Qiling Framework:
Learn how to build a fuzzer based on a 1day bug

HITB Lockdown 002, Virtual Lab
July, 2020

KaiJern LAU, kj -at- qiling.io
NGUYEN Anh Quynh, aquynh -at- gmail.com
huitao, CHEN null -at- qiling.io
TianZe DING, dliv3 -at- gmail.com
BoWen SUN, w1tcher.bupt -at- gmail.com
Tong YU, spikeinhouse -at- gmail.com

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

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 Training
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, Blackhat USA, Building IoT Fuzzer with Qiling
2020, Blackhat Singapore, Building Fuzzer with Qiling

Contributor from Dell, Intel, Fireeye and etc

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
1,100+ Github star, more than 3,000 pypi download, 40+ contributors worldwide

Contributor from Dell, Intel, Fireeye and etc
About Dliv3/w1tcher/Null/Sp1ke

Rest of the team members are from theshepherdlab, Dubhe CTF team & community
NGUYEN Anh Quynh

- Nanyang Technological University, Singapore
- PhD in Computer Science
- Operating System, Virtual Machine, Binary analysis, etc
- Usenix, ACM, IEEE, LNCS, etc
- Blackhat USA/EU/Asia, DEFCON, Recon, HackInTheBox, Syscan, etc
- Capstone disassembler: http://capstone-engine.org
- Unicorn emulator: http://unicorn-engine.org
- Keystone assembler: http://keystone-engine.org
Motivation
 Qiling framework
 Design & implementation
 Build dynamic analysis tools on top of Qiling Framework
 Hands On
What is IoT

- Camera
- Air-con
- TV
- FAN
- Heater
- Fridge
- Watch
- Lock
- Security
- Kitchen
- Phone
Traditional IoT Hacking
The Web Hacker

Exploits found on the INTERNET

This is live excerpt from our database. Available also using API.

