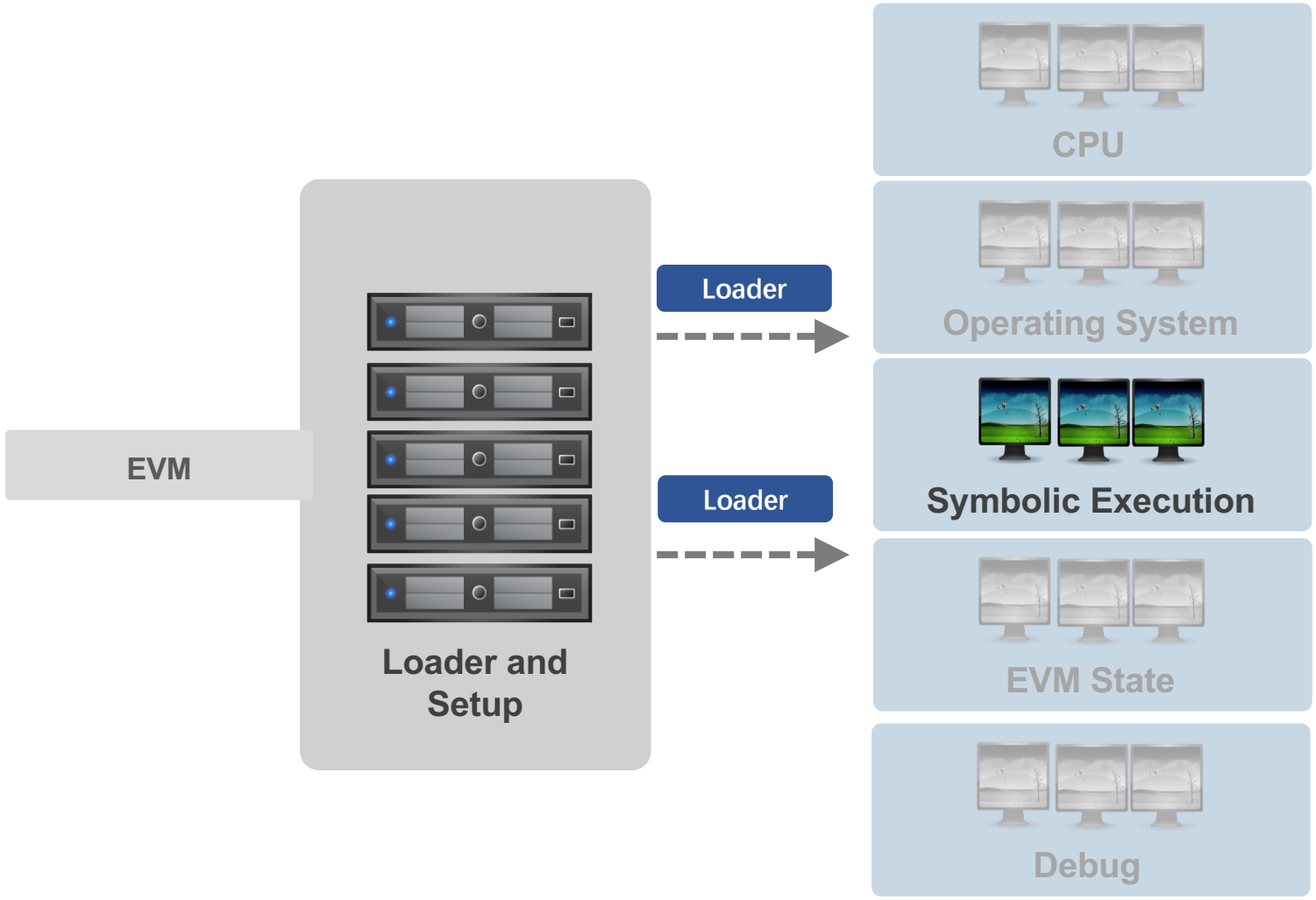
Examples: Smart Contract Hooking

```python
def hookcode_test(ql, *argv):
    print('\033[41;36m hook code success\033[0m ')

def hookinsn_test(ql, *argv):
    print('\033[41;34m hook insn success\033[0m ')

def h_addr(ql):
    print('success!')

h0 = ql.hook_code(hookcode_test)
h1 = ql.hook_address(h_addr, 9)

# message1: deploy runtime code
msg0 = ql.arch.evm.create_message(user1, b'', code=code, contract_address=c1)
ql.run(code=msg0)

ql.hook_del(h0)
ql.hook_del(h1)
h2 = ql.hook_insn(hookinsn_test, 'PUSH4')
```

