

Qiling Framework: Introduction

November 2020



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

~ \$ 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.

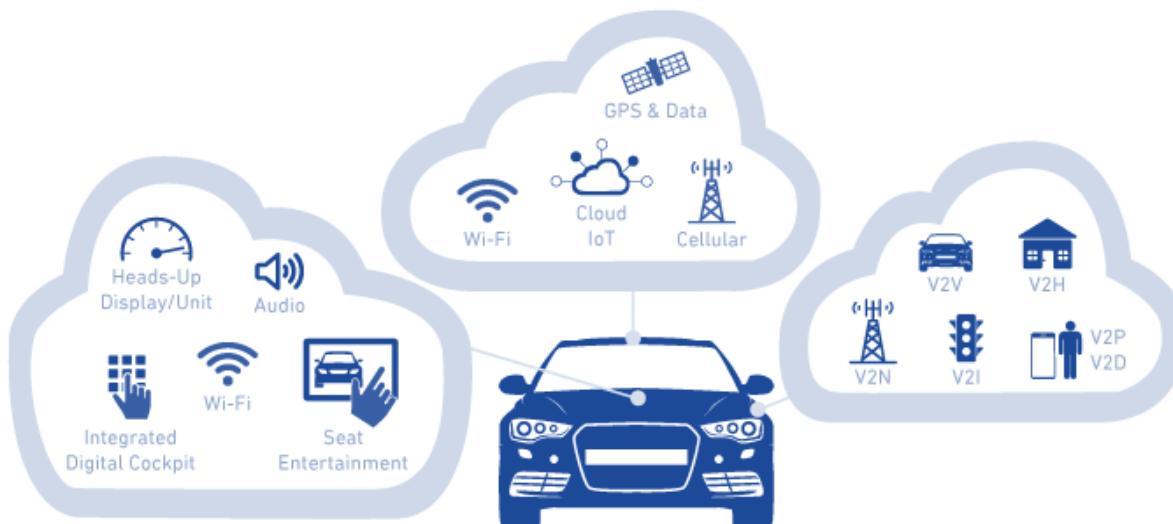
Core developer of Qiling.

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 IoT Reverse Engineering Great

Today's IoT Analysis



To under a firmware

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

How We Fix It

GPS

Wi-Fi

Bluetooth

Audio

Screen



And more on our smartphone app. (Now available on Qi teamaker Vita+)



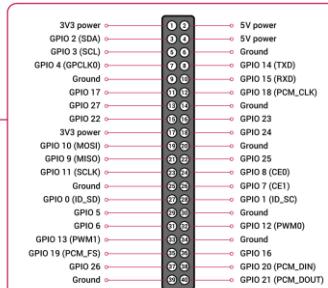
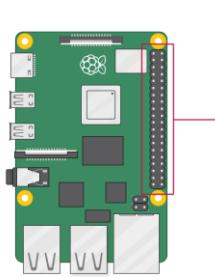
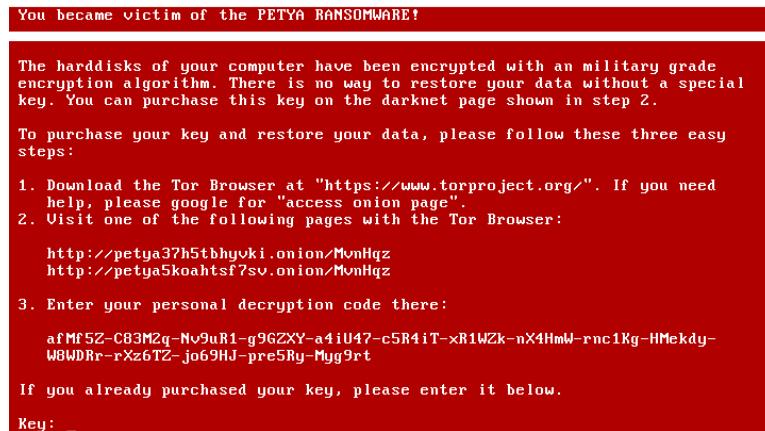
No Actual Hardware Needed



Bring the entire car firmware's binary into emulation
with virtual devices support

Wait, There are Virtual Machines

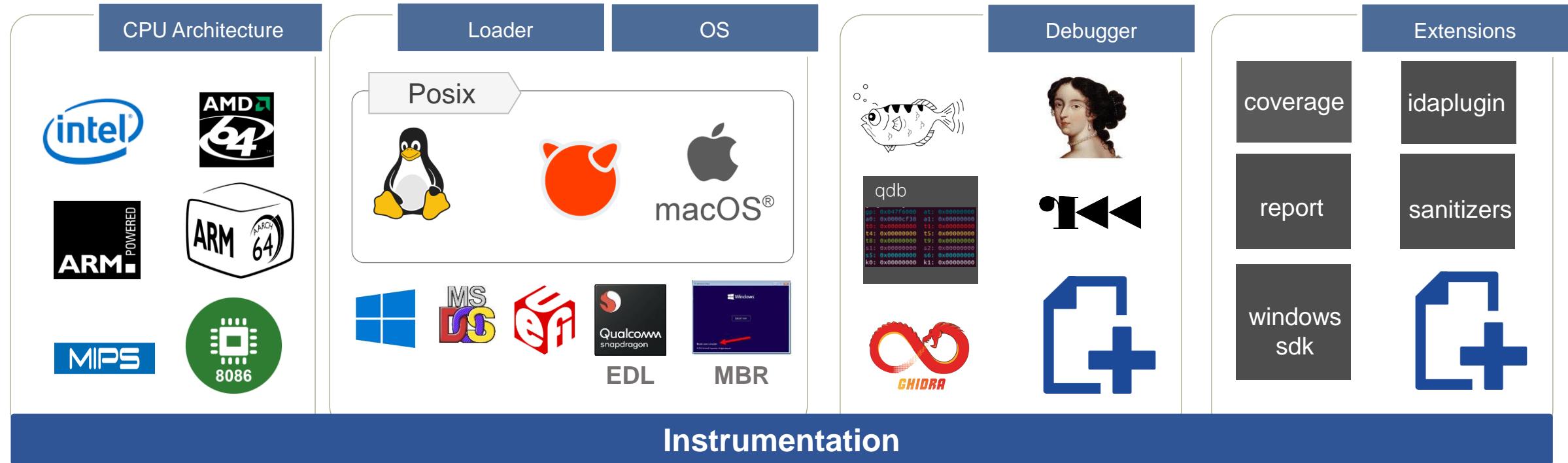
Current Virtual Machine Limitation



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

Qiling Framework

Overview



External Hardware Emulation

Qiling Framework

Features

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

	8086	x86	x86-64	ARM	ARM64	MIPS
Windows (PE)	-	<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"/>	-	-	-	-	-

Similarity

User Mode Emulation



qemu-usermode

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



usercorn

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



Binee

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



WINE

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



Speakeasy

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



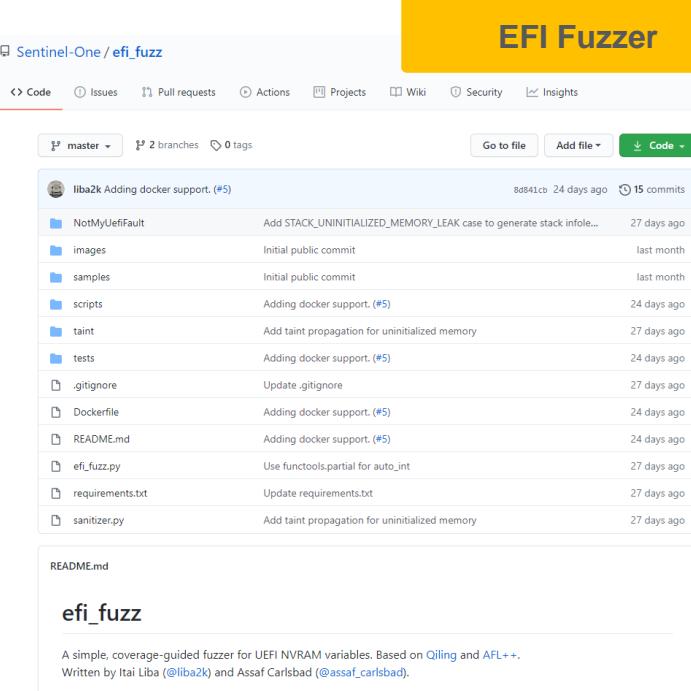
Zelos

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

Framework

Framework, NOT Tools

EFI Fuzzer



sentinel-one/efi_fuzz

Code Issues Pull requests Actions Projects Wiki Security Insights

master 2 branches 0 tags

Go to file Add file Code

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

NotMyUefiFault Add STACK_UNINITIALIZED_MEMORY_LEAK case to generate stack infole... 27 days ago

images Initial public commit last month

samples Initial public commit last month

scripts Adding docker support. (#5) 24 days ago

taint Add taint propagation for uninitialized memory 27 days ago

tests Adding docker support. (#5) 24 days ago

.gitignore Update .gitignore 27 days ago

Dockerfile Adding docker support. (#5) 24 days ago

README.md Adding docker support. (#5) 24 days ago

efi_fuzz.py Use functools.partial for auto_int 27 days ago

requirements.txt Update requirements.txt 27 days ago

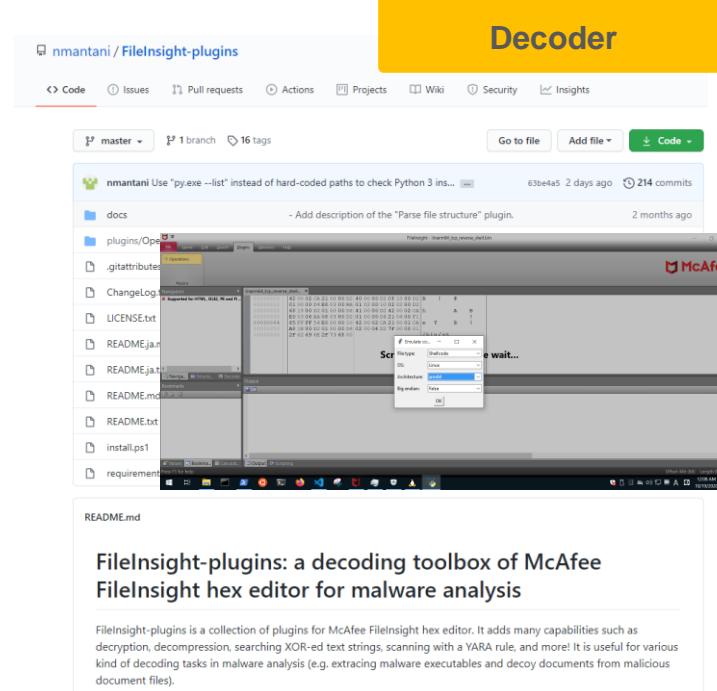
sanitizer.py Add taint propagation for uninitialized memory 27 days ago

README.md

efi_fuzz

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

Decoder



nmantani/FileInsight-plugins

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 16 tags

Go to file Add file Code

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

docs - Add description of the "Parse file structure" plugin. 2 months ago

plugins/Opener.githandler ChangeLog

githandler.githandler.py LICENSE.txt

README.ja README.ja

README.md README.md

README.txt README.txt

install.ps1 requirements

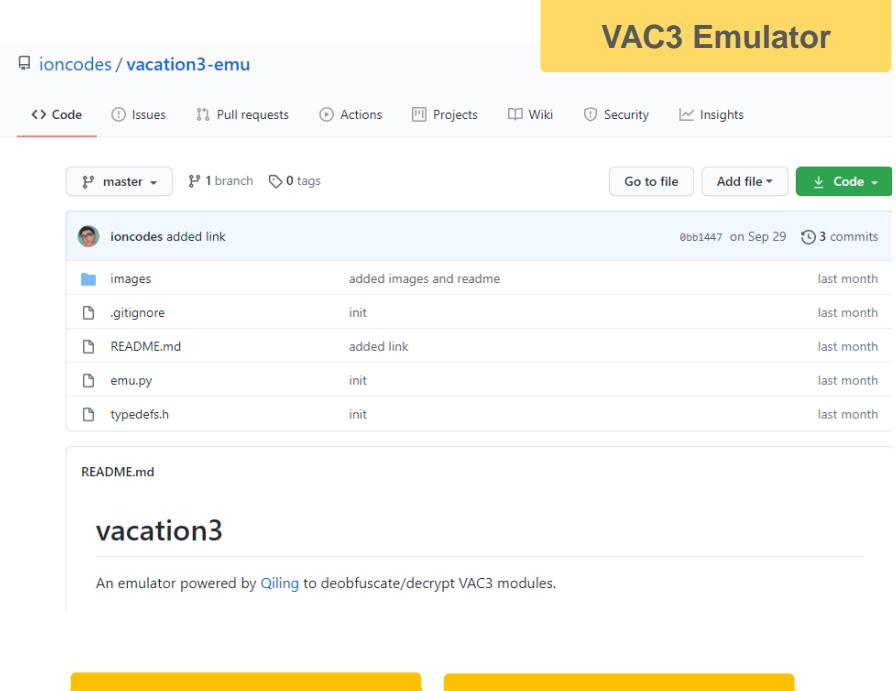
McAfee hex editor screenshot showing a file being decoded.

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. extracting 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.

Binary Fuzzer IoT Fuzzer Malware Sandbox CTF Solver

IoT Emulator MacOS Emulator

IOS Emulator Binary Decrypt

Qiling Framework



Instrumentation (Qiling's API)