<table>
<thead>
<tr>
<th>Edit</th>
<th>Date</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2019-08-25</td>
<td>D-Link DIR-600M Authentication Bypass</td>
<td>Metasploit</td>
</tr>
<tr>
<td></td>
<td>2019-06-01</td>
<td>D-Link WLE-AP XSS</td>
<td>DoS</td>
</tr>
<tr>
<td></td>
<td>2019-06-07</td>
<td>D-Link DVA-260AP</td>
<td>DoS</td>
</tr>
<tr>
<td></td>
<td>2019-06-11</td>
<td>D-Link L-530U</td>
<td>Cross Site Injection</td>
</tr>
<tr>
<td></td>
<td>2019-05-03</td>
<td>Keep1.0.2</td>
<td>FDP</td>
</tr>
<tr>
<td></td>
<td>2018-12-23</td>
<td>D-Link DSL-2750L</td>
<td>D-99</td>
</tr>
<tr>
<td></td>
<td>2018-12-23</td>
<td>D-Link DSL-2750L</td>
<td>D-99</td>
</tr>
<tr>
<td></td>
<td>2018-11-09</td>
<td>D-Link Central Firewallanager</td>
<td>CNAW 160.1.03</td>
</tr>
<tr>
<td></td>
<td>2018-11-09</td>
<td>D-Link Central Firewallanager</td>
<td>CNAW 160.1.03</td>
</tr>
<tr>
<td></td>
<td>2018-11-09</td>
<td>D-Link Central Firewallanager</td>
<td>CNAW 160.1.03</td>
</tr>
<tr>
<td></td>
<td>2018-10-10</td>
<td>D-Link IAP-1200</td>
<td>Cross Site Injection</td>
</tr>
<tr>
<td></td>
<td>2018-10-13</td>
<td>D-Link DSL-2847</td>
<td>Cross Site Injection</td>
</tr>
<tr>
<td></td>
<td>2018-09-06</td>
<td>D-Link D-850N</td>
<td>N-150</td>
</tr>
<tr>
<td></td>
<td>2018-08-03</td>
<td>D-Link DFL-615</td>
<td>Delivery of Service</td>
</tr>
<tr>
<td></td>
<td>2018-08-26</td>
<td>D-Link DSL-2735U Setup Wizard</td>
<td>Authentication</td>
</tr>
<tr>
<td></td>
<td>2018-08-24</td>
<td>D-Link EyeOS Baby Monitor DCG-200L</td>
<td>Remote Code Execution</td>
</tr>
<tr>
<td></td>
<td>2018-06-24</td>
<td>D-Link EyeOS Baby Monitor DCG-200L</td>
<td>Command Injection</td>
</tr>
<tr>
<td></td>
<td>2018-07-25</td>
<td>D-Link DAP-1500</td>
<td>Path Traversal</td>
</tr>
<tr>
<td></td>
<td>2018-07-03</td>
<td>D-Link DFL-615 A2 Improper Access Control</td>
<td>Published</td>
</tr>
<tr>
<td></td>
<td>2018-05-26</td>
<td>D-Link DSL-2750S</td>
<td>CSS Command Injection Metasploit</td>
</tr>
<tr>
<td></td>
<td>2018-05-26</td>
<td>D-Link DSL-2750S</td>
<td>CSS Command Injection</td>
</tr>
<tr>
<td></td>
<td>2018-05-26</td>
<td>D-Link DSL-2750S</td>
<td>CSS Command Injection</td>
</tr>
<tr>
<td></td>
<td>2018-04-17</td>
<td>D-Link DFL-615</td>
<td>Persistent Cross Site Infection</td>
</tr>
<tr>
<td></td>
<td>2018-03-31</td>
<td>D-Link CH-490XL</td>
<td>Writable ACL</td>
</tr>
<tr>
<td></td>
<td>2018-03-01</td>
<td>D-Link DCS-3600-107C</td>
<td>Cross Site Request Forgery</td>
</tr>
<tr>
<td></td>
<td>2018-01-15</td>
<td>D-Link DNS-340</td>
<td>ShareCenter</td>
</tr>
<tr>
<td></td>
<td>2018-01-15</td>
<td>D-Link DNS-320</td>
<td>ShareCenter</td>
</tr>
</tbody>
</table>
Firmware Hacking

```plaintext
<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>HEXADECIMAL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>218040</td>
<td>0x353B8</td>
<td>CRC32 polynomial table, little endian</td>
</tr>
<tr>
<td>524288</td>
<td>0x80000</td>
<td>uImage header, header size: 64 bytes, header CRC: 0x4687D1AC, created: 2007-06-15 10:36:26, image size: 2217656 bytes, Data Address: 0x20000000, Entry Point: 0x800040, data CRC: 0xA54D09E1, OS: Linux, CPU: ARM, image type: OS Kernel Image, compression type: none, image name: &quot;gm8136&quot;</td>
</tr>
<tr>
<td>524352</td>
<td>0x80040</td>
<td>Linux kernel ARM boot executable zImage (little-endian)</td>
</tr>
<tr>
<td>542452</td>
<td>0x846F4</td>
<td>gzip compressed data, maximum compression, from Unix, last modified: 1970-01-01 00:00:00 (null date)</td>
</tr>
<tr>
<td>3670112</td>
<td>0x380060</td>
<td>xz compressed data</td>
</tr>
<tr>
<td>3800908</td>
<td>0x39FF4C</td>
<td>xz compressed data</td>
</tr>
<tr>
<td>3931872</td>
<td>0x3BFE0</td>
<td>xz compressed data</td>
</tr>
<tr>
<td>4979008</td>
<td>0x4BF94C</td>
<td>xz compressed data</td>
</tr>
</tbody>
</table>
```

**extract the front and back parts of the file system and repack the file system**

```plaintext
dd if=/dev/rdisk0s5 20170912 140739.BIN bs=3678016 count=1 of=part1.bin
dd if=/dev/rdisk0s5 20170912 140739.BIN bs=1153436 skip=1 of=part2.bin
mkfsqfs squashfs-root squashfs-customize.bin -comp xz
```
Hardware Hacking
Additional Note: What To Buy
Case Study
Buying a China Only Cam
Talking Cam’s Warming

“Not Allow To Use Outside China”
Answer from Google and Baidu
Recently I bought a Xiaomi Xiao Yi (IP) camera (also known as Yi Home). The Chinese version. The camera looks nice, the picture quality is ok, and worked fine on my local Wi-Fi.