- Hook everything
  - ql.hook_code()
- Hook specific instruction
  - ql.hook_insn()
- Hook specific address
  - ql.hook_address()

Provide a variety of Instrumentation

Symbolic Execution



- create_analyzer()
- symbolic_cfg()

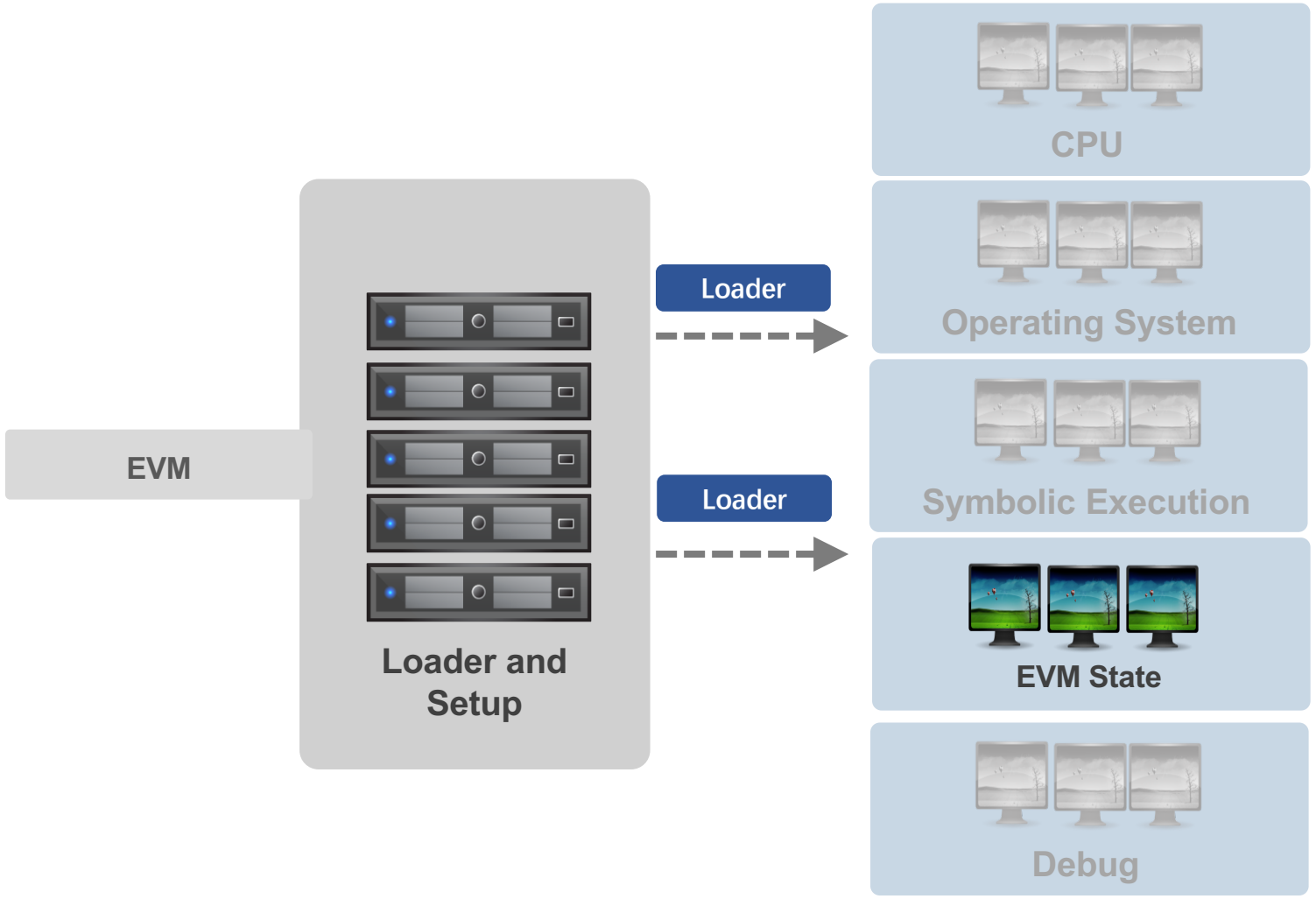
Examples: Symbolic Execution

```
contract = '6060604052341561000f57600080fd5b60405160208061031c83398
bal = ql.arch.evm.abi.convert(['uint256'], [20000000])
contract = contract + bal

svm = EthSVM()
svm.disassembler.load_from_bytecode(code=contract)
svm.create_analyzer(svm.disassembler)

sym_cfg = svm.symbolic_cfg()
```