Instrumentation

What Is Instrumentation

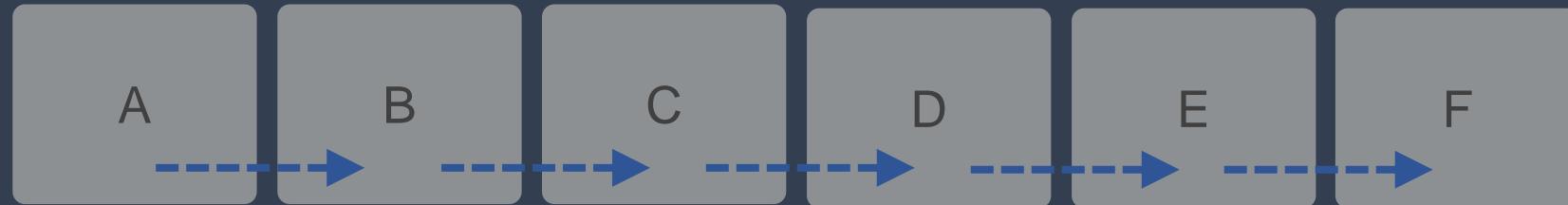
Binary Execution Flow



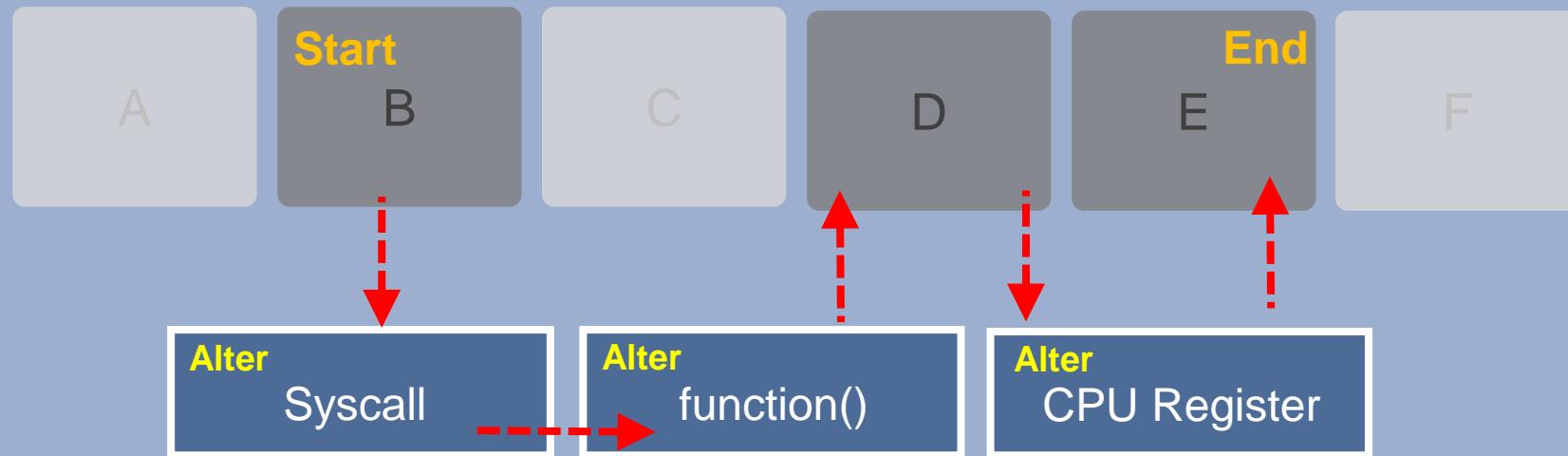
One Function After Another

What Is Instrumentation

Binary Execution Flow

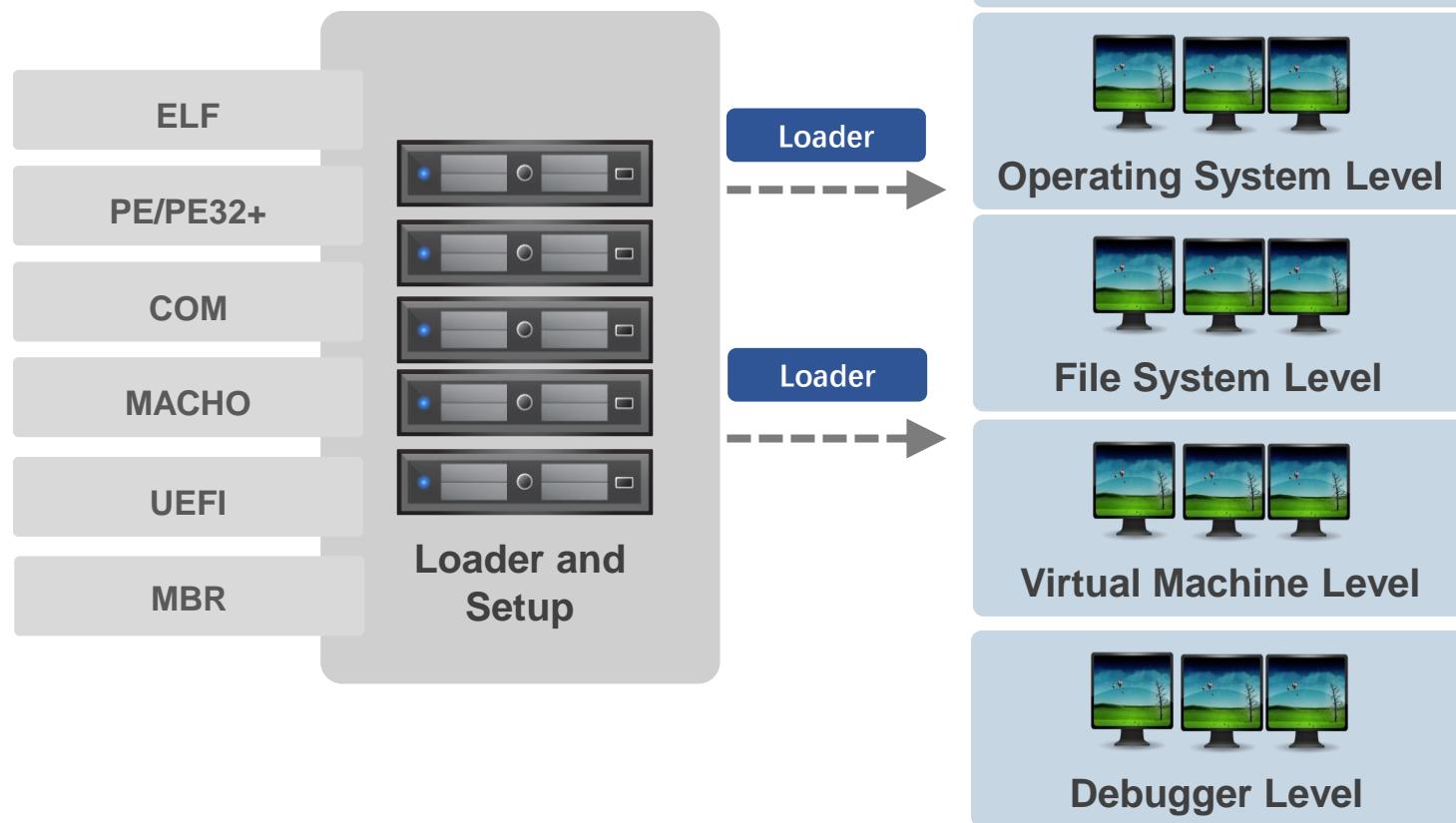


Qiling's Instrumentation



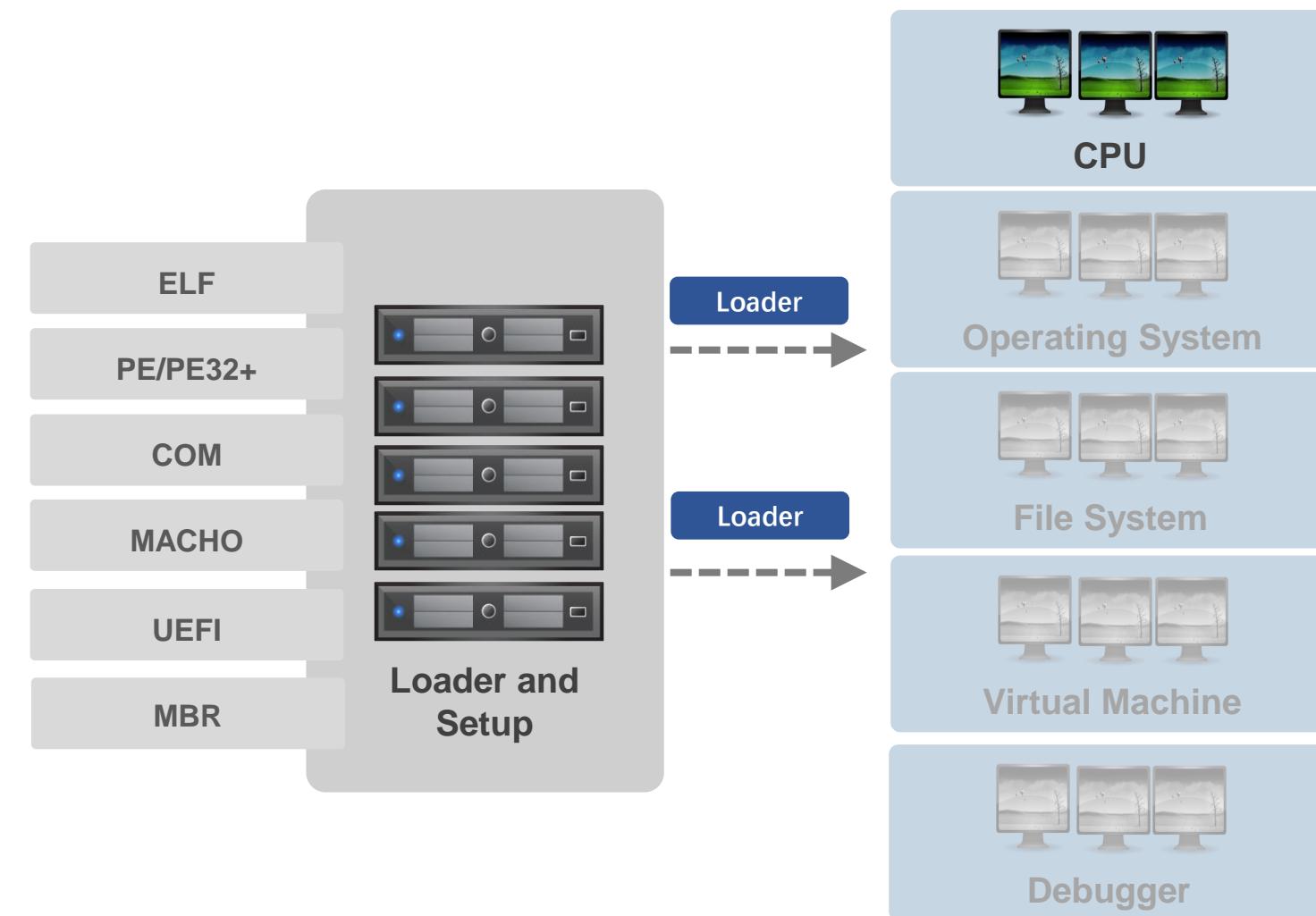
Qiling and APIs

Qiling Framework and Its Interface



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

CPU Instrumentation



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

CPU Instrumentation: Examples

```
def my_puts(ql):
    addr = ql.os.function_arg[0]
    print("puts(%s)" % ql.mem.string(addr))
    reg = ql.reg.read("rax")
    print("reg : 0x%08x" % reg)
    ql.reg.rax = reg
    self.set_api = reg

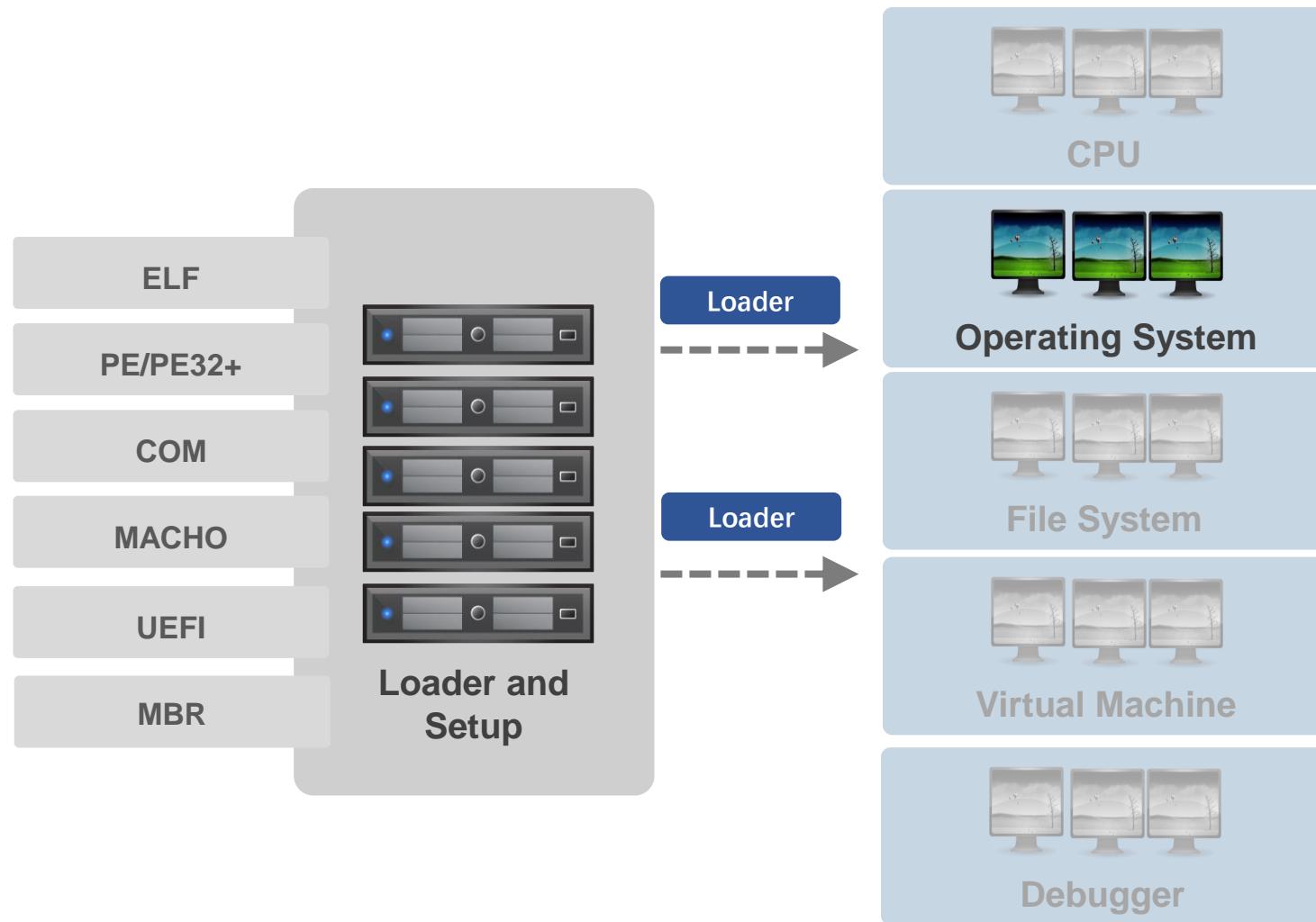
def write_onEnter(ql, arg1, arg2, arg3, *args):
    print("enter write syscall!")
    ql.reg.rsi = arg2 + 1
    ql.reg.rdx = arg3 - 1
    self.set_api_onenter = True

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

ql = Qiling(["../examples/rootfs/x86_64_linux/bin/x86_64_args", "1234test", "12345678", "bin/x86_64_hello"], "../etc/qiling.conf")
ql.set_syscall(1, write_onEnter, QL_INTERCEPT.ENTER)
ql.set_api('puts', my_puts)
ql.set_syscall(1, write_onexit, QL_INTERCEPT.EXIT)
ql.mem.map(0x1000, 0x1000)
ql.mem.write(0x1000, b"\xFF\xFE\xFD\xFC\xFB\xFA\xFB\xFC\xFC\xFE\xFD")
ql.mem.map(0x2000, 0x1000)
ql.mem.write(0x2000, b"\xFF\xFE\xFD\xFC\xFB\xFA\xFB\xFC\xFC\xFE\xFD")
ql.run()
```

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

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()`

Example: Operating System

```
from qiling import *

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

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

    ql.os.definesyscall_return(regreturn)

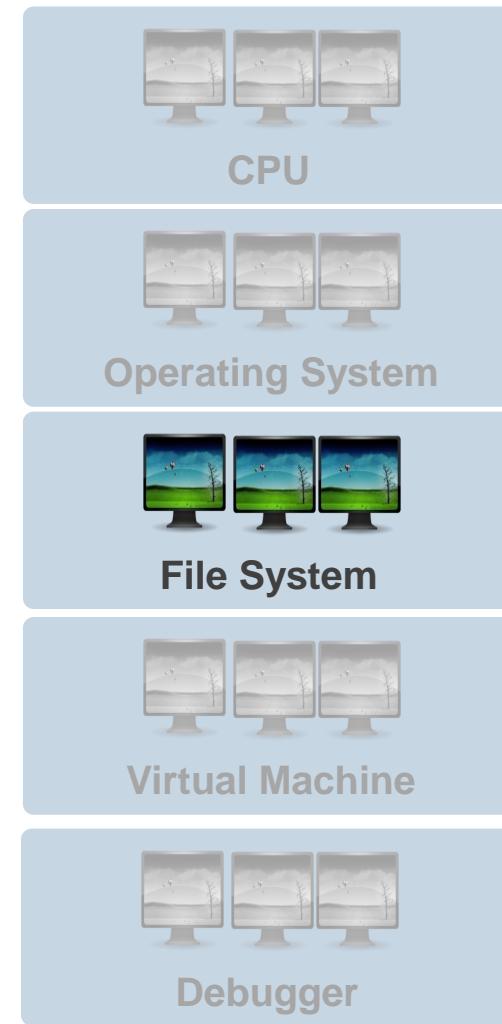
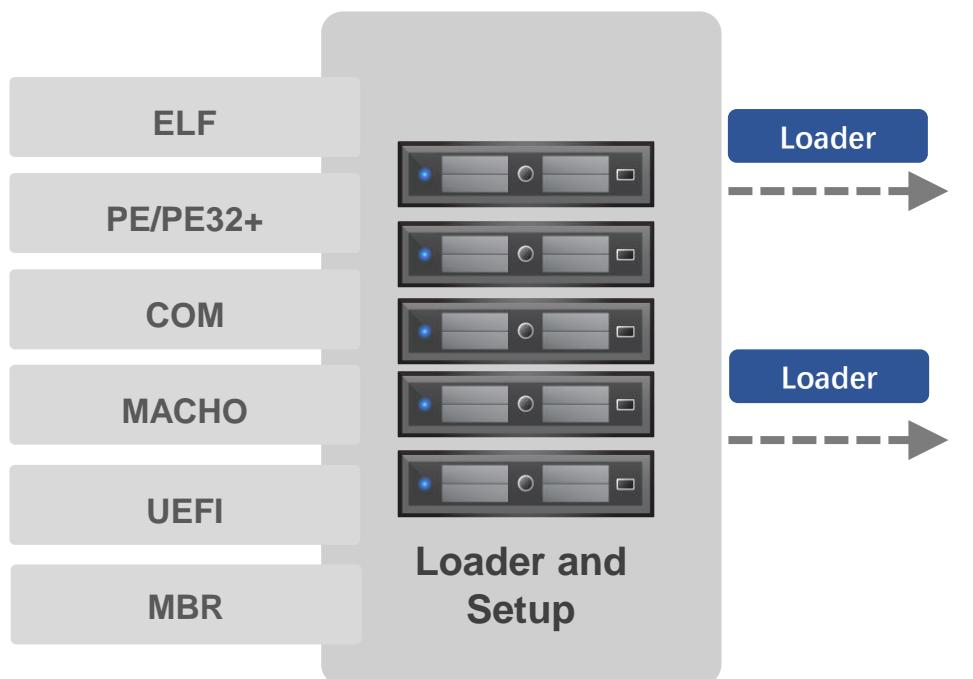
if __name__ == "__main__":
    ql = Qiling(["rootfs/arm_linux/bin/arm_hello"], "rootfs/arm_linux", output = "debug")
    # Custom syscall handler by syscall name or syscall number.
    # Known issue: If the syscall func is not be implemented in qiling, qiling does
    # not know which func should be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

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

    ql.run()
```

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

File System Instrumentation



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

File System Instrumentation

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

class Fake_urandom(QlFsMappedObject):

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

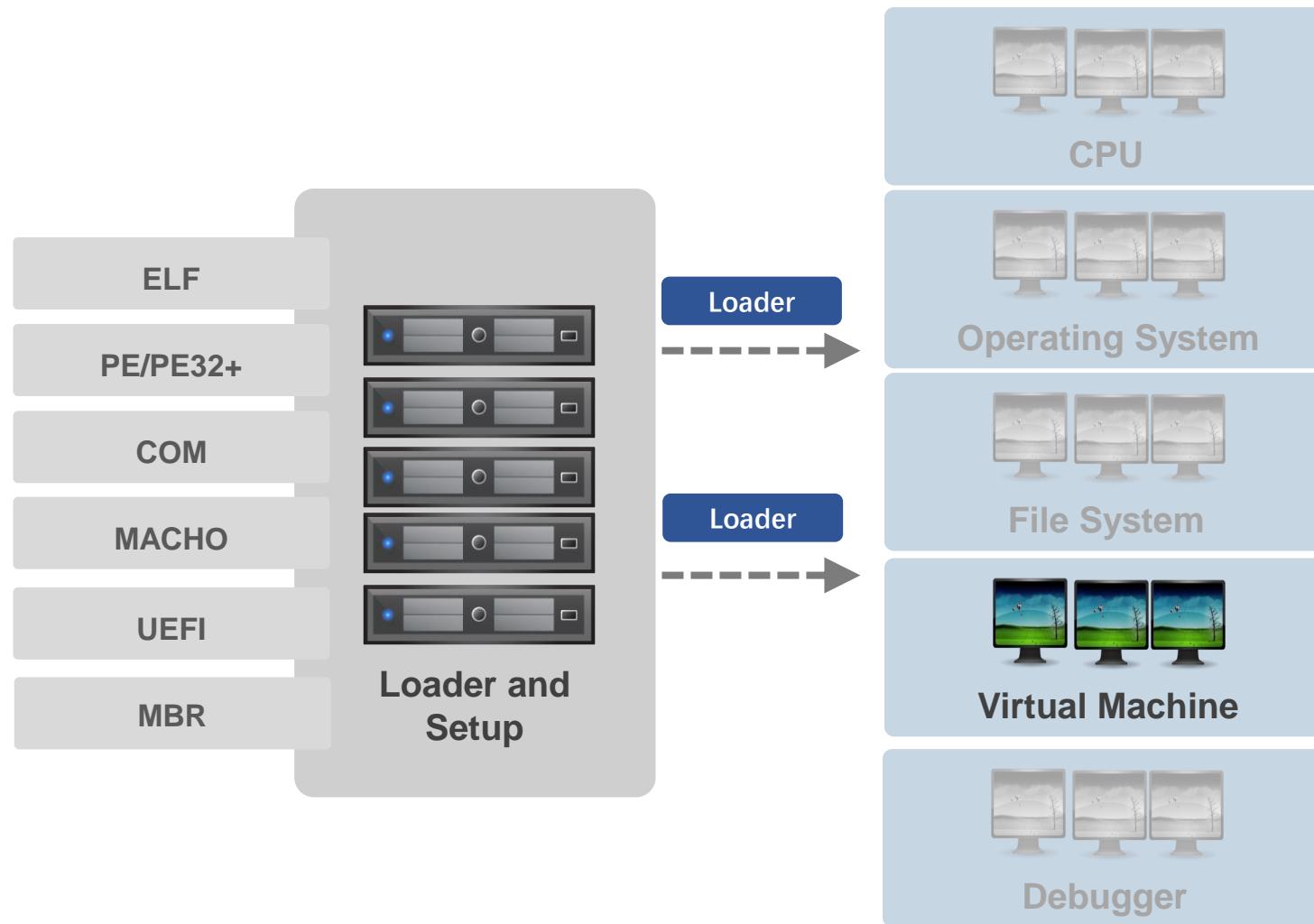
    def fstat(self): # syscall fstat will ignore it if return -1
        return -1

    def close(self):
        return 0

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

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

Virtual Machine Instrumentation



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

Virtual Machine Instrumentation

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

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

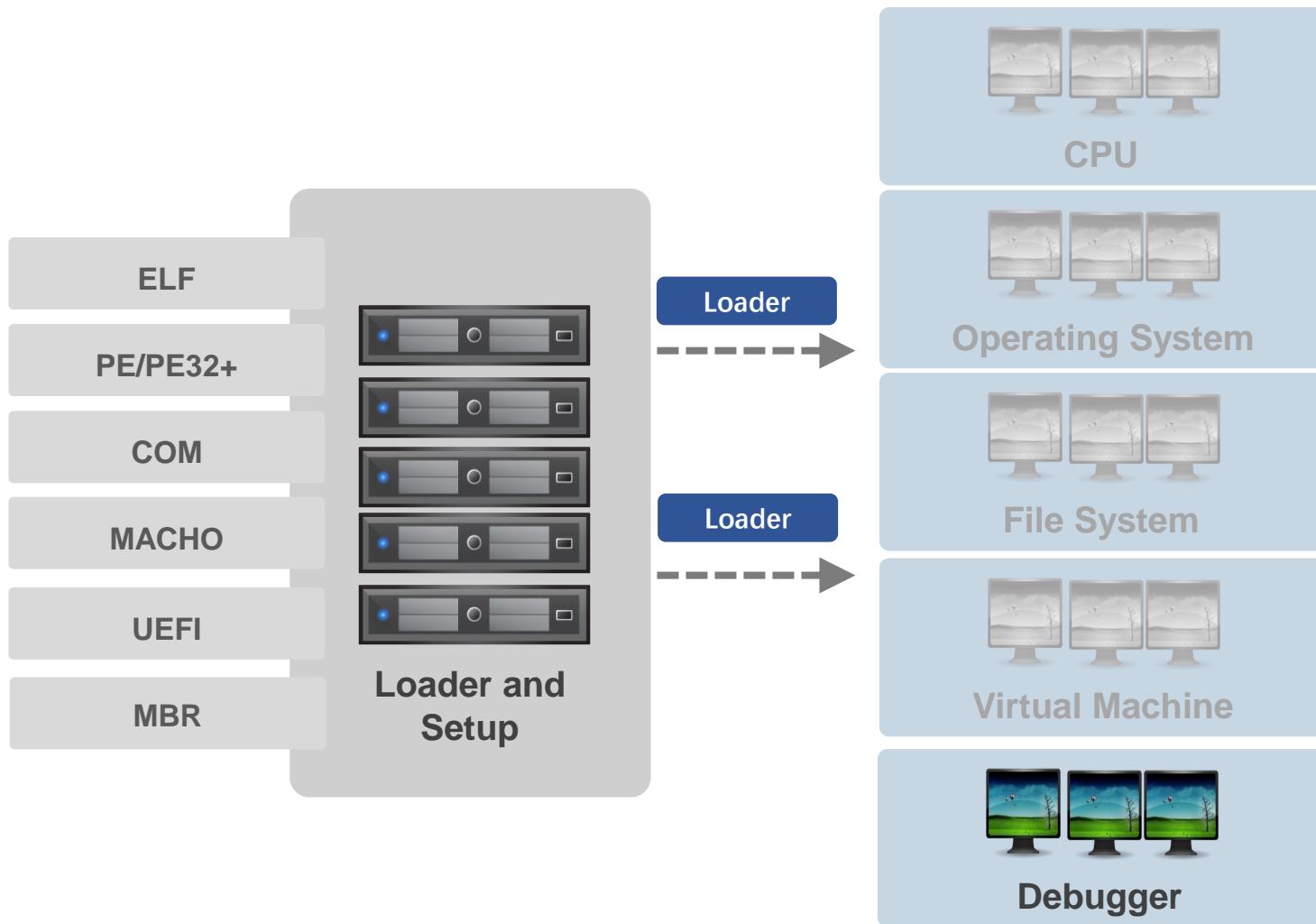
def check_pc(ql):
    print("=" * 50)
    print("[!] Hit fuzz point, stop at PC = 0x%x" % ql.reg.arch_pc)
    print("=" * 50)
    ql.emu_stop()