However, I was unfortunate enough to receive and test the camera when Xiaomi decided to deny access from the iOS app to the camera outside of China (error 5400). I was hoping a firmware upgrade would solve this issue so I have upgraded it from 1.8.5.11 to 1.8.6.18. Now my camera was useless. The camera would say “This camera can only be used in China” and would shut down.

This was the tipping point when I have decided I will investigate what’s happening with this camera and what can be done to make it functional again.

At the time of writing the remote access error (5400) has been solved by the provider so no additional action is required. I tried to convert a Chinese camera to international one by changing the serial of the device, but couldn’t test from a European or US IP and probably I would have needed access to the system files of a functional international camera to compare.

So the remaining issue was the camera shut down with the latest firmware (tested with 1.8.6.1A and 1.8.6.1B).

If you do a search there are heaps of websites describing how you can gain access to the camera and ultimately enable remote access via telnet. I won’t get into those details, you can check some of the websites I listed below.

Once you logged into the camera via telnet the fun part begins. The camera is running a Linux version.
Try to Connect to USB TTL

- Power
- USB TTL
- No Way To Get Near USB TTL
Solving Puzzle

- Finding GND
- Guessing RX TX
- Multi meter
What To We Want To Archive

- Work without Xiaomi app
- Turn on WiFi while Boot
- Turn on telnet while boot
- Turn on ftp while boot
- Turn RTSP whole boot

**Network settings**

```
/etc/init.d # cat /home/conf/wpa_supplicant.conf

ctrl_interface=/var/run/wpa_supplicant
ap_scan=1
network=
    ssid="MY_WIFI_L4H"
    scan_ssid=1
    proto=WPA RSN
    key_mgmt=WPA-PSK
    pairwise=CCMP TKIP
    group=CCMP TKIP
    psk="my_PASSWORD_L4h"
```
Enabling Services

Bring up some services

```
/etc/init.d # cat S88telnet

#!/bin/sh
/home/app/telnetd &
(sleep 10; /home/base/tools/wpa_supplicant -i wlan0 -c /home/conf/wpa_supplicant.conf) &
(sleep 20; /sbin/ifconfig wlan0 192.168.0.100 netmask 255.255.255.0) &
```

```
/etc/init.d # cat S89ftp

#!/bin/sh
/home/app/tcpsvd -V 0.0.0.0 21 ftpd -w &
```

RTSP returns segmentation fault

Fire up IDA pro and look at the RTSP Binary, we found few files required before it can run, so this is how we fix it.

```
l -s /tmp/hd1 /home/hd1
l -s /tmp/hd2 /home/hd2
l -s /tmp /home/mmap tmpfs
mkdir /home/jrview
l -s /home/app/busybox /bin/renice
l -s /home/lib/libcrypt-0.9.32.1.so
l -s /home/lib/libcrypt.so.0
l -s /home/lib/libstdc++-v3.so.6
```

Forgotten to mount FS after boot
sdcard is not readable while boot
Analyzing The Actual Firmware

XiaoYI Ants unofficial info page

Firmwares
Hardware version v2.1 needs a firmware version 1.8.5.1K or higher!
You can find the how to on the firmware flash instruction page.
Note: flash firmware is at your own risk!

Original for CN hardware
1.8.5.1K_201502111014
1.8.5.1K_201502021024
1.8.5.1K_201503111331
1.8.5.1K_201506291725
1.8.5.1K_201511201815
1.8.5.1K_201512212009
1.8.6.1A_201602241619
1.8.6.1B_20160318307

Original for international hardware
1.8.5.1K_201601071325

Modified for CN hardware
Additional features are added to this firmwares (RTSP, FTP, telnet, timezone, ...) How to use the different additional features is described on the Instruction page.
1.8.5.1E_rtp
1.8.5.1f_sanimation
1.8.5.1K_rtpfix_v3
1.8.5.1L_rtpfix_v3
1.8.5.1M_rtpfix_v4
1.8.6.1B_rtpfix