- create_analyzer()
- symbolic_cfg()

Provides symbol execution analysis capabilities

EVM State



- Get
  - get_storage()
  - get_balance()
  - get_code()
  - …
- Set
  - set_storage()
  - set_balance()
  - set_code()
  - …
- Account
  - touch_account()
  - delete_account()

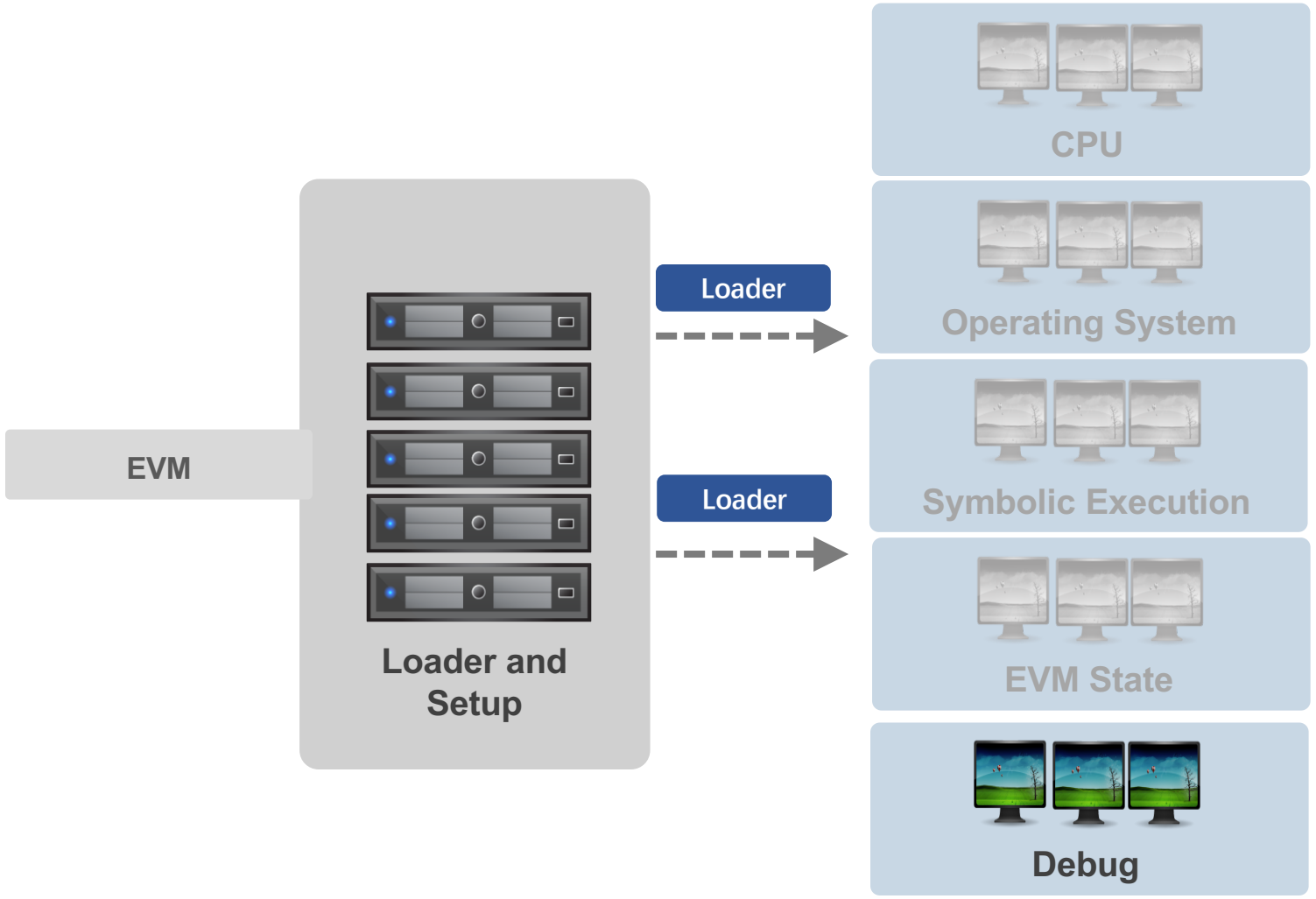
Examples: EVM State

```python
def get_storage(self, address: Address, slot: int, from_journal: bool
    return self._account_db.get_storage(address, slot, from_journal)

def set_storage(self, address: Address, slot: int, value: int) -> None
    return self._account_db.set_storage(address, slot, value)

def delete_storage(self, address: Address) -> None:
    self._account_db.delete_storage(address)

def delete_account(self, address: Address) -> None:
    self._account_db.delete_account(address)

def get_balance(self, address: Address) -> int:
    return self._account_db.get_balance(address)

def set_balance(self, address: Address, balance: int) -> None:
    self._account_db.set_balance(address, balance)

def delta_balance(self, address: Address, delta: int) -> None:
    self.set_balance(address, self.get_balance(address) + delta)
```