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

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

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

Debugger



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

Debugger

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

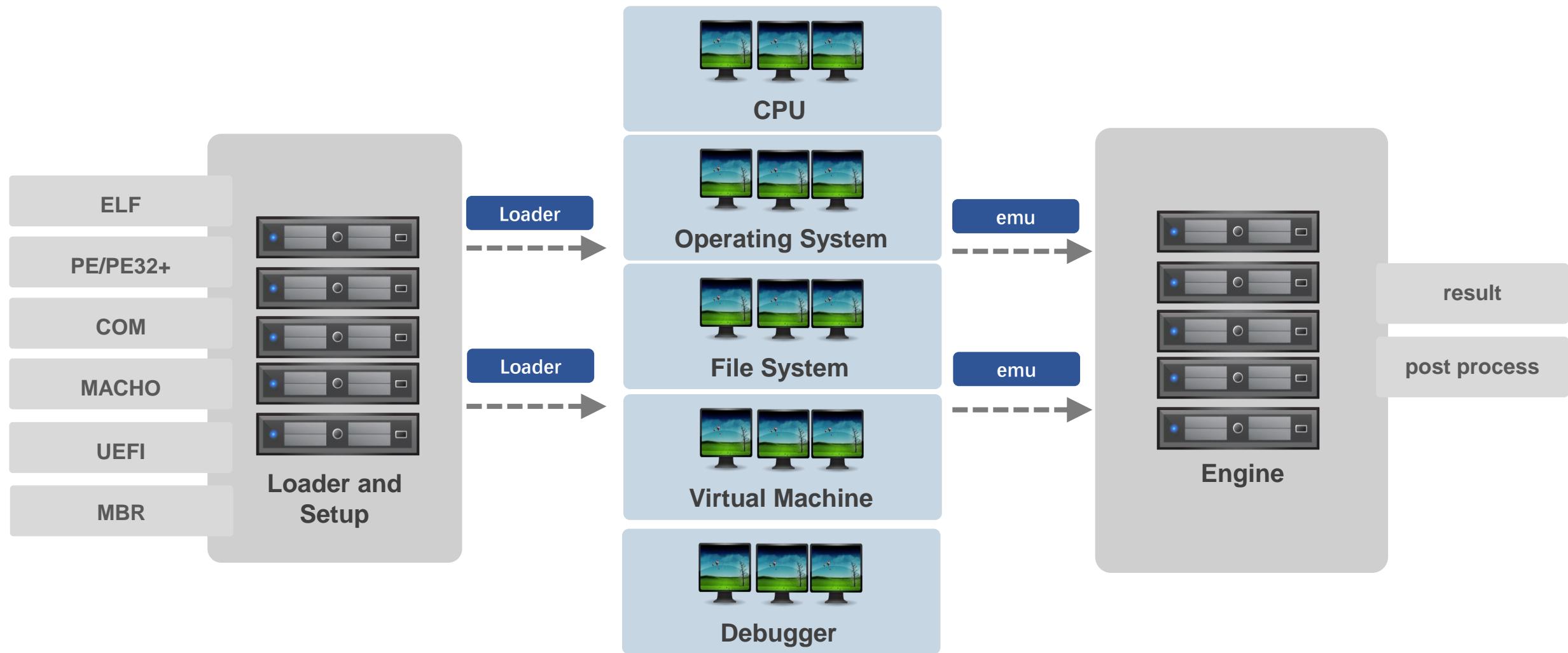
```
if __name__ == "__main__":
    run_sandbox(["rootfs/arm_linux/bin/arm_hello"], "rootfs/arm_linux", "debug")
```

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

```
from qiling import *

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

Qiling Framework: In a Nutshell



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

Preview

Demo Setup

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

➤ Required OS

- Ubuntu 18.04 / 20.04
- WSL2

➤ Install Qiling Framework

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

➤ Install AFL++

- git clone <https://github.com/AFLplusplus/AFLplusplus.git>
- cd AFLplusplus
- make
- cd unicorn_mode
- ./build_unicorn_support.sh

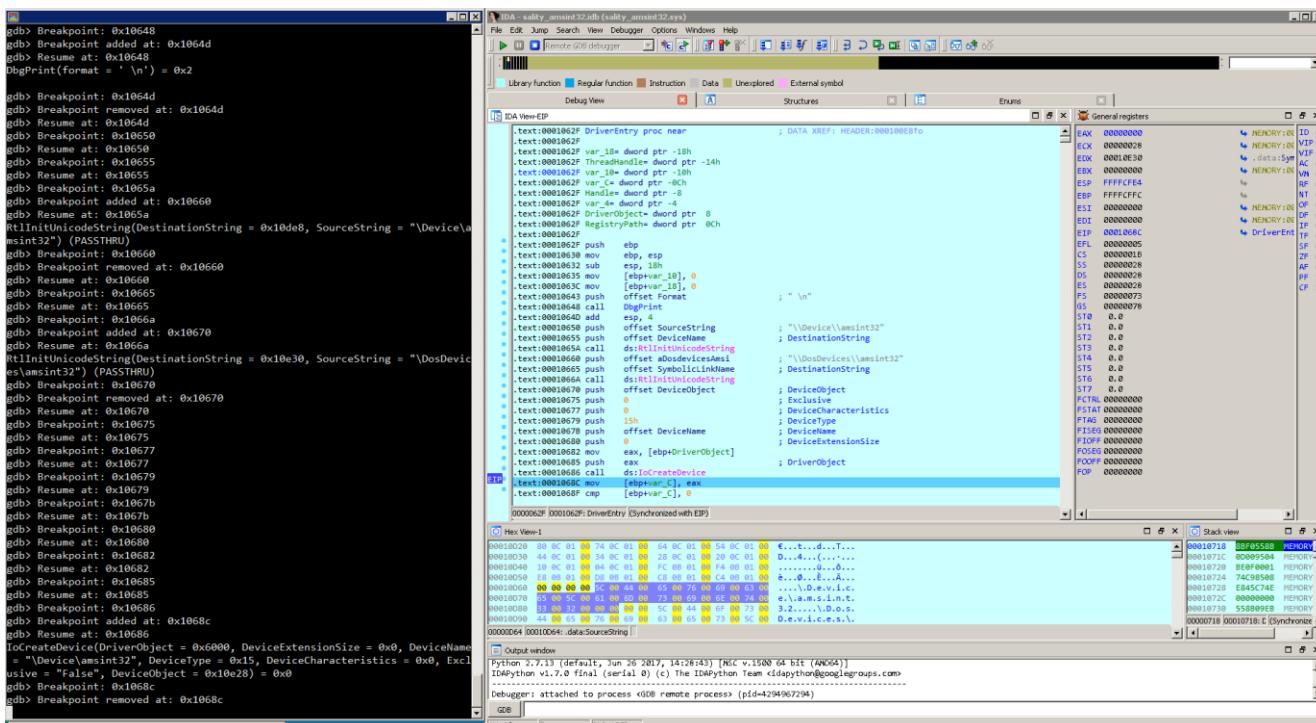
Microsoft ❤️ Linux

Malware & Rootkit Analysis

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

```
root@7fd8b304fccec:/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 UnhandledExceptionFilterTest [=] GOOD
[!] Checking OutputDebugString [=] GOOD
[!] Checking Hardware Breakpoints [=] GOOD
[!] Checking Software Breakpoints [=] GOOD
[!] Checking Interrupt 0x2d [=] BAD
We are finally done
root@7fd8b304fccec:/qilingdev#
```

```
(22:43:04):xwings@bespin:~/projects/qiling>
(15)$ python3 qltool run -f examples/rootfs/x86_windows/bin/al-khaser.bin --rootfs examples/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 jexamples/rootfs/x86_windows/bin/al-khaser.bin
GetSystemTimeAsFileTime(lpSystemTimeAsFileTime = 0xfffffcfc)
GetCurrentThreadId() = 0x0
GetCurrentProcessId() = 0x2005
QueryPerformanceCounter(lpPerformanceCount = 0xfffffcfe) = 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) = 0x1
[+] 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) = 0x1
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x425398, dwSpinCount = 0x0) = 0x1
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x4253b0, dwSpinCount = 0x0) = 0x1
InitializeCriticalSectionAndSpinCount(lpCriticalSection = 0x4253c8, dwSpinCount = 0x0) = 0x1
```



Fuzzer

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

```
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%)
stage progress
  now trying : havoc
  stage execs : 2742/32.8k (8.37%)
  total execs : 122k
  exec speed : 161.7/sec
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%
map coverage
  map density : 1.55% / 1.60%
  count coverage : 1.36 bits/tuple
findings in depth
  favored paths : 4 (11.11%)
  new edges on : 8 (22.22%)
  total crashes : 9 (1 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 4
  pending : 25
  pend fav : 0
  own finds : 35
  imported : n/a
stabi
```

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 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

搜索您想了解的产品



AC23

2033Mbps/5G频段4发4收7*6dBi穿墙天线/三芯片架构/支持IPv6



AC18

5口全千兆，光纤网线绝配，500m别墅级覆盖，支持USB3.0存储 1900M 11ac千兆双频双流无线路由器



AC15

一款极速若无物，速度快得超乎你想象的1900M路由器 1900M 11ac双频无线千兆路由器

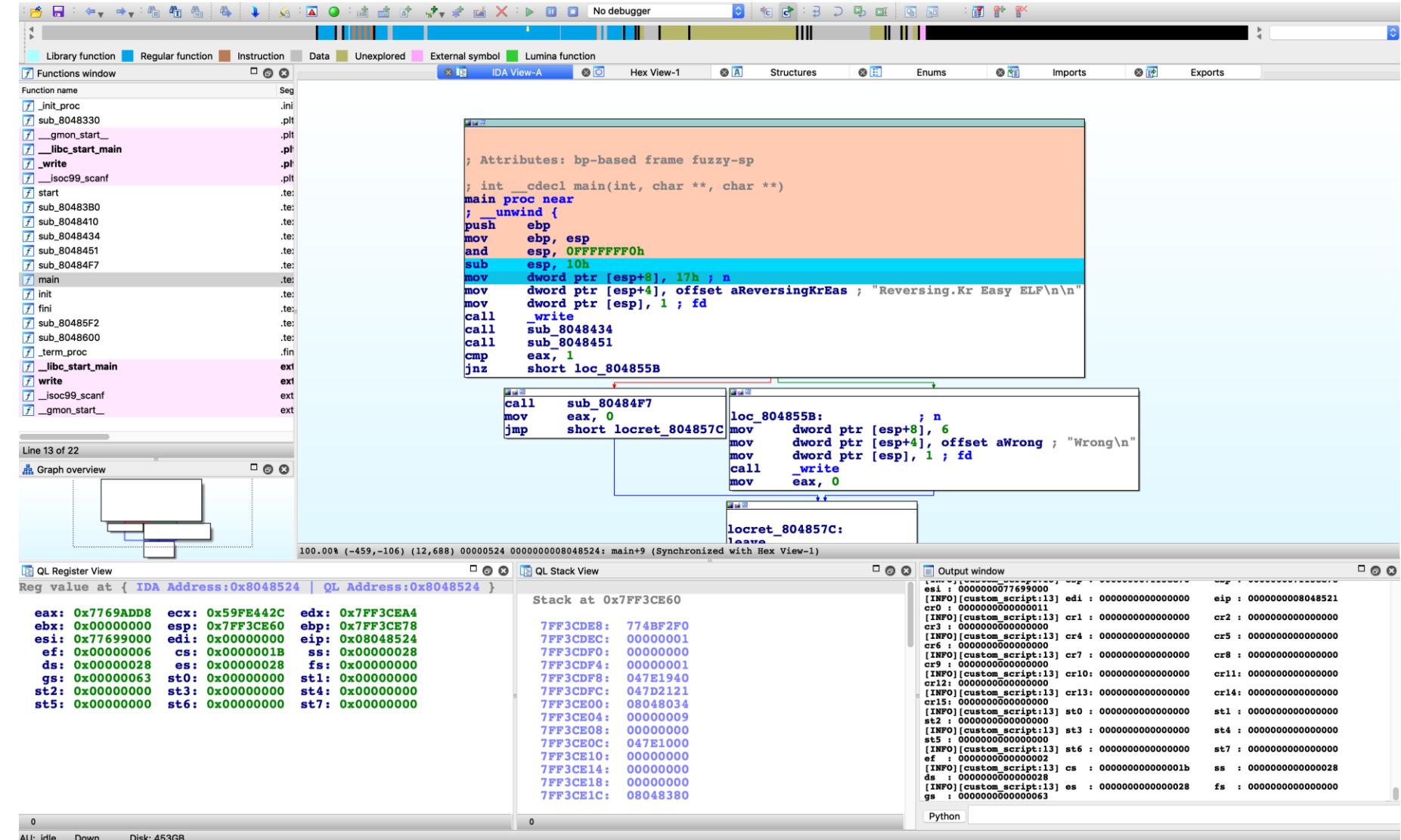
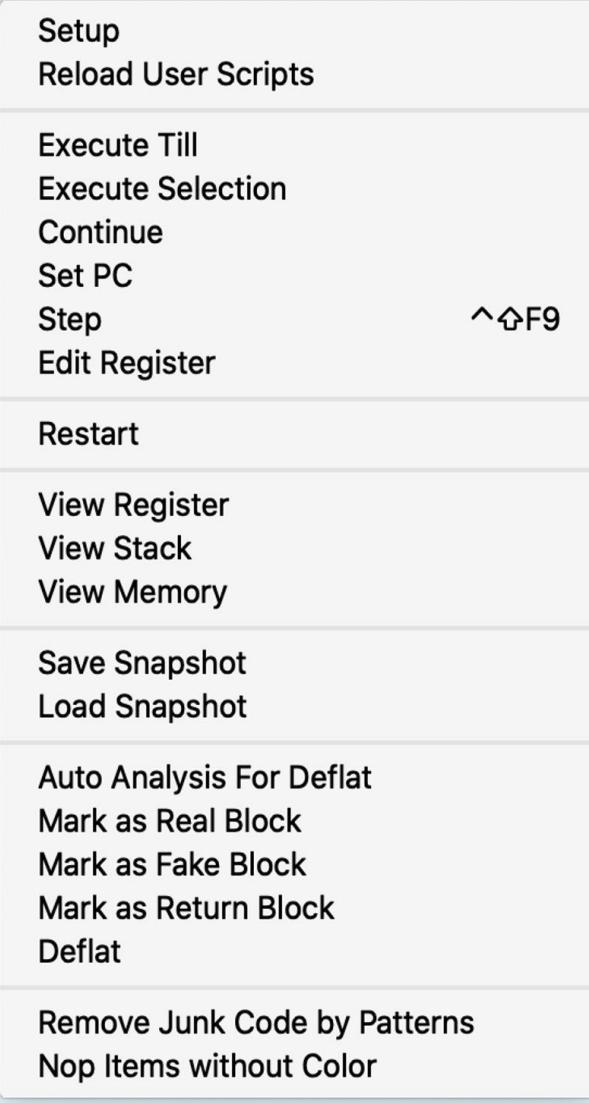
MBR Analysis

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

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

```
fish /Users/mio/qiling
Y ff 0A }0~Vdr\ c0^?mK sJ cE a@ tX aU ukL iV gwS xm jD ^?? 1Z~Gtf3 ^OT nH hD i0
l0 ^FA ↵
~/qiling (doogie_fix_crlf...) $
```

IDA Plugin



Execution path drawing

```
.text:0804851B
.text:0804851B
.text:0804851B ; Attributes: bp-based frame fuzzy-sp
.text:0804851B
.text:0804851B ; int __cdecl main(int, char **, char **)
.text:0804851B main proc near
.text:0804851B ; __ unwind {
.text:0804851B push    ebp
.text:0804851C mov     ebp, esp
.text:0804851E and    esp, 0FFFFFFF0h
.text:08048521 sub    esp, 10h
.text:08048524 mov    dword ptr [esp+8], 17h ; n
.text:0804852C mov    dword ptr [esp+4], offset aReversingKrEas ; "Reversing.Kr Easy ELF\n\n"
.text:08048534 mov    dword ptr [esp], 1 ; fd
.text:0804853B call   _write
.text:08048540 call   sub_8048434
.text:08048545 call   sub_8048451
.text:0804854A cmp    eax, 1
.text:0804854D jnz    short loc_804855B
```

```
.text:0804854F call   sub_80484F7
.text:08048554 mov    eax, 0
.text:08048559 jmp    short locret_804857C
```

```
.text:0804855B loc_804855B:             ; n
.text:0804855B mov    dword ptr [esp+8], 6
.text:08048563 mov    dword ptr [esp+4], offset aWrong ; "Wrong\n"
.text:0804856B mov    dword ptr [esp], 1 ; fd
.text:08048572 call   _write
.text:08048577 mov    eax, 0
```

```
.text:0804857C locret_804857C:
.text:0804857C leave
.text:0804857D retn
.text:0804857D ; } // starts at 804851B
.text:0804857D main endp
.text:0804857D
```

Demo 0

Prepare for Qiling IDA plugin

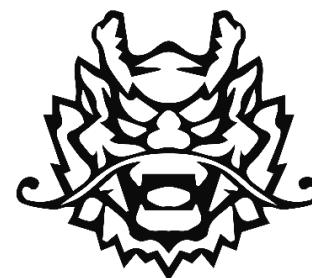
IDA 7.4+



Python3



Qiling



Demo

- Analysis sample:
 - Flare-On 5(2018) Challenge 6 - magic
- This challenge required enter 666 keys
- Only call validation function after each input

```
v20 = 0x2C16022663LL;
memset(v21, 0, sizeof(v21));
for ( i = 0; i < strlen("Run, Forrest, run!!"); i += 4 )
    seed ^= *aRunForrestRun[i];
srand(seed);                                     // seed
if ( sub_402D47() )
{
    v3 = strlen(a2[1]);
    sub_402DCF(a2[1], v3, v11);
    sub_4037BF(*a2);
    puts("Generated first permutation!");
    exit(0);
}
puts("Welcome to the ever changing magic mushroom!");
printf("%d trials lie ahead of you!\n", 666LL);
for ( j = 0; j < 666; ++j )
{
    printf("Challenge %d/%d. Enter key: ", j + 1, 666LL);
    if ( !fgets(v10, 128, stdin) )           // input len=128
        return 0xFFFFFFFFLL;
    v5 = strlen(v10);
    sub_402DCF(v10, v5, v11);               // validate input && change self
    for ( k = 0; k < strlen(v10); ++k )
        *(&v12 + k) ^= v10[k];
    sub_4037BF(*a2);
}
printf("Congrats! Here is your price:\n%s\n", &v12);
return 0LL;
}
```

Prepare for Qiling IDA plugin

https://raw.githubusercontent.com/kabeor/qiling/dev/examples/extensions/idaplug/FlareOn5_6.py
<https://github.com/asuna-amawaka/Flareon5-2018/raw/master/Ch6%20-%20magic.7z>

```
python3 -m venv qiling_env
source qiling_env/bin/activate
cd qiling.git
python -m pip -e .
```

```
import sys
sys.path.append('/Users/home/Pipenv/qiling_run/lib/python3.7/site-packages')
sys.path.append('/Users/home/Documents/Codes/Python/qiling_dev/qiling')
```

Sample Analysis

- The validation function validates the user input in a loop.
- In each iteration of the validation loop, a different part of the entered key is validated.