Branch: master  yi-hack-v3 / src /

- shadow-1 Fixed errors in startup scripts.
- busybox Added ability to randomly select the number of proxy servers to downl...
- home/yi-hack-v3 Fixed errors in startup scripts.
- libwebsockets-plugins Firmware no longer affected by Xiaomi updates.
- libwebsockets Firmware no longer affected by Xiaomi updates.
- proxychains-ng Firmware no longer affected by Xiaomi updates.
- rootfs/etc Fixed errors in startup scripts.
- uClibc Initial tested version of the firmware for Yi 1080p Dome camera.
brd: module loaded
Check Flash Memory Controller v100 ... Found.
SPI Nor(cs 0) ID: 0xc8 0x40 0x18
Block:64KB Chip:16MB Name:"GD25Q128"
SPI Nor total size: 16MB
8 cmdlinepart partitions found on MTD device hi_sfc
8 cmdlinepart partitions found on MTD device hi_sfc
Creating 8 MTD partitions on "hi_sfc":
0x00000000000-0x000000040000 : "boot"
0x00000004000-0x000000050000 : "env"
0x00000005000-0x000000060000 : "conf"
0x00000006000-0x0000001f0000 : "os"
0x0000001f000-0x000000330000 : "rootfs"
0x00000033000-0x000000fe0000 : "home"
0x000000fe000-0x000000ff0000 : "vdi"
0x000000ff000-0x000001000000 : "ver"
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
hiusb-ehci hiusb-ehci.0: HIUSB EHCI
hiusb-ehci hiusb-ehci.0: new USB bus registered, assigned bus number 1
hiusb-ehci hiusb-ehci.0: irq 15, io mem 0x100b0000
hiusb-ehci hiusb-ehci.0: USB 0.0 started, EHCI 1.00
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
i2c /dev entries driver
hisi_i2c hisi_i2c.0: Hisilicon [i2c-0] probed!
hisi_i2c hisi_i2c.1: Hisilicon [i2c-1] probed!
hisi_i2c hisi_i2c.2: Hisilicon [i2c-2] probed!
Dumping The Firmware

- Making sure the firmware is the same with the one on the internet
Debug and Patch
Taking Partition Notes

Partition by size, take from the boot log

```
0x000000000000-0x000000040000 : "boot"
0x000000040000-0x000000050000 : "env"
0x000000050000-0x000000060000 : "conf"
0x000000060000-0x0000010000 : "os"
0x0000010000-0x0000030000 : "rootfs"
0x0000030000-0x00000f0000 : "home"
0x00000f0000-0x0000100000 : "vd1"
0x0000100000-0x0000100000 : "ver"
```

Dump using bus pirate

```
flashrom -p buspirate_spi:dev=/dev/ttyUSB0 -c GD25Q128C -r yicam_night_GD25Q128C.bin -V -f
```

Spliting the image

This is how you split the file according to partition size

```
dd if=yicam_night_test_GD25Q128C.bin of=yicam_night_test_GD25Q128C_bootloader.bin bs=1 count=$(0x040000)
ss

dd if=yicam_night_test_GD25Q128C.bin of=yicam_night_test_GD25Q128C_env.bin bs=1 count=$(0x050000-0x040000) skip=$(0x040000)
```

```
TL;DR

Here’s a quick overview of the entire mounting process:

1. Extract the JFFS2 file system image from the U-Boot image:
   ```
   uImage.py -x home
   ```
2. Pad the JFFS2 image to make it work with block2mtd:
   ```
   ./jffs2.py --pad=0 7518-hi3518-home
   ```
3. Load the kernel modules:
   ```
   modprobe block2mtd mtdblock
   ```
4. Setup the loopback device:
   ```
   losetup /dev/loop0 7518-hi3518-home
   ```
5. Associate loopback device with MTD device
6. Mount the MTD device (finally)

If all this seems tedious, I wrote a `mount-jffs2` shell script that performs steps 3 to 6. You just need to specify the (padded) image file, mount point and block size:

   ```
   ./mount-jffs2 7518-hi3518-home /mnt/image 64KiB
   ```
Making The Firmware