- Get
  - get_storage()
  - get_balance()
  - get_code()
  - …
- Set
  - set_storage()
  - set_balance()
  - set_code()
  - …
- Account
  - touch_account()
  - delete_account()

The ability to obtain EVM State in real time

Debug



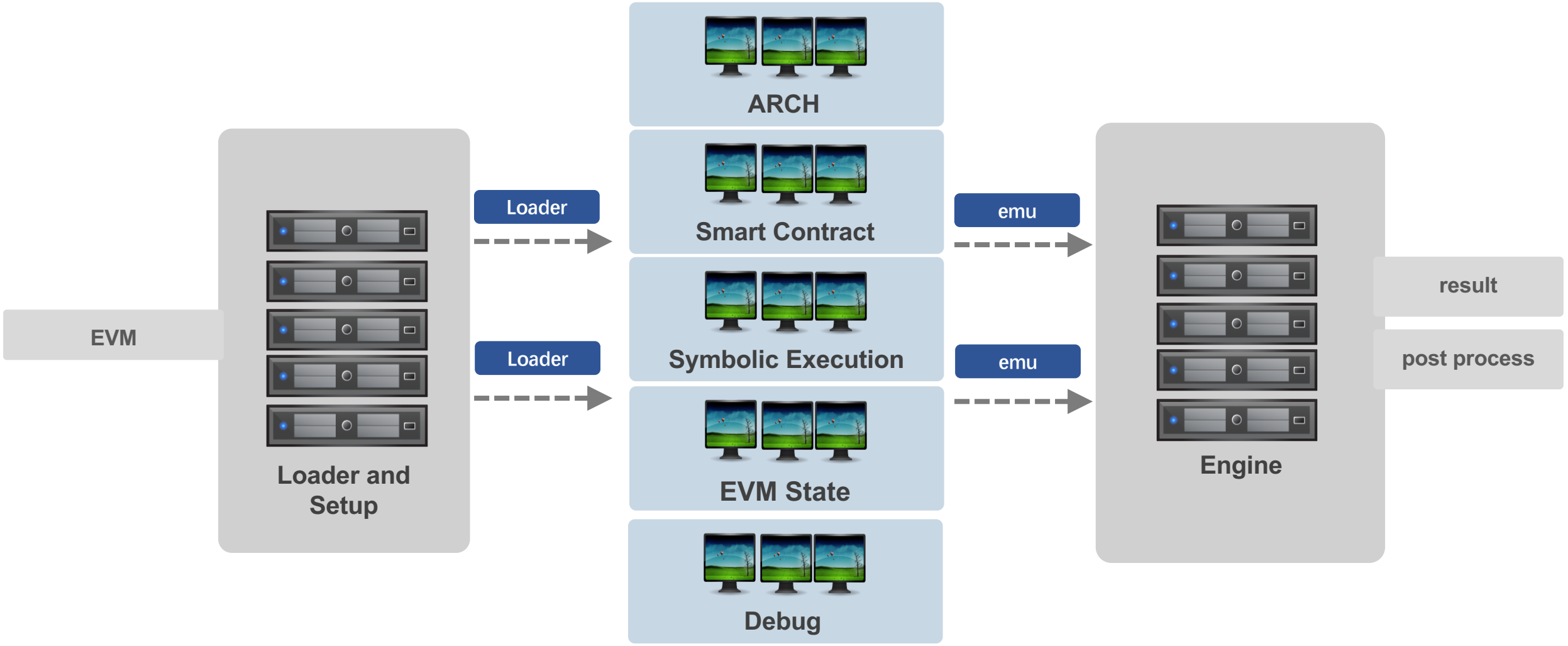
- execute_debug()