```
__int64 __fastcall sub_402DCF(__int64 a1, unsigned __int64 a2, __int64 a3)
{
    __int64 result; // rax
    unsigned int i; // [rsp+2Ch] [rbp-4h]

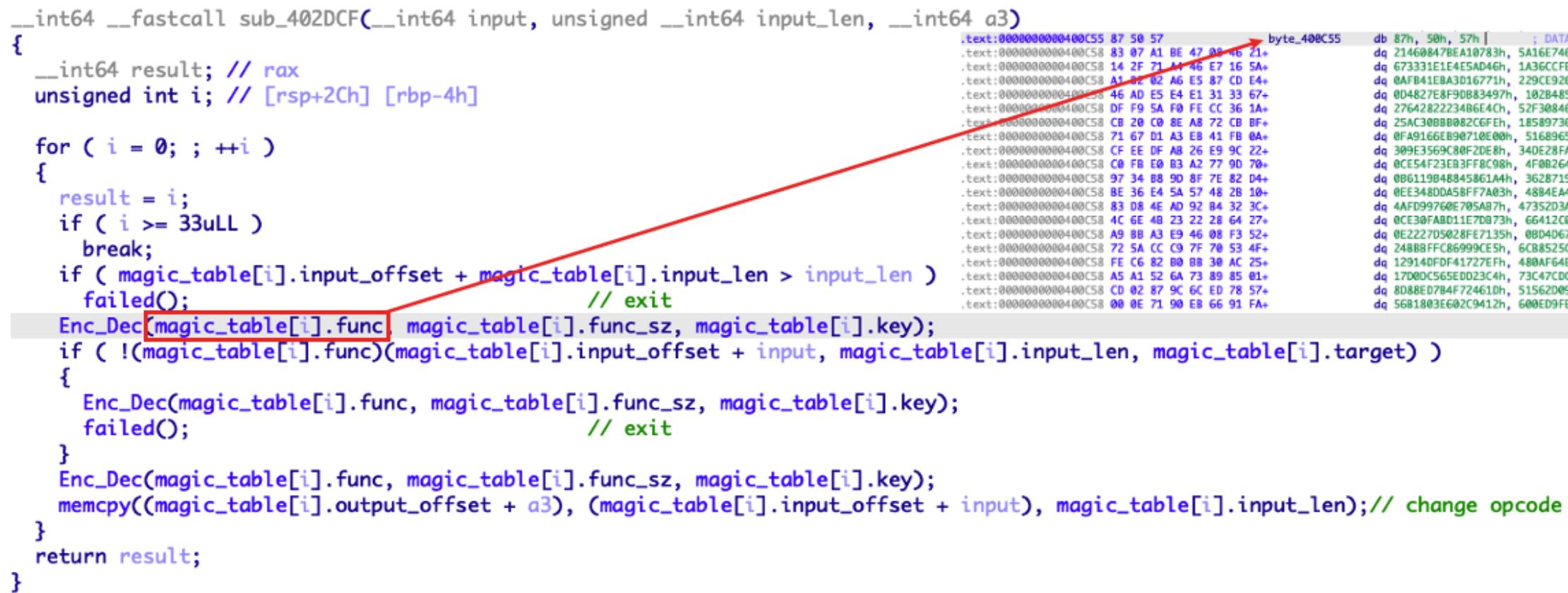
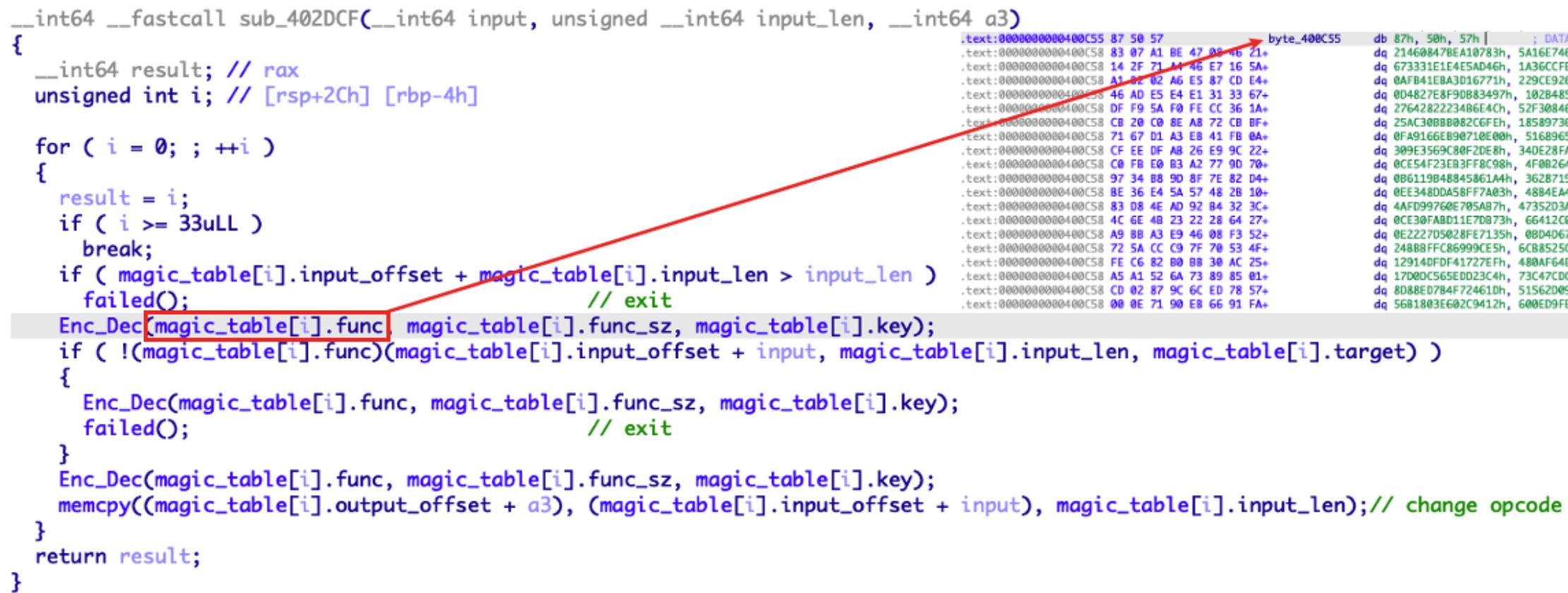
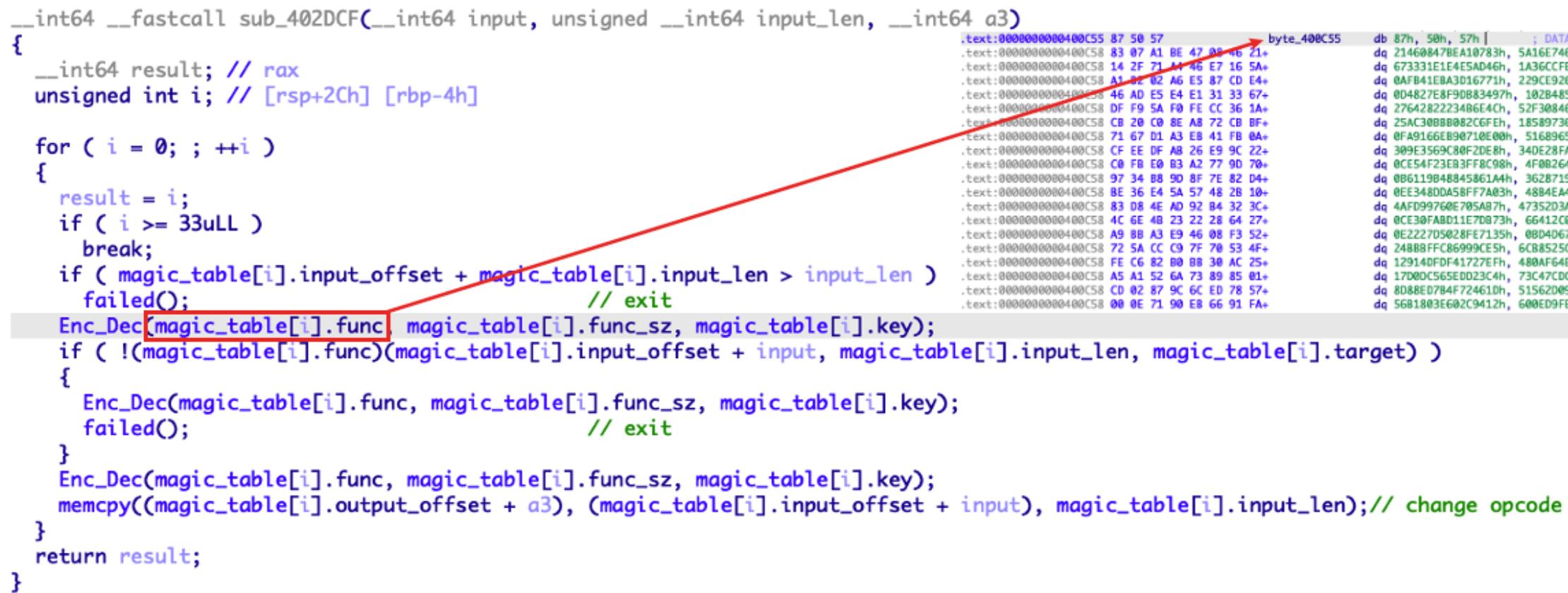
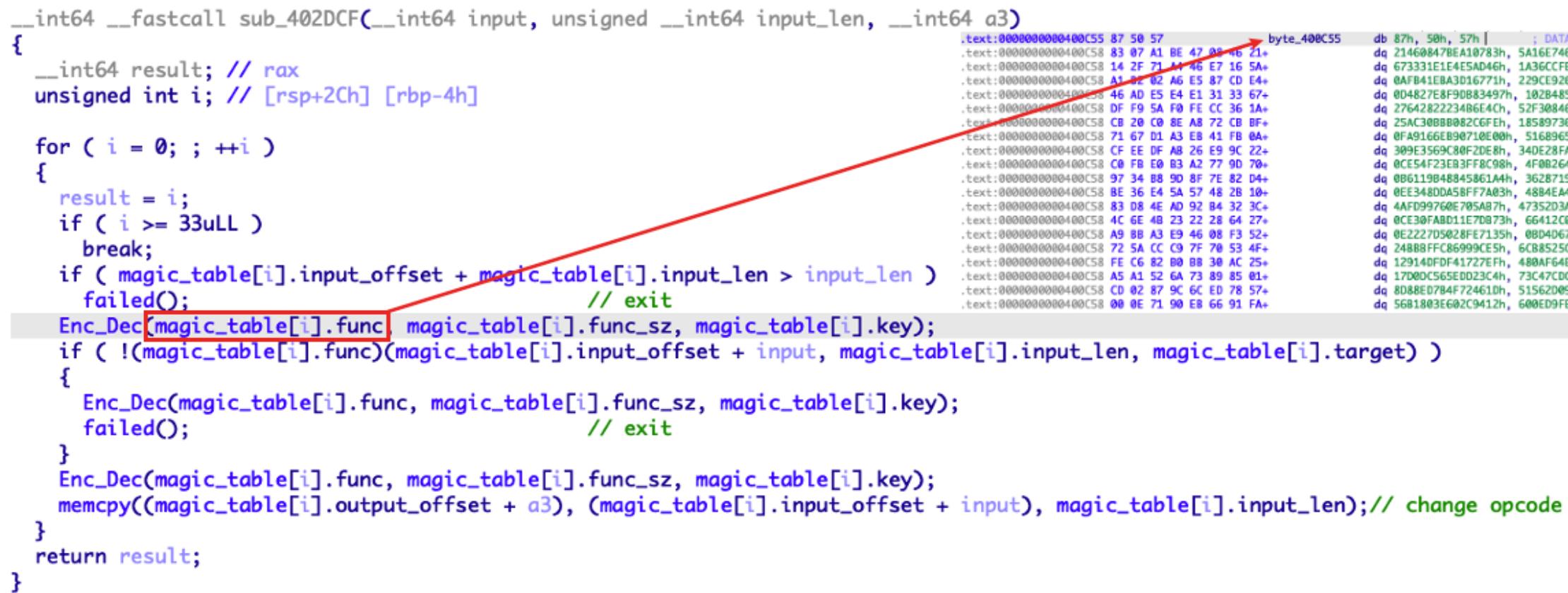
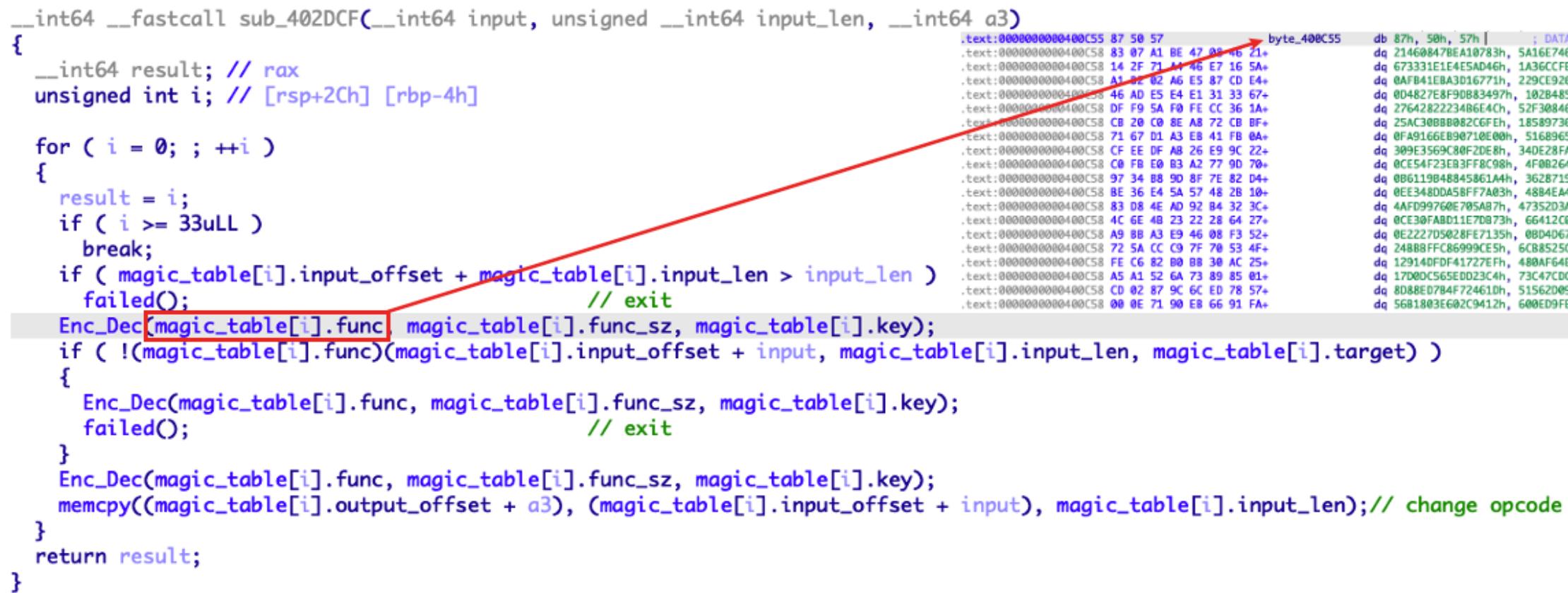
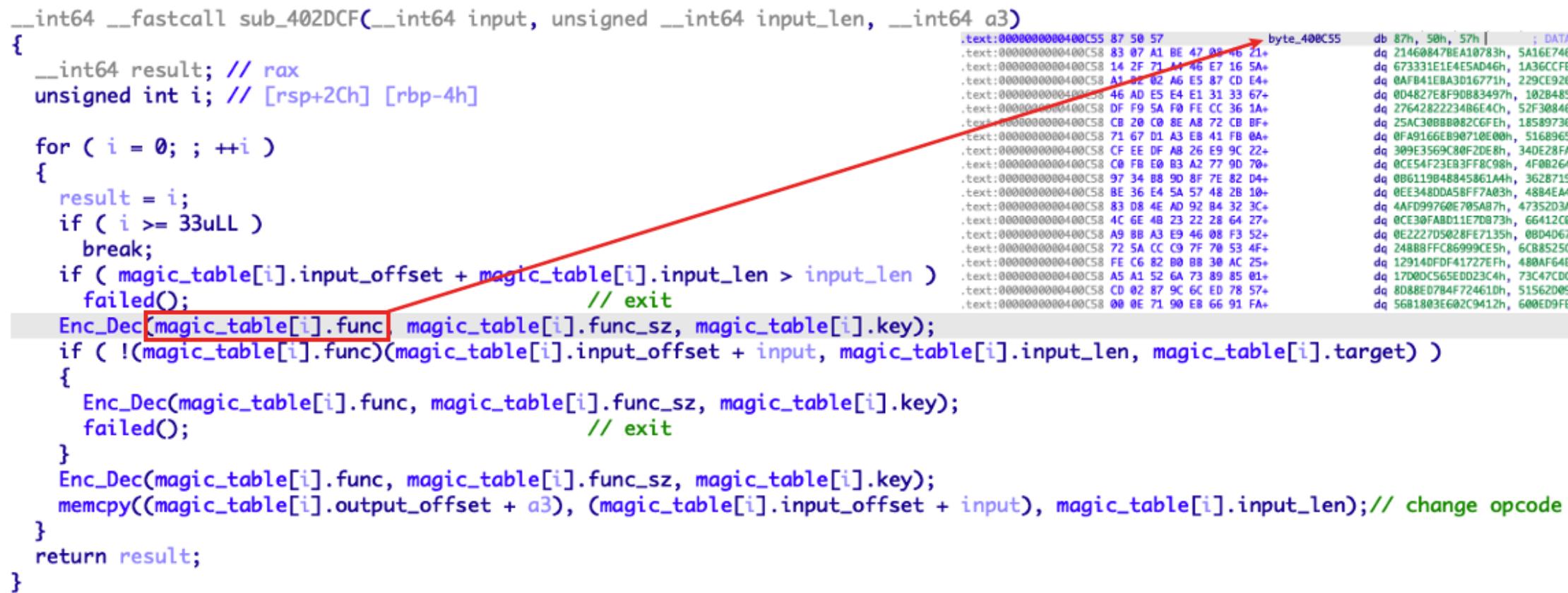
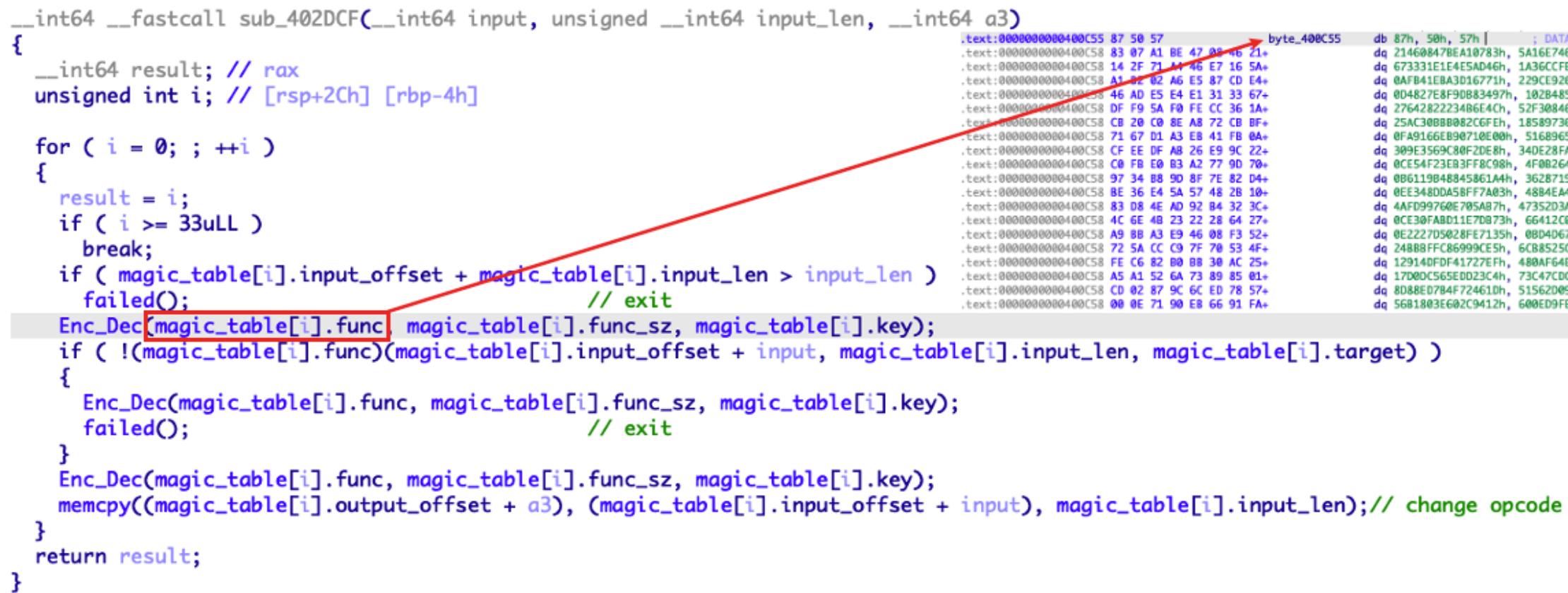
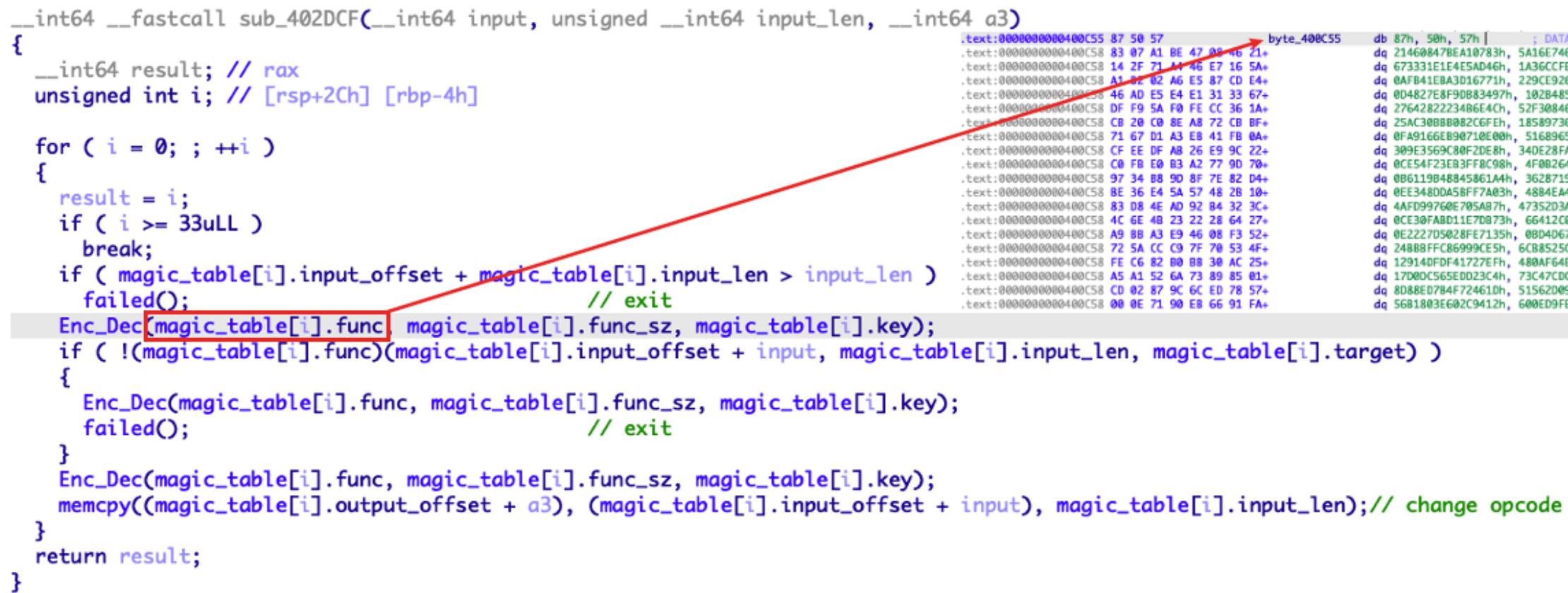
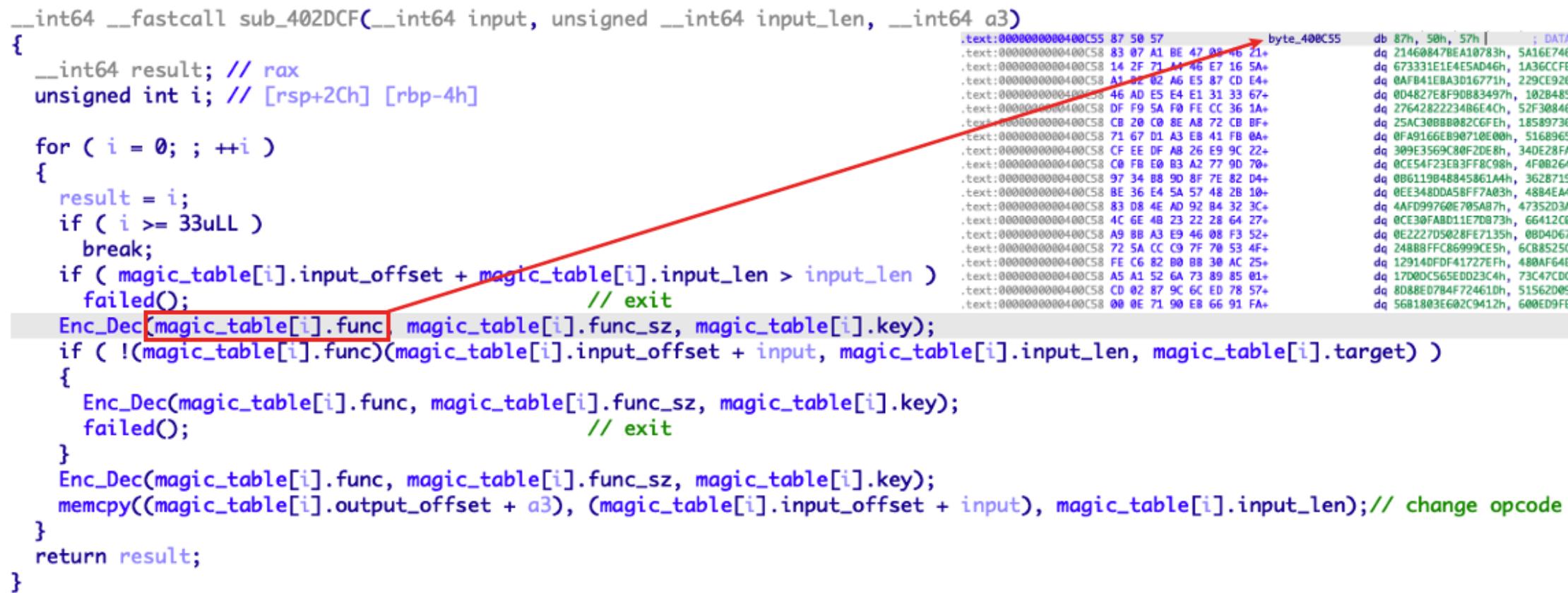
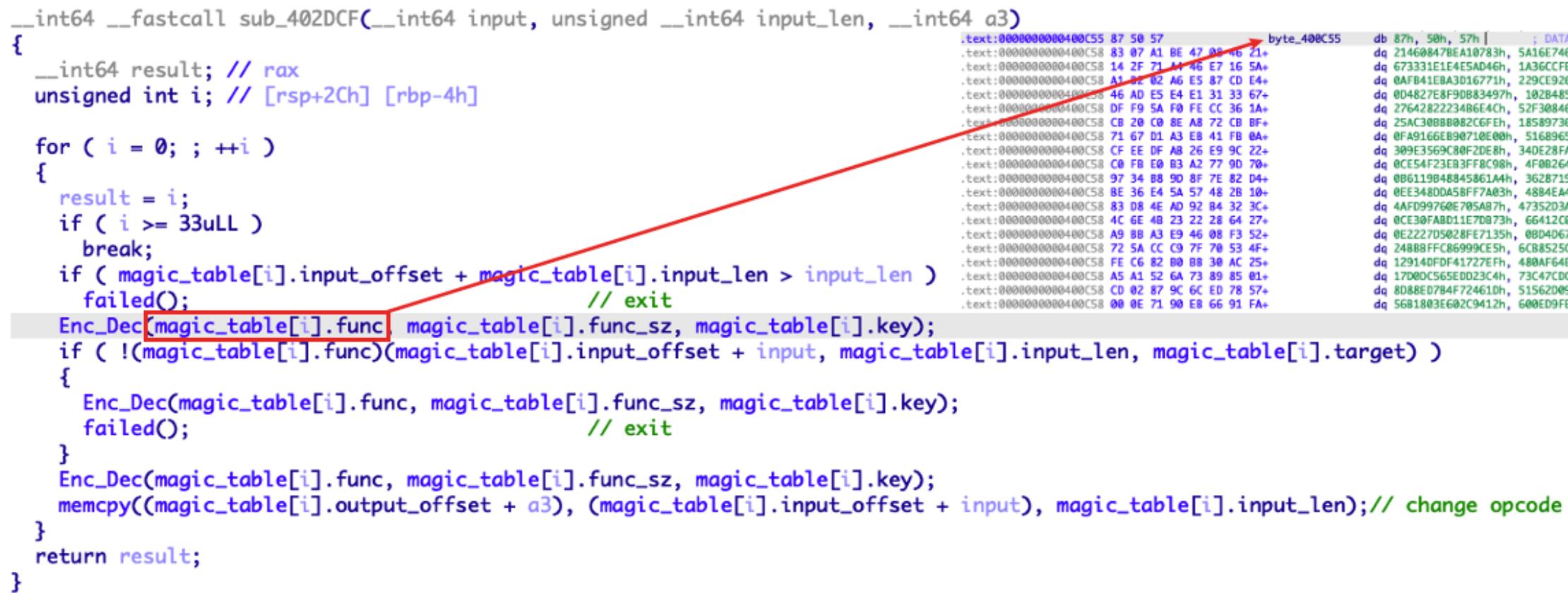
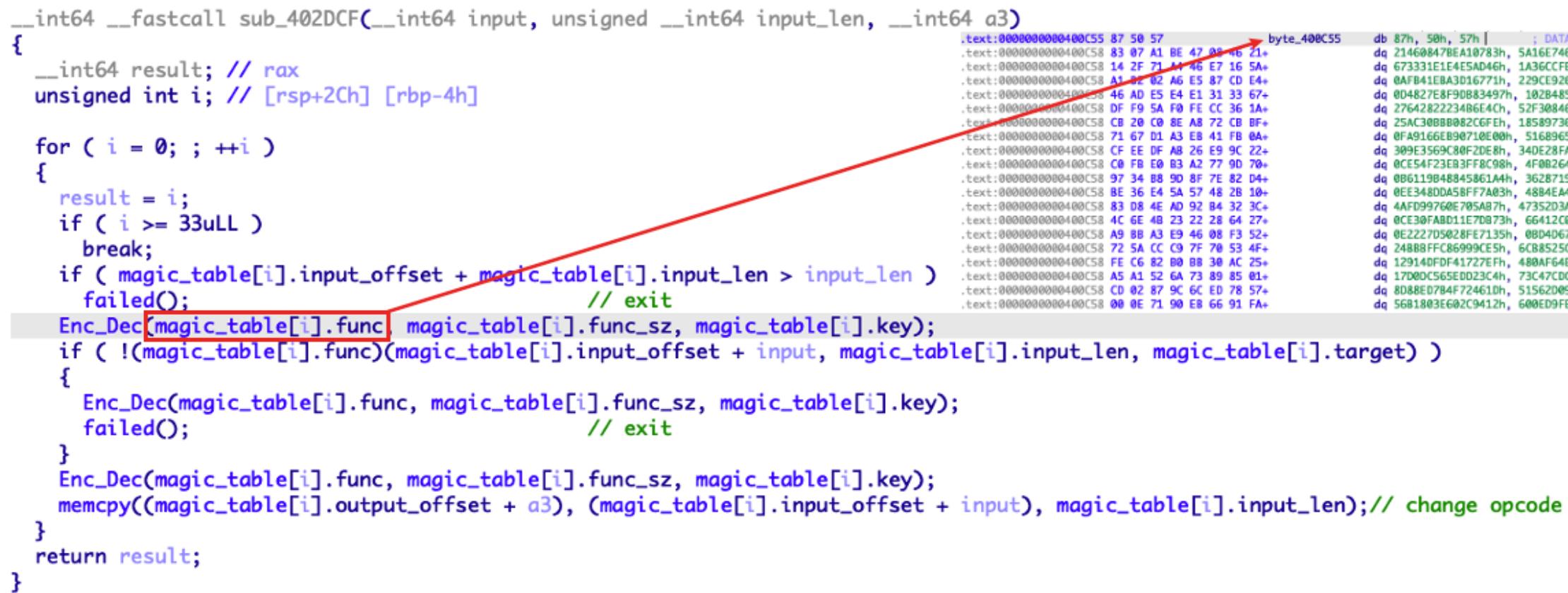
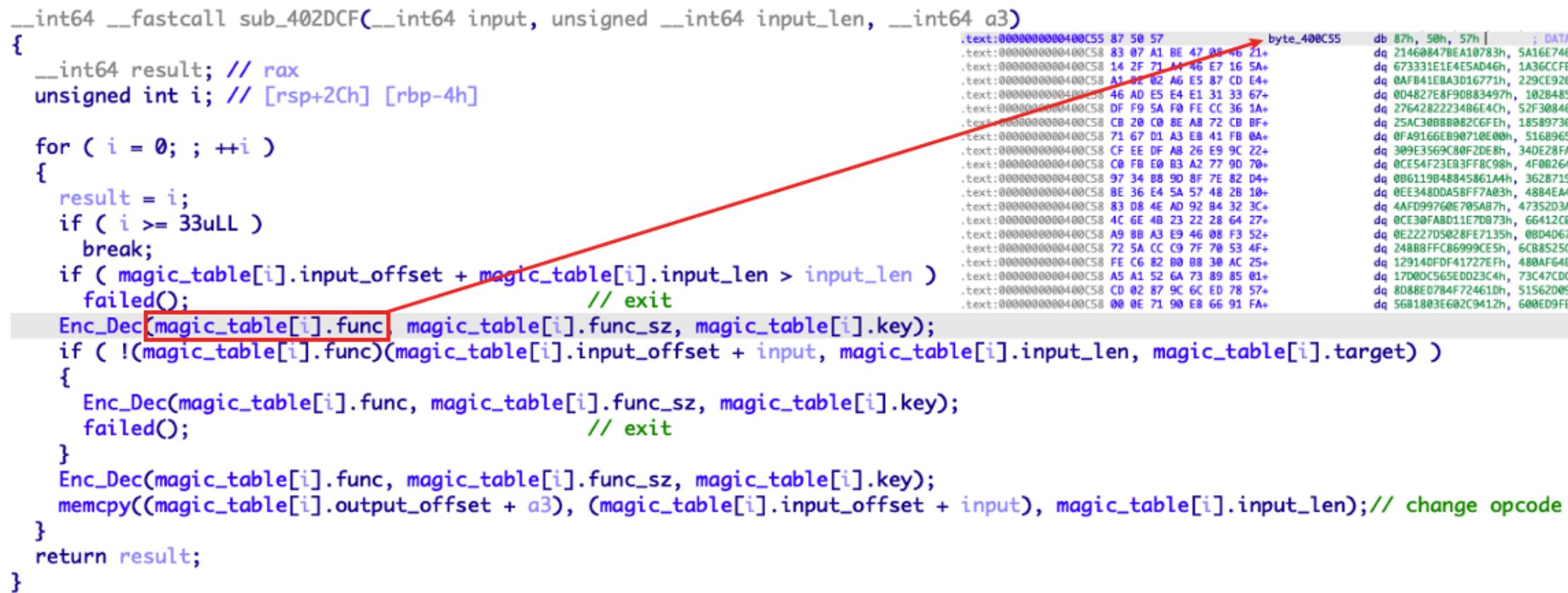
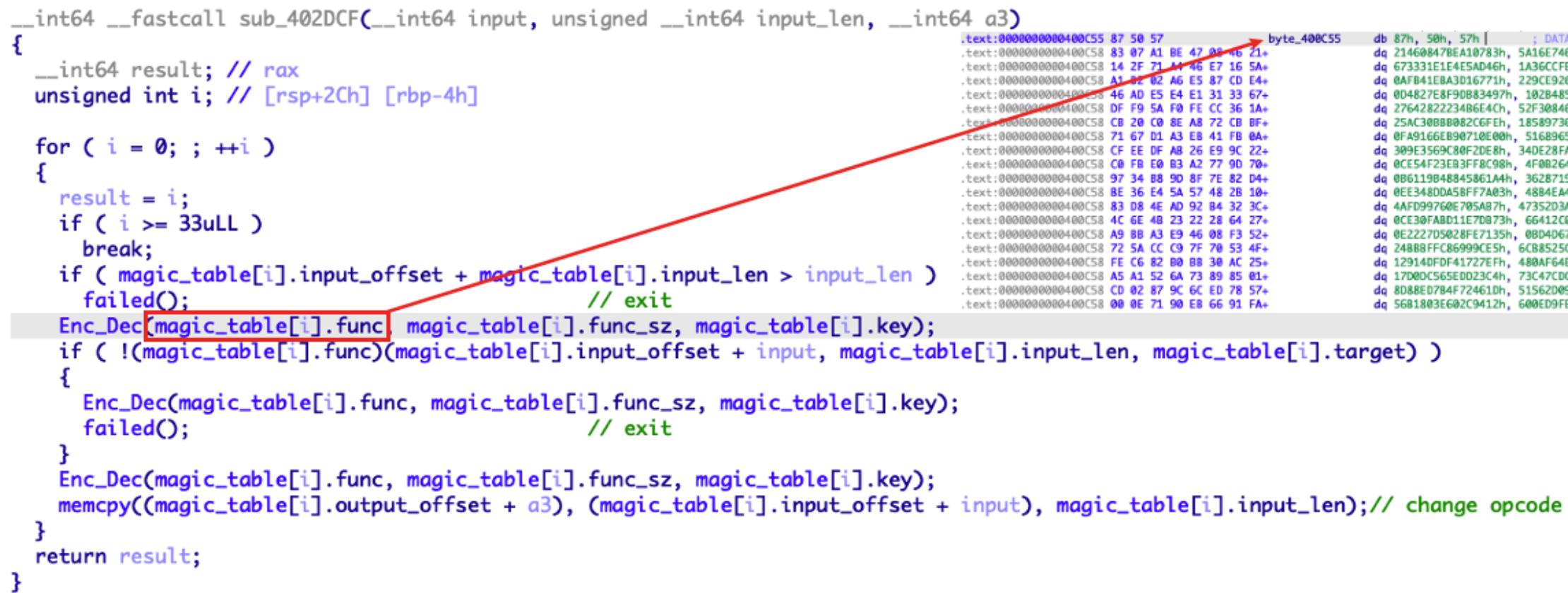
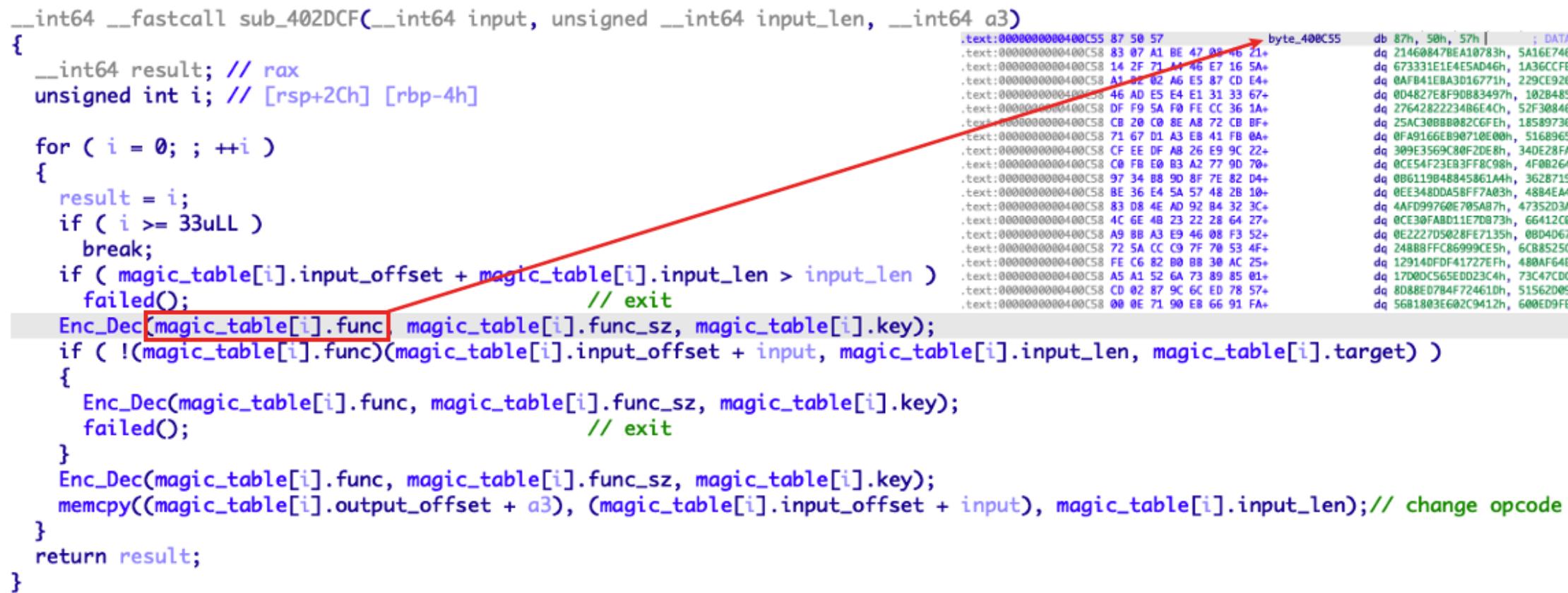
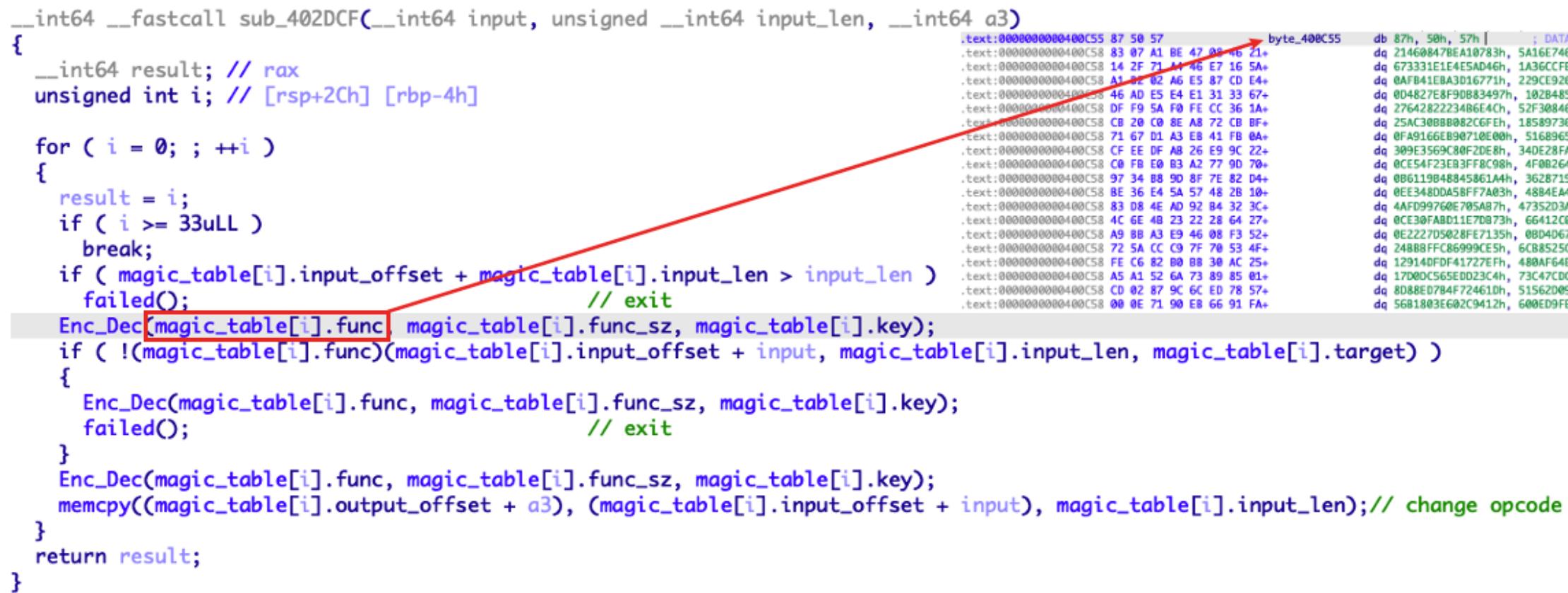
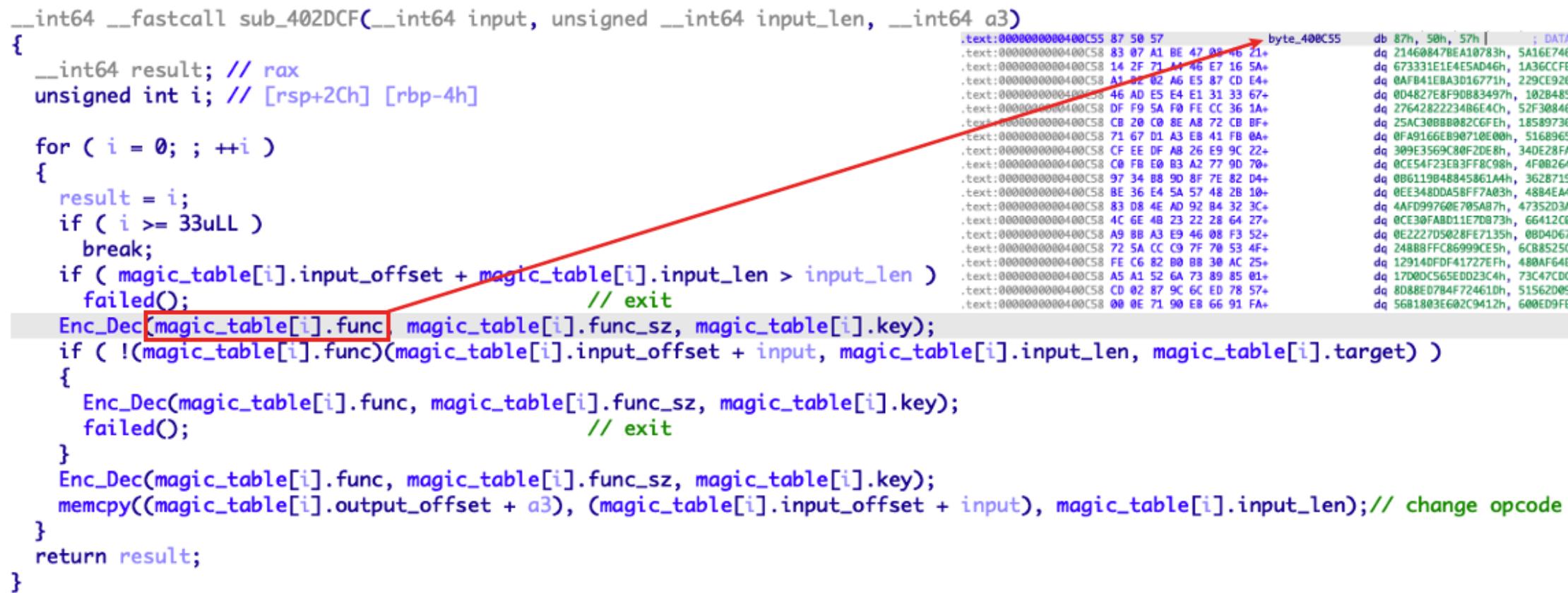