Examples: Debug

```
[evm]> help
[evm help]>
        si:  step into
        c: continue
        bp [pc]: set breakpoint
        list:
                list bp: list all breakpoint
[evm]> si
[0] 0x60 PUSH1 0x60
    stack ==>
[evm]> si
[2] 0x60 PUSH1 0x40
    stack ==>
                60
[evm]> si
[4] 0x52 MSTORE
    stack ==>
                60
                40
[evm]> bp 0x10
[evm]> list bp
[16]
[evm]>
```

- execute_debug()
  - step in
  - continue
  - let breakpoint
  - list breakpoint

Qiling Framework: In a Nutshell



Base OS can be Windows/Linux/BSD or OSX

And not limited to ARCH

Demo

Re-entry Detection

```
EtherStore_contract = '0x6080604052670de0b6b3a764000060005534801561001c57600080fd5b506103b08061002c6000396000f300608
Attack_contract     = '0x6080604052348015610010057600080fd5b506040516020806104e8339810180604052810190808051906020019

print('Init User1 balance is', vm.state.get_balance(User1))
print('Init User1 balance is', vm.state.get_balance(User2))


code1 = bytecode_to_bytes(EtherStore_contract)
print('-----------------------------------')
# 1. deploy EtherStore
msg1 = vm.build_message(None, 1, 3000000, CREATE_CONTRACT_ADDRESS, User1, 0, b'', code1, contract_address=C1)
res = vm.execute_message(msg1)

res_code = bytecode_to_bytes(res.output)
runtime_code, aux_data, constructor_args = runtime_code_detector(res_code)
rt_code = bytecode_to_bytes(runtime_code)
print('User1 balance: ', vm.state.get_balance(User1))

print('-----------------------------------')
# 2. User1 depositFunds 20ETH to bank
call_data = '0xe2c41dbc'
msg2 = vm.build_message(None, 1, 3000000, C1, User1, 20*10**18, bytecode_to_bytes(call_data), rt_code)
res = vm.execute_message(msg2)
# print(res.output)
print('User1 balance: ', vm.state.get_balance(User1))

code2 = bytecode_to_bytes(Attack_contract+ql.arch.evm.abi.convert(['address'], [C1]))
# print(code2.hex())
print('-----------------------------------')
# 3. deploy Attack
msg3 = vm.build_message(None, 1, 3000000, CREATE_CONTRACT_ADDRESS, User2, 0, b'', code2, contract_address=C2)
res = vm.execute_message(msg3)

res_code = bytecode_to_bytes(res.output)
runtime_code, aux_data, constructor_args = runtime_code_detector(res_code)
rt_code1 = bytecode_to_bytes(runtime_code)

print('-----------------------------------')
# 4. User2 pwnEtherStore with 1ETH
call_data = '0x6289d385'
msg4 = vm.build_message(None, 1, 3000000, C2, User2, 1*10**18, bytecode_to_bytes(call_data), rt_code1)
res = vm.execute_message(msg4)
# print(res.output)
print('User2 balance: ', vm.state.get_balance(User2))

print('-----------------------------------')
# 5. User2 collectEther
call_data = '0xff11e1db'
msg5 = vm.build_message(None, 1, 3000000, C2, User2, 0, bytecode_to_bytes(call_data), rt_code1)
res = vm.execute_message(msg5)
# print(res.output)
print('User2 balance: ', vm.state.get_balance(User2))
```

```
Init User1 balance is 10000000000000000000
Init User1 balance is 10000000000000000000
-----------------------------------
User1 balance:  9999999999999769962
-----------------------------------
User1 balance:  7999999999999727572
-----------------------------------
-----------------------------------
User2 balance:  9899999999999346456
-----------------------------------
User2 balance:  11899999999999315191
```