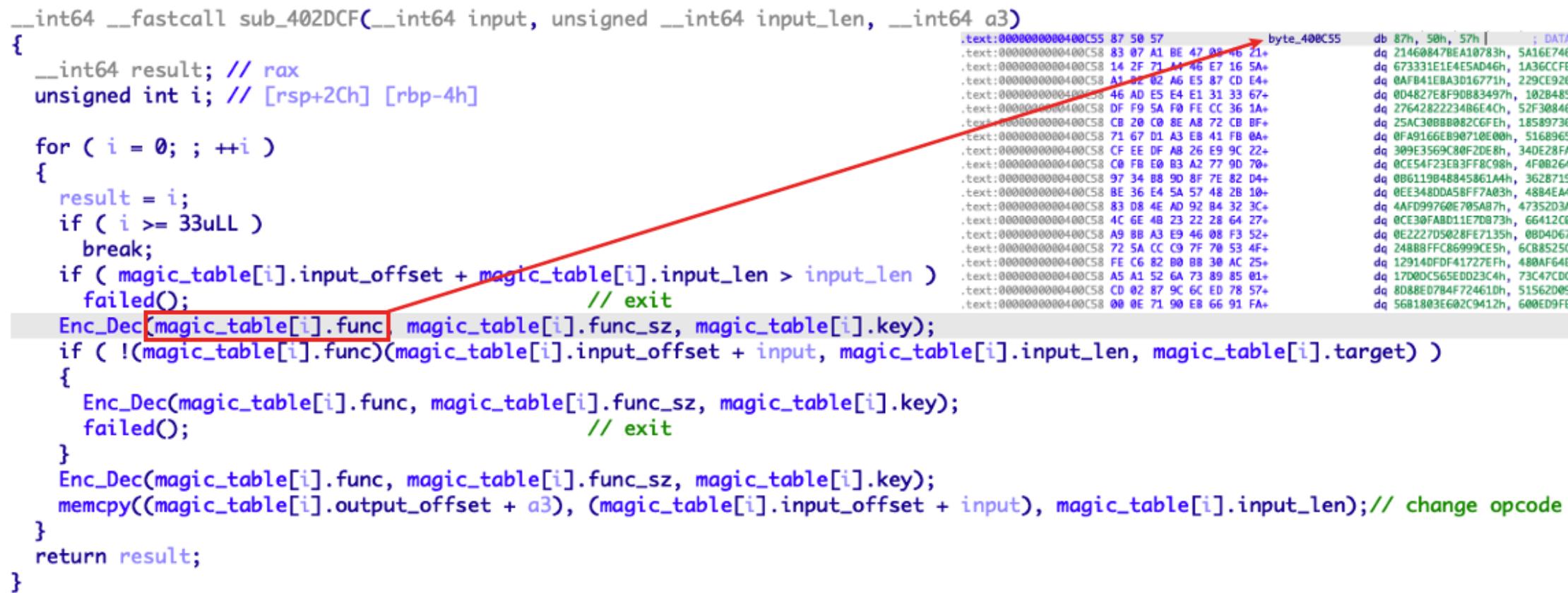
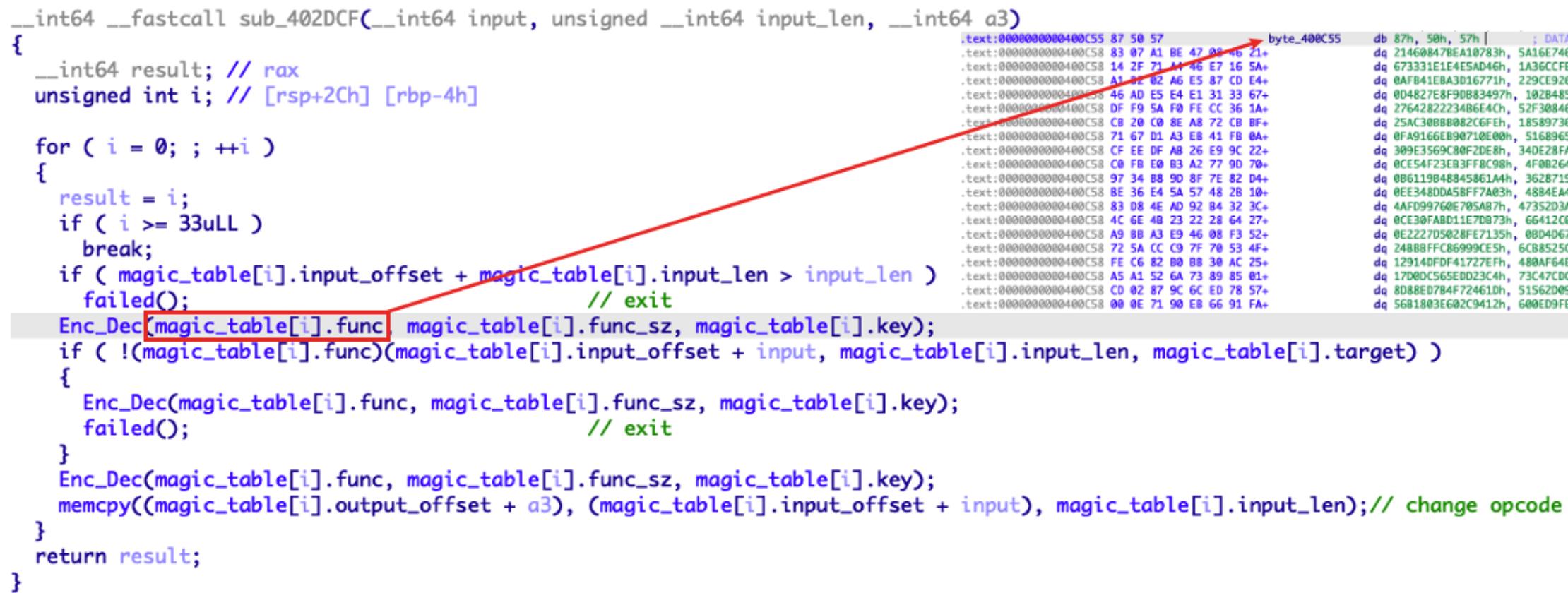
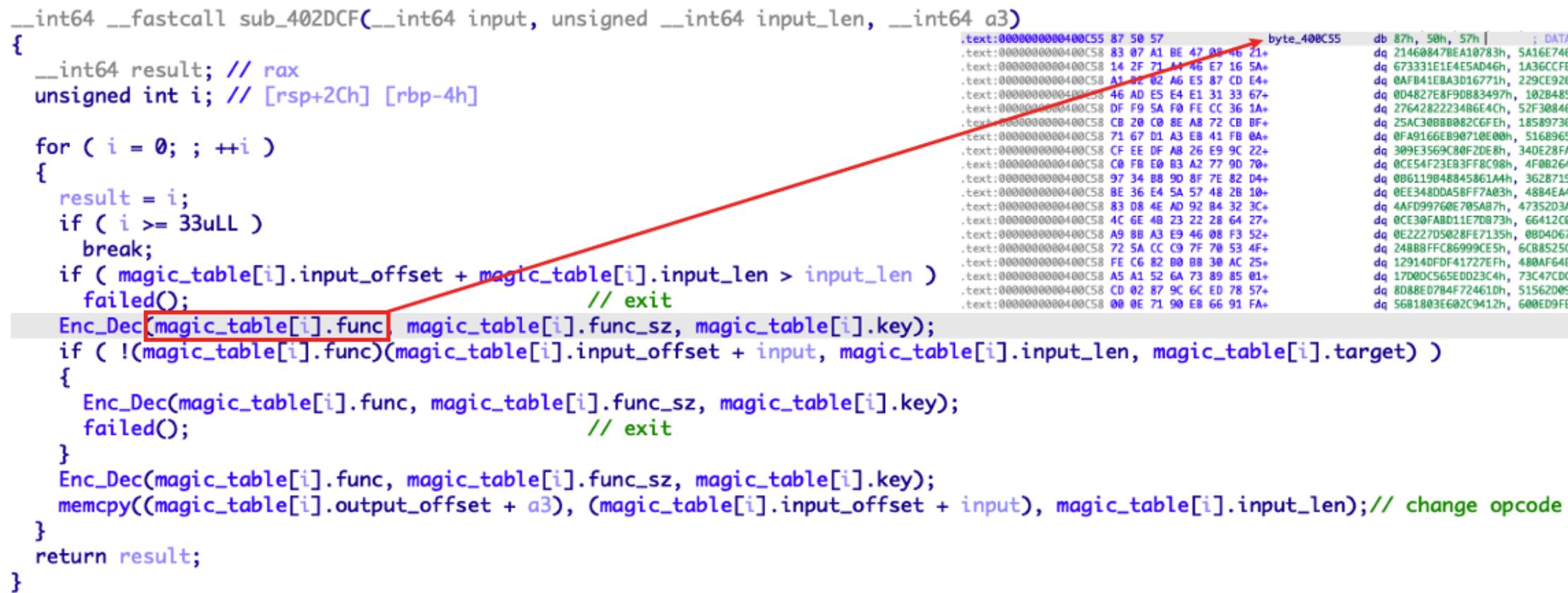
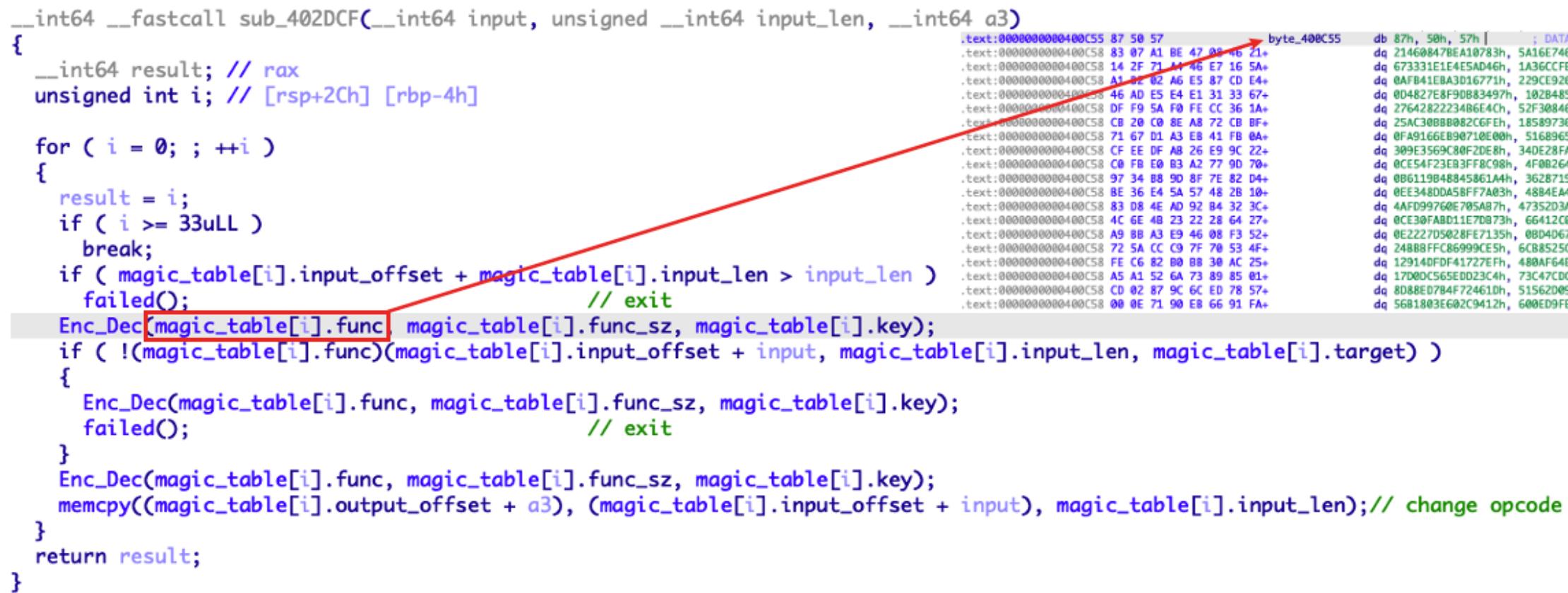
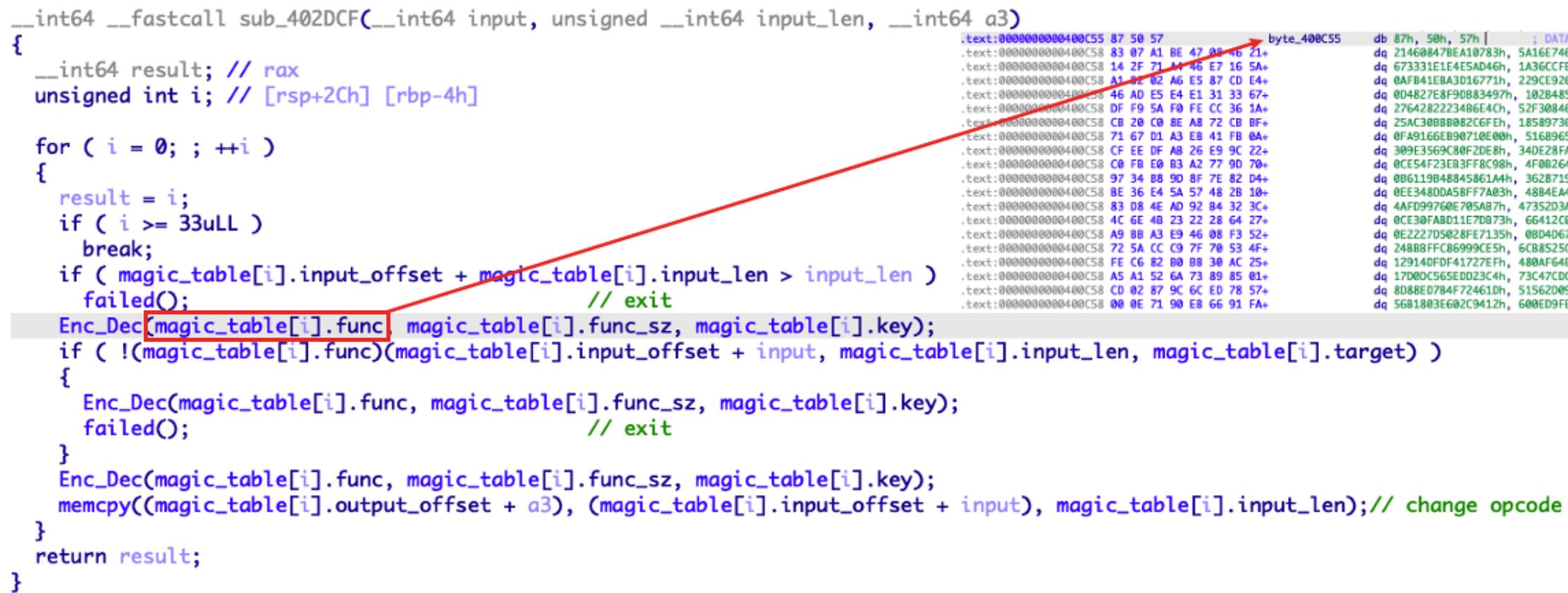
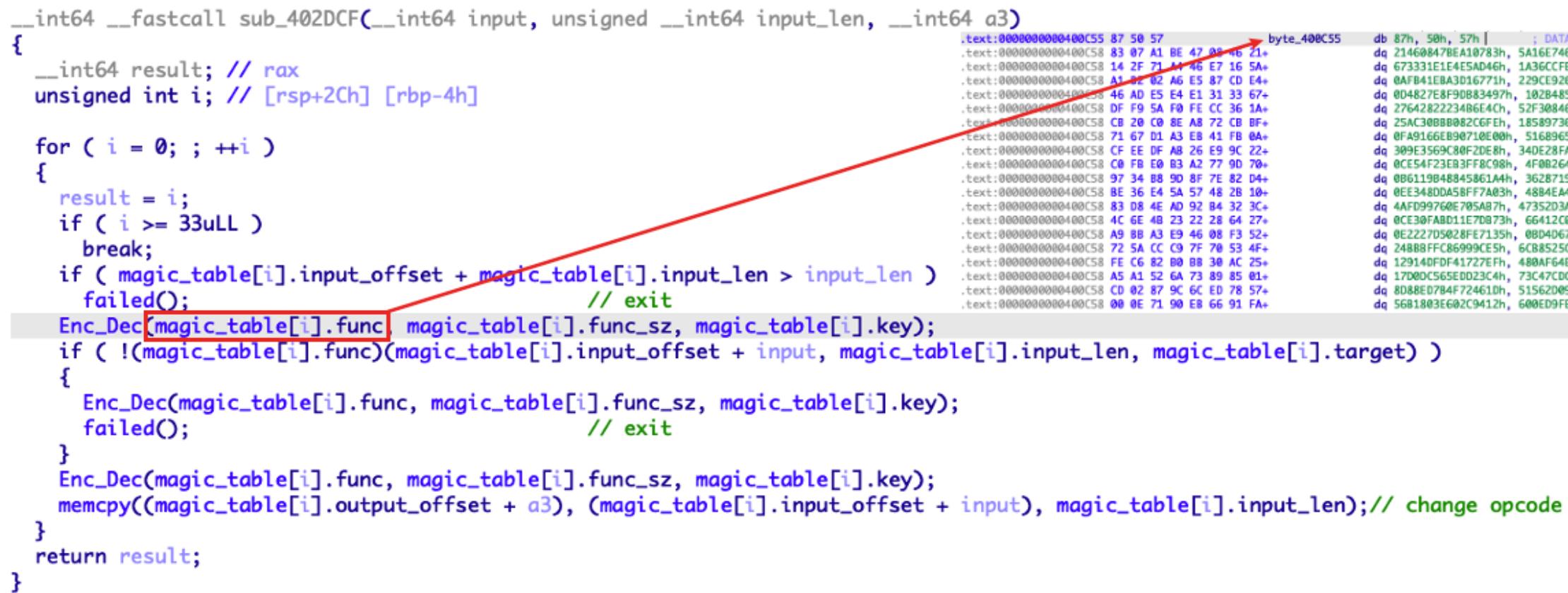
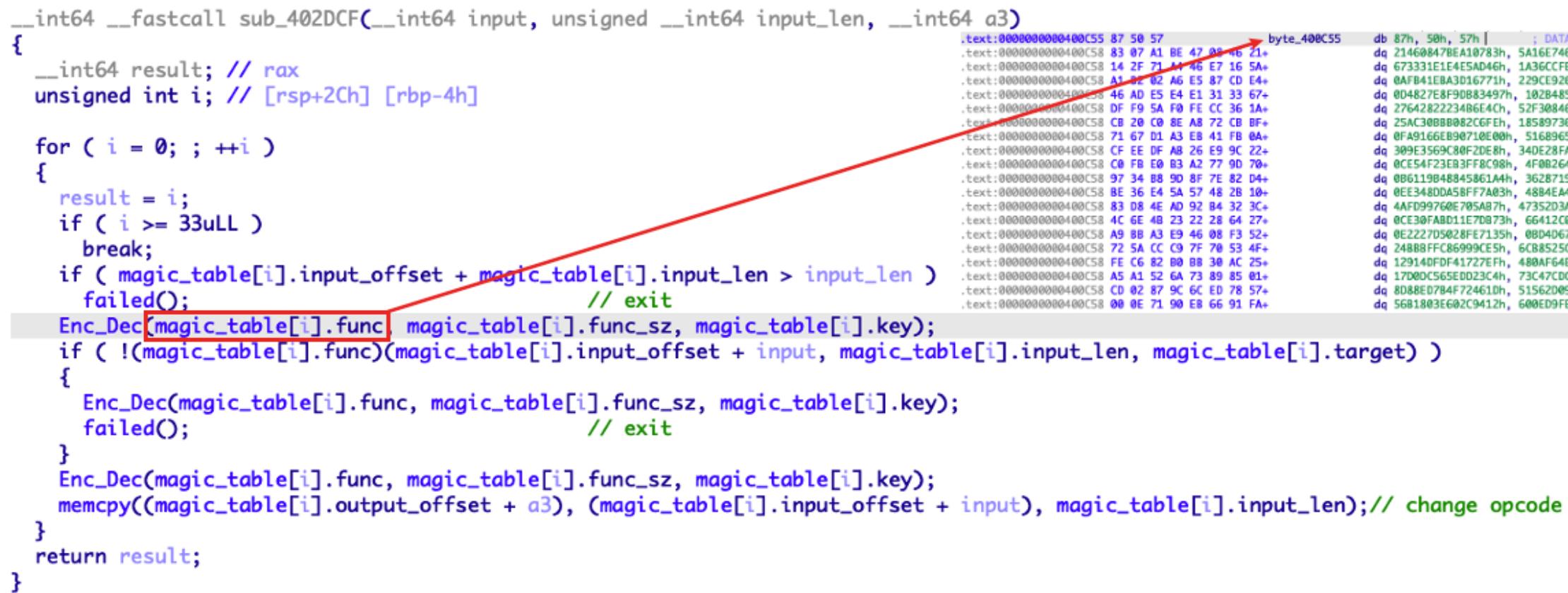
    for ( i = 0; ; ++i )
    {
        result = i;
        if ( i >= 33uLL )
            break;
        if ( dword_605108[72 * i + 1] + dword_605108[72 * i + 2] > a2 )
            sub_402CC7();
        sub_402CDF((&off_605100)[36 * i], dword_605108[72 * i], *(_QWORD *)&dword_605108[72 * i + 4]);
        if ( !((unsigned int (__fastcall *)(__int64, _QWORD, char (**)[3]))(&off_605100)[36 * i])(
            dword_605108[72 * i + 1] + a1,
            dword_605108[72 * i + 2],
            &off_605100 + 36 * i + 4) )
        {
            sub_402CDF((&off_605100)[36 * i], dword_605108[72 * i], *(_QWORD *)&dword_605108[72 * i + 4]);
            sub_402CC7();
        }
        sub_402CDF((&off_605100)[36 * i], dword_605108[72 * i], *(_QWORD *)&dword_605108[72 * i + 4]);
        memcpy(
            (void *)(dword_605108[72 * i + 3] + a3),
            (const void *)(dword_605108[72 * i + 1] + a1),
            dword_605108[72 * i + 2]);
    }
    return result;
}
```

Sample Analysis

- After analysis, it is found that the verification algorithm is stored in `magic_table`.
- "`magic_table`" is **encrypted** but it will **decrypt in memory** in each iteration of the validation loop.
- The verification algorithm will automatically change after each round of verification.

```
_int64 __fastcall sub_402DCF(_int64 input, unsigned __int64 input_len, __int64 a3)
{
    __int64 result; // rax
    unsigned int i; // [rsp+2Ch] [rbp-4h]

    for ( i = 0; ; ++i )
    {
        result = i;
        if ( i >= 33uLL )
            break;
        if ( magic_table[i].input_offset + magic_table[i].input_len > input_len )
            failed(); // exit
        Enc_Dec[magic_table[i].func, magic_table[i].func_sz, magic_table[i].key];
        if ( !(magic_table[i].func)(magic_table[i].input_offset + input, magic_table[i].input_len, magic_table[i].target) )
        {
            Enc_Dec(magic_table[i].func, magic_table[i].func_sz, magic_table[i].key);
            failed(); // exit
        }
        Enc_Dec(magic_table[i].func, magic_table[i].func_sz, magic_table[i].key);
        memcpy((magic_table[i].output_offset + a3), (magic_table[i].input_offset + input), magic_table[i].input_len); // change opcode
    }
    return result;
}
```



Sample Analysis

- We will use qiling to:
 - emulate this challenge
 - decrypt magic_table
 - cover bytes to asm
- Use IDA Pro Hex-Rays Decompiler to:
 - decompile the verification algorithm

Encrypted



Decrypted



Decompile

```
byte_400C55 db 87h, 50h, 57h, ... ; DATA XREF: .data:magi
46 21+ dq 21460847BEA10783h, 5A16E746A4712F14h, 0E4CD8
16 5A+ dq 673331E1E4E5AD46h, 1A36CCFF05AF9DFh, 0FBCB7
CD E4+ dq 0AF841E8A3D1671h, 229CE926ABDFEECFh, 709077
33 67+ dq 04827E8F9D883497h, 102848575AE436BEh, 3C32B
36 1A+ dq 27642822234864ECh, 52F30846E9A388A9h, 0F5370
CB 8F+ dq 25AC3088B082C6FEh, 18589736A52A1A5h, 5778ED6
FB 0A+ dq 0F9166EB90710E00h, 51689656F17A710Dh, 0C2A3
9C 22+ dq 309E3569C80F2DE8h, 34DE28FAEF0D02C3h, 6EBE43
9D 70+ dq 0CE54F23EB3FF8C98h, 4F0B2641A29A1D9Ch, 2E442
82 D4+ dq 08611984848561A4h, 36287197E8C8F151h, 0EAF0
2B 10+ dq 0E3348DAS5BF7A03h, 4884EA44C0A0B59h, 0E910
32 3C+ dq 44FD99760E705AB7h, 47352D3A80D36911h, 8D2103
64 27+ dq 0CE30FABD11E7DB73h, 66412C0F99611B72h, 6B14D
F3 52+ dq 0E2227D5028FE7135h, 0B4D67CB2A1EAAh, 0D3A
53 4F+ dq 2488BFFC86999CE5h, 6CB8525C8D47C3C8h, 36387D
AC 25+ dq 129140DFD417272EFh, 480AF64E8AF0A30h, 546066
85 01+ dq 1700DC565ED023C4h, 73C47CDC8066803Bh, 67F8F4
78 57+ dq 8088ED784F72461Dh, 51562D093DFF4F5C7h, 0BE16E
91 FA+ dq 5681803E602C9412h, 600ED9FESF021977h, 082F30
68 51+ dq 158349D548AC94Fh, 18C333FB0C6BE8A6h, 0CEF39
A3 C2+ dq 0A8A760582DB998ADh, 68FE2EA2BD42F88h, 0FAE1
9E 30+ dq 87C3AF489E97732Ch, 910579303F62A8F1h, 0A9E0D
DE 34+ dq 0FCC6C80A76ADE0Bh, 0E088546B20A50859h, 0E4F
BE 6E+ dq 52742720F0B93280h, 0C309AFB3D71A89EBh, 2C3F6
54 CE+ dq 70A7274301FF612Bh, 0CBA3F22455335539h, 8F0AC
0B 4F+ dq 9989847A29D4A2Ch, 0C3C2939831A04756h, 0B62E
44 2E+ dq 000ED590E54AC43C3h, 0C939E7806458FB73Eh, 0B0B
11 86+ dq 980E8CFD5323D736h, 0B0F7DE0A7F561D18h, 0D3B8
28 36+ dq 427C8469E6675A3h, 78F332BF2A2F5C7h, 4B387D
FC EA+ dq 0BAC446F96BC591EAh, 0BA627DF99F75648Fh, 8F33
```

```
400C55 sub_400C55 proc near
400C55 var_58 = qword ptr -58h
400C55 var_4C = dword ptr -4Ch
400C55 var_48 = qword ptr -48h
400C55 var_38 = qword ptr -38h
400C55 var_30 = qword ptr -30h
400C55 var_28 = qword ptr -28h
400C55 var_20 = qword ptr -20h
400C55 var_18 = qword ptr -18h
400C55 var_10 = qword ptr -10h
400C55 var_6 = byte ptr -6
400C55 var_5 = byte ptr -5
400C55 var_4 = dword ptr -4
400C55 push rbp
400C55 48 89 E5 mov rbp, rsp
400C59 48 89 7D 88 mov [rbp+var_48], rdi
400C60 89 75 B4 mov [rbp+var_4C], esi
400C60 48 89 55 A8 mov [rbp+var_58], rdx
400C64 C7 45 FC 00 00 00 00 mov [rbp+var_4], 0
400C68 E9 19 01 00 00 jmp loc_400D89
400C70 ; -----
400C70 loc_400C70: test rax, rax
400C70 48 85 C0 mov eax, [rbp+var_4]
400C70 48 8D 14 C5 00 00 00 00 lea rdx, ds:0[rax*8]
400C78 48 8B 45 A8 mov rax, [rbp+var_58]
400C7F 48 01 D0 add rax, rdx
400C82 48 8B 00 mov rax, [rax]
400C85 48 85 C0 test rax, rax
400C88 75 7A jnz short loc_400D04
400C8A 48 45 FC mov eax, [rbp+var_4]
400C8D 48 8D 14 C5 00 00 00 00 lea rdx, ds:0[rax*8]
```

```
int64 __fastcall sub_400C55(int64 a1, unsigned int a2, __int64 a3)
{
    __int64 v4; // [rsp+20h] [rbp-38h]
    __int64 v5; // [rsp+28h] [rbp-30h]
    __int64 v6; // [rsp+30h] [rbp-28h]
    __int64 v7; // [rsp+38h] [rbp-20h]
    __int64 v8; // [rsp+40h] [rbp-18h]
    __int64 v9; // [rsp+48h] [rbp-10h]
    char v10; // [rsp+52h] [rbp-6h]
    char v11; // [rsp+53h] [rbp-5h]
    unsigned int i; // [rsp+54h] [rbp-4h]

    for ( i = 0; i < a2; ++i )
    {
        if ( *(8LL * i + a3) )
        {
            v11 = *(i + a1);
            v6 = 0LL;
            v5 = 1LL;
            v4 = 0LL;
            while ( v11 )
            {
                v6 = v5 + v4;
                v4 = v5;
                v5 = v6;
                --v11;
            }
            if ( v6 != *(8LL * i + a3) )
                return 0LL;
        }
        else
        {
            v10 = *(i + a1);
            v9 = 0LL;
```

Demo 1

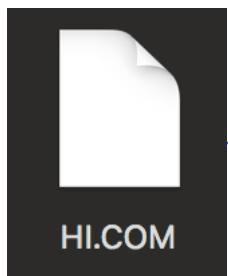
Agenda

- Diving into Qiling Framework – 10min
 - Show how real mode emulation is implemented.
 - Learn the internal design of the Qiling Framework.
 - A good start if you would like become a contributor.
- Solve a CTF Challenge with MBR emulation. – 10min + 10min
 - How Qiling API helps our analysis.

Qiling Internals

Basics

- Core problem: How Qiling emulates a real mode binary? Two Layers.
 - Loader: Parse binary and load it into memory.
 - OS: Implement interrupts.
- While the instrumentation is provided as an API, it is also heavily used internally to implement the OS layer.