Underflow Detection

```python
ql = Qiling(archtype="evm", verbose=4)
code = '0x6060604052341561000f57600080fd5b6040516020806...
argu = ql.arch.evm.abi.convert(['uint256'], [20])
code = code + argu

user1 = ql.arch.evm.create_account(balance=100*10**18)
user2 = ql.arch.evm.create_account(balance=100*10**18)
c1 = ql.arch.evm.create_account()

def hookcode_test(ql, *argv):
    print('\033[41;36m hook code success\033[0m ')

def hookinsn_test(ql, *argv):
    print('\033[41;34m hook insn success\033[0m ')

def h_addr(ql):
    print('success!')

h0 = ql.hook_code(hookcode_test)
h1 = ql.hook_address(h_addr, 9)

# message1: deploy runtime code
msg0 = ql.arch.evm.create_message(user1, b'', code=code, contract_address=c1)
ql.run(code=msg0)

ql.hook_del(h0)
ql.hook_del(h1)
h2 = ql.hook_insn(hookinsn_test, 'PUSH4')

#  # SMART CONTRACT DEPENDENT - message2: check balance of user1, should be 20
def check_balance(sender, destination):
    call_data = '0x70a08231'+ql.arch.evm.abi.convert(['address'], [sender])
    msg2 = ql.arch.evm.create_message(sender, destination, call_data)
    return ql.run(code=msg2)

result = check_balance(user1, c1)
print('\n\nuser1 balance =', int(result.output.hex()[2:], 16))
ql.hook_del(h2)

# SMART CONTRACT DEPENDENT - message3: transform 21 from user1 to user2
call_data = '0xa9059cbb'+ ql.arch.evm.abi.convert(['address'], [user2]) + \
                          ql.arch.evm.abi.convert(['uint256'], [21])
msg1 = ql.arch.evm.create_message(user1, c1, call_data)
result = ql.run(code=msg1)
print('\n\nis success =', int(result.output.hex()[2:], 16))

# message4: check balance of user1, should be MAX - 1
result = check_balance(user1, c1)
print('\n\nuser1 balance =', hex(int(result.output.hex()[2:], 16)))
self.assertEqual(hex(int(result.output.hex()[2:], 16)), '0xfffffffffffffffffffffffffffffffffffffff
```

```
[+]     [205] 0x80 DUP1
[+]         stack ==>
                0x70a08231
                0x80
                0000000000000000000000000000000000000000000000000000000000000060
[+]     [206] 0x91 SWAP2
[+]         stack ==>
                0x70a08231
                0x80
                0000000000000000000000000000000000000000000000000000000000000060
                0000000000000000000000000000000000000000000000000000000000000060
[+]     [207] 0x3 SUB
[+]         stack ==>
                0x70a08231
                0000000000000000000000000000000000000000000000000000000000000060
                0000000000000000000000000000000000000000000000000000000000000060
                0x80
[+]     [208] 0x90 SWAP1
[+]         stack ==>
                0x70a08231
                0000000000000000000000000000000000000000000000000000000000000060
                0x20
[+]     [209] 0xf3 RETURN
[+]         stack ==>
                0x70a08231
                0x20
                0000000000000000000000000000000000000000000000000000000000000060


user1 balance = 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Fuzzer

- AFL++
- With Qiling EVM engine

```
ount(balance=20)
 = evm.create_contract(user1, bytecode=code, name='c1')


rt_to_hex(['address'], [user1.address])
+ addr1
to_bytes(input1)
saction(user1, 0, contract, call_data1)
balance = ', int.from_bytes(output, byteorder='big'))

nce=21)
ount(balance=0)
rt_to_hex(['address'], [user2.address])
BI.convert_to_hex(['uint256'], [fuzz_balance])
 addr2 + trans_balance
to_bytes(input2)
saction(user1, 0, contract, call_data2)
esult: ', bool(int.from_bytes(output, byteorder='big')))


+ addr1
to_bytes(input3)
saction(user1, 0, contract, call_data1)
ut, byteorder='big') > 20:

balance = ', int.from_bytes(output, byteorder='big'))
```

```
american fuzzy lop 2.52b (python)

┌─ process timing ─────────────────────┬─ overall results ──────┐
│        run time : 0 days, 0 hrs, 0 min, 41 sec │  cycles done : 159 │
│   last new path : none yet (odd, check syntax!) │  total paths : 1   │
│ last uniq crash : 0 days, 0 hrs, 0 min, 18 sec │ uniq crashes : 5   │
│  last uniq hang : none seen yet      │   uniq hangs : 0   │
├─ cycle progress ─────┬─ map coverage ─┴────────────────────┘
│  now processing : 0 (0.00%)  │   map density : 0.02% / 0.02%
│ paths timed out : 0 (0.00%)  │ count coverage : 1.00 bits/tuple
├─ stage progress ─────┼─ findings in depth ─────────────────┐
│  now trying : havoc  │ favored paths : 1 (100.00%)         │
│ stage execs : 57/256 (22.27%) │  new edges on : 1 (100.00%) │
│ total execs : 39.7k  │ total crashes : 1658 (5 unique)     │
│  exec speed : 996.4/sec │  total tmouts : 0 (0 unique)     │
├─ fuzzing strategy yields ────────────┴─ path geometry ──────┤
│   bit flips : 0/16, 0/15, 1/13       │    levels : 1       │
│  byte flips : 0/2, 0/1, 0/0          │   pending : 0       │
│ arithmetics : 0/112, 0/25, 0/0       │  pend fav : 0       │
│  known ints : 0/9, 0/28, 0/0         │ own finds : 0       │
│  dictionary : 0/0, 0/0, 0/0          │  imported : n/a     │
│       havoc : 4/39.4k, 0/0           │ stability : 100.00% │
│        trim : n/a, 0.00%             ├─────────────────────┘
└──────────────────────────────────────┘
                                                [cpu000: 15%]
```

Debugger

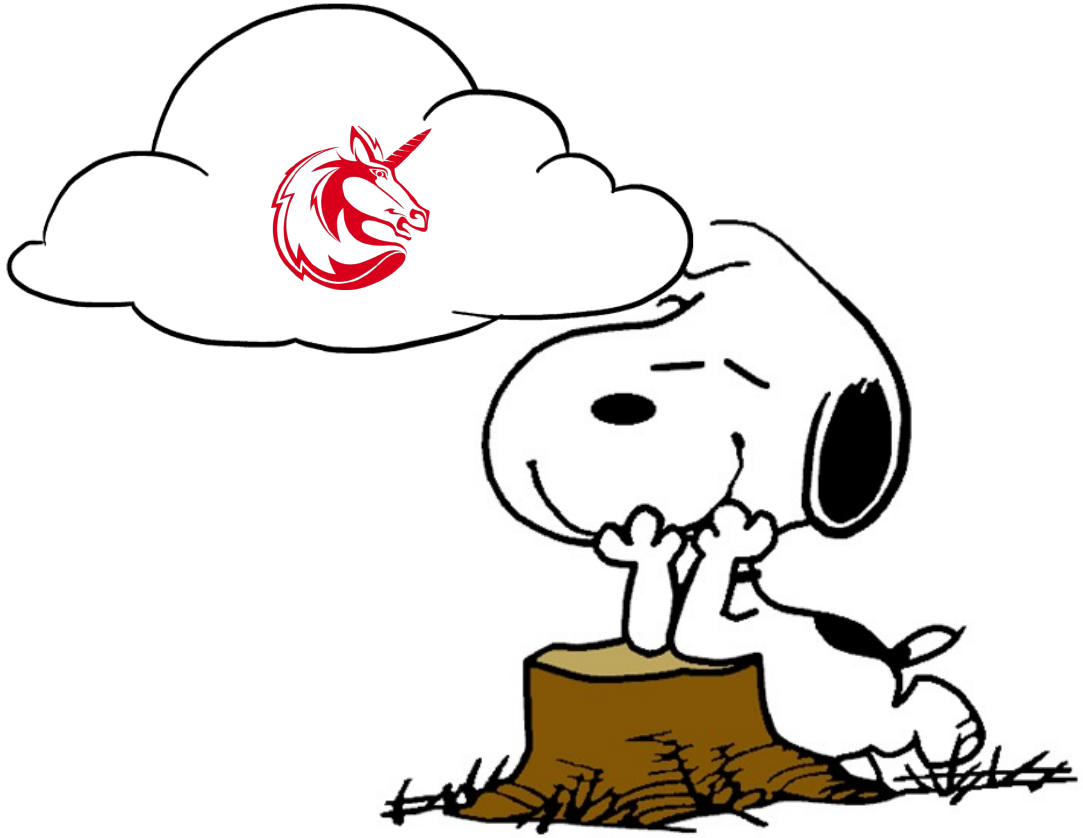
- Real time execution debugger