```
seg000:0100 ; Attributes: noreturn
seg000:0100
seg000:0100
seg000:0100 start
seg000:0100
seg000:0102
seg000:0105
seg000:0105
seg000:0107
seg000:010A
seg000:010A start
```

```
public start
proc near
    mov    ah, 9
    mov    dx, 10Dh
    int    21h
    mov    ax, 4C00h
    int    21h
    endp
```

Emulation starts here.

Trap into Qiling.

Loader Layer

- Target binary: MBR file && COM file.
 - DOS EXE support is still WIP.
- Pretty similar, memory image without any header.
 - Setup registers, memory map and write the file into memory.
 - But disk image should be mounted for MBR file.
- `qiling/loader/dos.py`

OS Layer

- The place where we implement traditional interrupts.
- Example: INT 13h, ah=42h, read disk sectors.
 - Implemented with fs_mapper API.
 - Map any object which implements FsMappedObject interface to an emulated device/path.
 - QIDisk is inherited from FsMappedObject with CHS and LBA support.
 - Note that we mount the MBR file itself in loader.
- os/dos/dos.py
- os/mapper.py
- os/disk.py

```
if not self ql.os.fs_mapper.has_mapping(0x80):  
    self ql.os.fs_mapper.add_fs_mapping(0x80, QlDisk(path, 0x80))
```

Loader

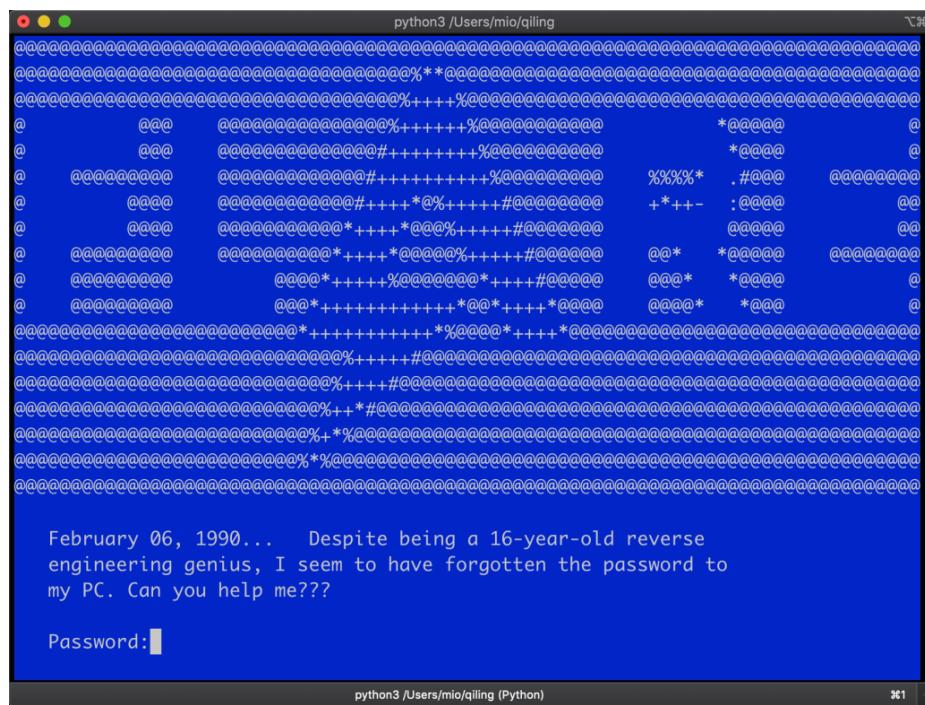
OS

```
disk = self ql.os.fs_mapper.open(idx, None)  
content = disk.read_sectors(lba, cnt)
```

Solve a CTF Challenge

Sample Analysis

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



python3 /Users/mio/qiling

```
Y ff 0A }0~Vdr\ c0^?mK sJ cE a@ tX aU ukL iV gwS xm jD ^?? 1Z~Gtf3 ^0T nH hD iO
10 ^FA
```

~/q/e/r/8/doogie (doogie|...) \$ file doogie.bin

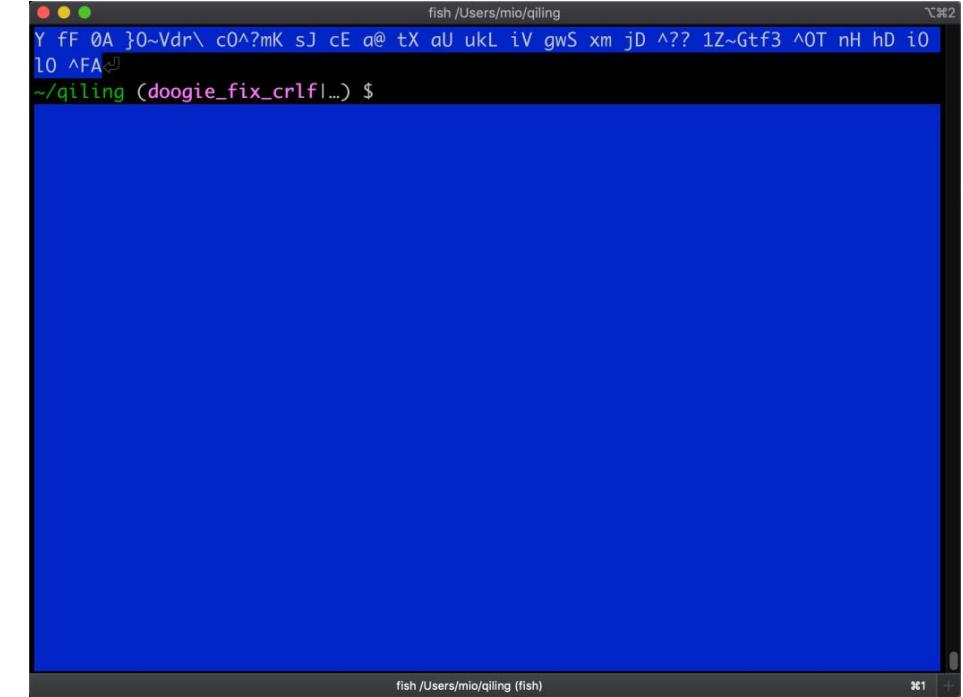
doogie.bin: DOS/MBR boot sector; partition 1 : ID=0x7, active, start-CHS (0x0,32,33), end-CHS (0x3ff,254,63), startsector 2048, 41938944 sectors

~/q/e/r/8/doogie (doogie|...) \$

February 06, 1990... Despite being a 16-year-old reverse engineering genius, I seem to have forgotten the password to my PC. Can you help me???

Password:

python3 /Users/mio/qiling (Python)



fish /Users/mio/qiling

```
Y ff 0A }0~Vdr\ c0^?mK sJ cE a@ tX aU ukL iV gwS xm jD ^?? 1Z~Gtf3 ^0T nH hD iO
10 ^FA
```

~/qiling (doogie_fix_crlf|...) \$

fish /Users/mio/qiling (fish)

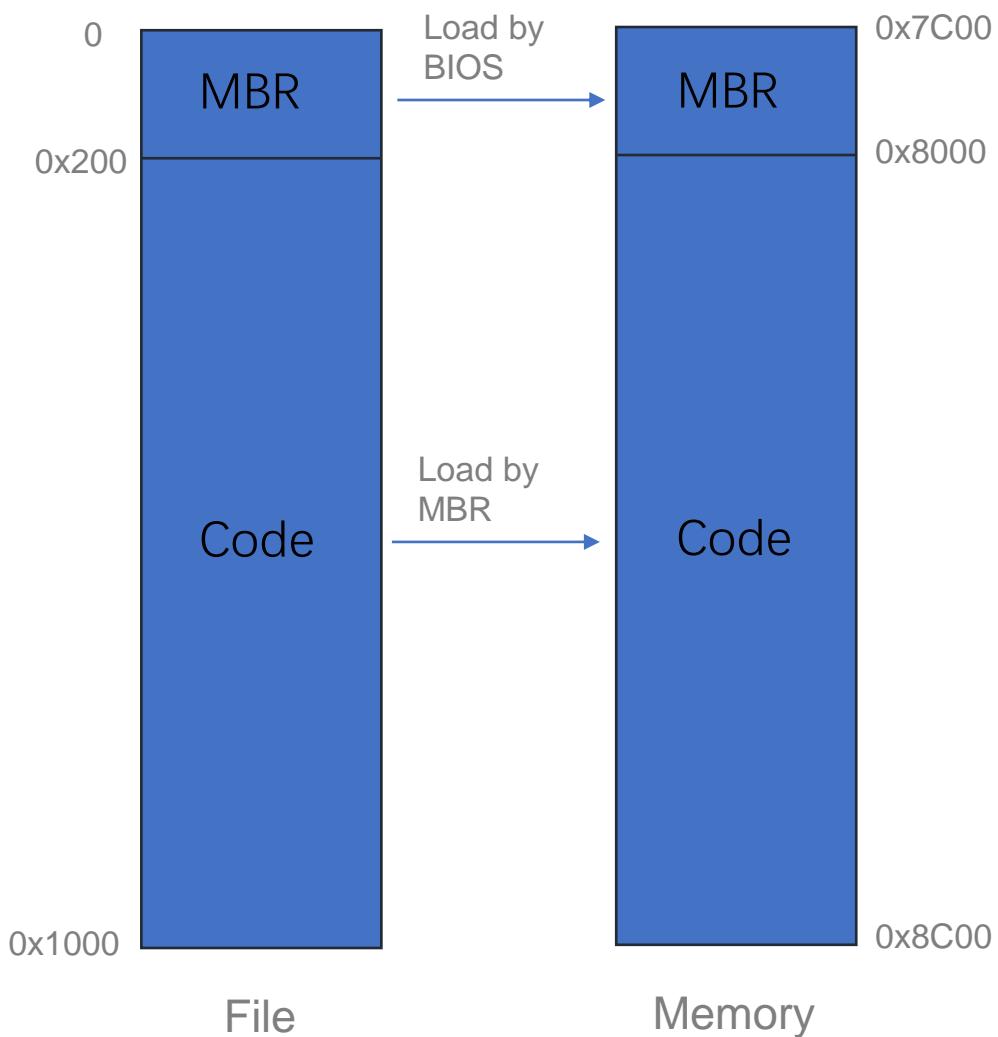
Sample Analysis: First Stage

- Like most operating systems, the program runs in two stage
 - MBR is responsible for loading code from file into 0x8000
 - Then it jumps to 0x8000 and execute the rest code

```
loc_7C00:          ; DATA XREF: seg000:7C09↓o
cli
xor  ax, ax
mov  ds, ax
mov  ss, ax
mov  es, ax
lea   sp, loc_7C00
sti
mov  eax, 20h ; 
mov  ds:byte_7C45, dl
mov  ebx, 1
mov  cx, 8000h
call sub_7C27
jmp  near ptr byte_7C4C+3B4h ; jump to 0x8000

; ===== S U B R O U T I N E =====

sub_7C27:          ; CODE XREF: seg000:7C21↑p
proc near
xor  eax, eax
mov  di, sp
push eax
push ebx          ; sectors offset = 1
push es
push (offset byte_7C4C+3B4h) ; destination address = 0x8000
push 7            ; sectors count = 7
push 10h
mov  si, sp
mov  dl, ds:byte_7C45
mov  ah, 42h ; 'B'
int  13h          ; DISK - IBM/MS Extension - EXTENDED READ
mov  sp, di
retn
endp
```



Sample Analysis: Second Stage

- The logic which starts from 0x8000 is pretty clear
 - Firstly, it gets current datetime by INT 1a and then xors the string at 0x8809 with that datetime
 - Then, it reads user input and xors the same string at 0x8809 with the input
 - Lastly, it initializes the screen and print the ascii art

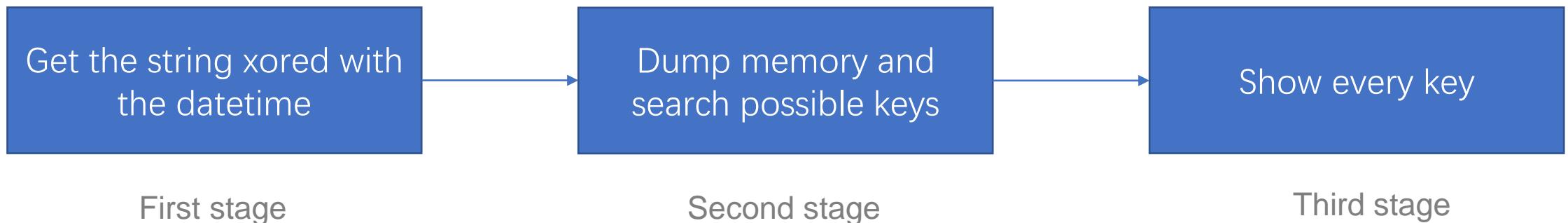
```
seg000    -- segment byte public 'CODE' use16
assume cs:seg000
;org 8000h
assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
call sub_805B
push ds:word_87F2
call get_date_and_write_to_87EE
push 4
push offset date_87EE
call xor8809
add sp, 4
push 87F4h
call read_input
add sp, 4
push ax
push 87F4h
call xor8809
add sp, 4
call initialize_screen
push 0
push 8809h
call print_ascii_art
add sp, 4
mov cx, 2607h
mov ah, 1
int 10h      ; - VIDEO - SET CURSOR CHARACTERISTICS
              ; CH bits 0-4 = start line for cursor in character cell
              ; bits 5-6 = blink attribute
              ; CL bits 0-4 = end line for cursor in character cell
loc_803D:    hlt
              ; CODE XREF: seg000:803E+j
;
jmp short loc_803D
```

Sample Analysis: Qiling's work

- What Qiling can do to speed up our analysis and find the key
 - Emulate the binary
 - Hook interrupts like INT 1a to give the program a specific time
 - Dynamic memory read/write API
 - Automatically test every key with partial execution and snapshot API
- The crack process can be divided into three stages
 - The key of this challenge is not unique, so we have to show every one

```
if __name__ == "__main__":
    ql = first_stage()
    # resume terminal
    curses.endwin()
    keys = second_stage(ql)
    for key in keys:
        print(f"Possible key: {key}")
    # The key of this challenge is not unique. The real
    # result depends on the last ascii art.
    print("Going to try every key.")
    time.sleep(3)
    third_stage(keys)
    # resume terminal
    curses.endwin()
```

- Hook API
- Partial execution
- Fs mapping
- Memory API
- Snapshot API



Crack: First stage

- The quick look before suggests that we have to set the datetime to Feburary 06, 1990
 - It's extremely easy to achieve that with `ql.set_api`
- The program also read disks directly
 - Use `ql.fs_mapper` API to emulate a disk
- Execute the program until 0x8018
 - At this time, the string at 0x8809 has been xored with date
- Dump memory at 0x8809 for the next stage
 - Use `ql.mem.read` API to dump memory

```
seg000:8000          call    sub_805B
seg000:8003          push    ds:word_87F2
seg000:8007          call    get_date_and_write_to_87EE
seg000:800A          push    4
seg000:800C          push    offset date_87EE
seg000:800F          call    xor8809
seg000:8012          add    sp, 4
seg000:8015          push    87F4h
seg000:8018          call    read_input
```

```
# In this stage, we get the encrypted data which xored with the specific date.
def first_stage():
    ql = Qiling(["rootfs/8086/doogie/doogie.bin"],
                "rootfs/8086",
                console=False,
                log_dir=".")
    ql.add_fs_mapper(0x80, QlDisk("rootfs/8086/doogie/doogie.bin", 0x80))
    # Doogie suggests that the datetime should be 1990-02-06.
    ql.set_api((0x1a, 4), set_required_datetime, QL_INTERCEPT.EXIT)
    # A workaround to stop the program.
    hk = ql.hook_code(stop, begin=0x8018, end=0x8018)
    ql.run()
    ql.hook_del(hk)
    return read_until_zero(ql, 0x8809)
```

February 06, 1990... Despite being a 16-year-old reverse engineering genius, I seem to have forgotten the password to my PC. Can you help me???

```
def set_required_datetime(ql: Qiling):
    ql.nprint("Setting Feburary 06, 1990")
    ql.reg.ch = BIN2BCD(19)
    ql.reg.cl = BIN2BCD(1990%100)
    ql.reg.dh = BIN2BCD(2)
    ql.reg.dl = BIN2BCD(6)
```

Crack: Second stage

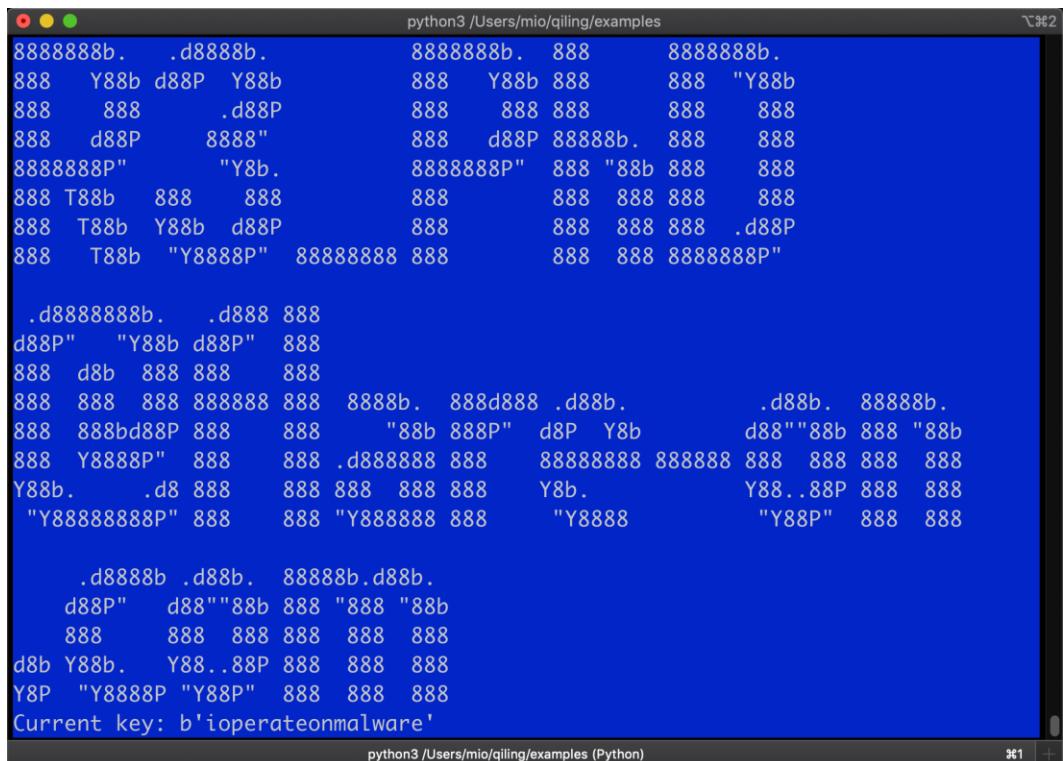
- Dump memory at 0x8809 for the next stage
 - Use 'ql.mem.read' API to dump memory
- Utilize some algorithms[1] to guess key size and search possible keys with the assumption that all the result should be printable ascii since it is likely an ascii art

```
# In this stage, we crack the encrypted buffer.
def second_stage(ql: Qiling):
    data = bytes(read_until_zero(ql, 0x8809))
    key_size = guess_key_size(data) # Should be 17
    seqs = []
    for i in range(key_size):
        seq = b""
        j = i
        while j < len(data):
            seq += bytes([data[j]])
            j += key_size
        seqs.append(seq)
    seqs_keys = cal_count_for_seqs(seqs)
    keys = search_possible_key(seqs, seqs_keys)
    return keys

def read_until_zero(ql: Qiling, addr):
    buf = b""
    ch = -1
    while ch != 0:
        ch = ql.mem.read(addr, 1)[0]
        buf += pack("B", ch)
        addr += 1
    return buf
```

Crack: Third stage

- Execute until 0x8018 and take a snapshot
- Fill in the key in the memory and Skip reading user input
- After completing one round, resume the program to previous snapshot and try next key
- One of the correct keys is: 'ioperateonmalware'



```
python3 /Users/mio/qiling/examples
8888888b. .d8888b. 8888888b. 888 8888888b.
888 Y88b d88P Y88b 888 Y88b 888 888 "Y88b
888 888 .d88P 888 888 888 888
888 d88P 8888" 888 d88P 888888b. 888 888
8888888P" "Y8b. 8888888P" 888 "88b 888 888
888 T88b 888 888 888 888 888 888
888 T88b Y88b d88P 888 888 888 888 .d88P
888 T88b "Y8888P" 88888888 888 888 88888888P"

.d8888888b. .d888 888
d88P" "Y88b d88P" 888
888 d8b 888 888
888 888 8888888 888 8888b. 888d888 .d88b. .d88b. 888888b.
888 888bd88P 888 888 "88b 888P" d8P Y8b d88"88b 888 "88b
888 Y8888P" 888 888 .d888888 888 88888888 8888888 888 888 888 888
Y88b. .d8 888 888 888 888 Y8b. Y88..88P 888 888
"Y88888888P" 888 888 "Y8888888 888 "Y8888
.d8888b .d88b. 88888b.d88b.
d88P" d88"88b 888 "888 "88b
888 888 888 888 888
d8b Y88b. Y88..88P 888 888 888
Y8P "Y8888P "Y88P" 888 888 888
Current key: b'ioperateonmalware'
```

```
def show_once(ql: Qiling, key):
    klen = len(key)
    ql.reg.ax = klen
    ql.mem.write(0x87F4, key)
    # Partial execution to skip input reading
    ql.run(begin=0x801B, end=0x803d)
    echo_key(ql, key)
    time.sleep(3)

# In this stage, we show every key.
def third_stage(keys):
    # To setup terminal again, we have to restart the whole program.
    ql = Qiling(["rootfs/8086/doogie/doogie.bin"],
                "rootfs/8086",
                console=False,
                log_dir="."))
    ql.add_fs_mapper(0x80, QlDisk("rootfs/8086/doogie/doogie.bin", 0x80))
    ql.set_api((0x1a, 4), set_required_datetime, QL_INTERCEPT.EXIT)
    hk = ql.hook_code(stop, begin=0x8018, end=0x8018)
    ql.run()
    ql.hook_del(hk)
    # Snapshot API.
    ctx = ql.save()
    for key in keys:
        show_once(ql, key)
        ql.restore(ctx)
```

Planning

Roadmap

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



Everything Else

>About Qiling Framework

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



Star us

qilingframework / qiling

Code Issues 32 Pull requests 4 Actions Projects Wiki Security Insights Settings

master 2 branches 10 tags Go to file Add file Code

xwings Merge pull request #532 from qilingframework/dev ... 7f27ec3 on Sep 30 3,445 commits

File	Description	Time
.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

About

Qiling Advanced Binary Emulation Framework

qiling.io

binary emulator framework
unicorn-emulator malware analysis
qiling reverse-engineering
cross-architecture uefi unicorn-engine

Readme

GPL-2.0 License

Releases 10

Version 1.1.3 Latest on Sep 30

+ 9 releases

Questions