```
# kabeor @ DESKTOP-6PC3B3J in ~/Dev/qiling_engine/examples/evm on git:dev x
$ python evm_reentrancy.py
[+]     Profile: Default
Init User1 balance is 10000000000000000000
Init User1 balance is 10000000000000000000
----------------------------------
[evm]> si
[+]     [0] 0x60 PUSH1 0x80
[+]         stack ==>
[evm]> si
[+]     [2] 0x60 PUSH1 0x40
[+]         stack ==>
                80
[evm]> si
[+]     [4] 0x52 MSTORE
[+]         stack ==>
                80
                40
[evm]> si
[+]     [5] 0x67 PUSH8 0x0de0b6b3a7640000
[+]         stack ==>
[evm]> si
[+]     [14] 0x60 PUSH1 0x00
[+]         stack ==>
                0de0b6b3a7640000
[evm]> c
[+]     [16] 0x55 SSTORE
[+]         stack ==>
                0de0b6b3a7640000
                00
[+]     [17] 0x34 CALLVALUE
[+]         stack ==>
[+]     [18] 0x80 DUP1
[+]         stack ==>
                0x0
[+]     [19] 0x15 ISZERO
[+]         stack ==>
                0x0
```

Next Step

Roadmap

- Force Unicorn Engine sync with QEMU 5, Code name **Unicorn 2**
  - More architectures, more CPU instructions set
  - Almost Done
  - **Looking for Release *Sponsor***
- Android Java bytecode layer instrumentation
- **Forward to host implementation**
- iPhoneOS/MacOS/M1 emulation support
- More robust Windows emulation
  - Introduce wine && Cygwin or something
- **ETA: Smart Contract emulation (EVM, soon WASM)**
- MCU emulation

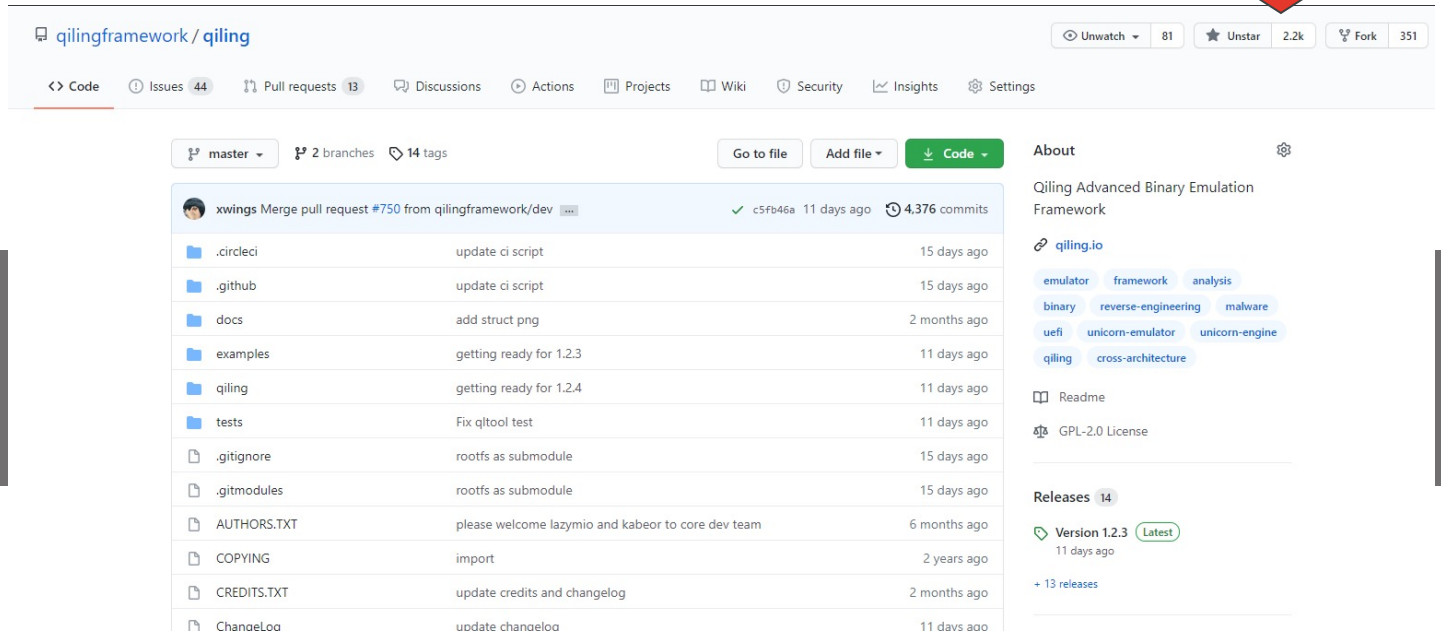


Join Us and Make Pull Request !!!

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