```
qemu-img create test.img 1024M
mkfs.ext2 -F test.img
mount -t ext2 -o loop,rw test.img /mnt

[23:52:06]:xwings@kali32:$</yicam_home_720p/yi-hack-v3/rootfs_mount>
ls -aFl
total 60
drwxr-xr-x 15 root root 4096 Jan 1 1970 ./
drwxr-xr-x  5 xwings xwings 4096 Aug 15 23:11 ../
drwxr-xr-x  2 root root 4096 Jul  2 22:34 bin/
drwxr-xr-x  2 root root 4096 Jul  2 22:24 dev/
drwxr-xr-x  4 root root 4096 Jul  2 22:24 etc/
drwxr-xr-x  2 root root 4096 Jul  2 22:24 home/
drwxr-xr-x  2 root root 4096 Jul  2 22:24 lib/
lrwxrwxrwx  1 root root  11 Jul  2 22:34 linuxrc -> bin/busybox*
drwxr-xr-x  3 root root 4096 Jul  2 22:24 mnt/
drwxr-xr-x  2 root root 4096 Jul  2 22:24 proc/
drwxr-xr-x  2 root root 4096 Jul  2 22:24 root/
drwxr-xr-x  2 root root 4096 Jul  2 22:34 sbin/
drwxr-xr-x  2 root root 4096 Jul  2 22:24 sys/
drwxr-xr-x  2 root root 4096 Jul  2 22:24 tmp/
drwxr-xr-x  4 root root 4096 Jul  2 22:34 usr/
drwxr-xr-x  3 root root 4096 Jul  2 22:24 var/

(23:52:08):xwings@kali32:$</yicam_home_720p/yi-hack-v3/rootfs_mount>
```

- # qemu-img create test.img 1024M
- # mkfs.ext2 -F test.img
- # mount -t ext2 -o loop,rw test.img /mnt/test
- Copy all files
- umount
Test Booting with QEMU

Mount, Edit and Pad

Look for JFFS mounting tutorial, make all the changes you need. In case you need padding before merging the ROM,

```
ruby -e 'print "\xFF" * 393216' >> rootfs_e.jffs
```

Merging the ROM

```
(dd if=yicam_night_test_GD25Q128C_bootloader.bin ) > yicam_full_e.bin
(dd if=yicam_night_test_GD25Q128C_env.bin ) >> yicam_full_e.bin
(dd if=yicam_night_test_GD25Q128C_conf.bin ) >> yicam_full_e.bin
(dd if=yicam_night_test_GD25Q128C_os.bin ) >> yicam_full_e.bin
(dd if=yicam_night_test_GD25Q128C_rootfs_e.bin ) >> yicam_full_e.bin
(dd if=yicam_night_test_GD25Q128C_home.bin ) >> yicam_full_e.bin
(dd if=yicam_night_test_GD25Q128C_vd1.bin ) >> yicam_full_e.bin
(dd if=yicam_night_test_GD25Q128C_ver.bin ) >> yicam_full_e.bin
```
Flashing Back The Firmware
Getting Firmware
Firmware and Hardware

Extract From Flash, Extract From APK, Traffic Sniffing or Just Download
Technically 1. Download 2. Patch with Backdoor 3. Flash 4. pwned

If we need more?
1. RCE 2. Fuzz
Work Around
Complete Kit to Success

MIPS
How Many Dev Board

ARM
Classic LIBC Issue

AARCH64

Hardware is not “down gradable”
Assembly Instruction Compatibility

ARM

AARCH64
Why Firmware Emulation
More Resources = More Power

Processor
Normally 1-2 Core

RAM
Normally 256MB/512MB

FLASH
Normally 8MB/16MB/32MB/256MB

Most Important, we got apt-get
Objectives
Only One Process with Interaction

Hunt for the one that spawn listener port

most of the devices comes with one big binary
Boot
Distro and Kernel Mix and Match

**Argument:** running new or old distro + kernel

**Script to boot arm**
```
#!/bin/bash
sudo tuntctl -d tap0
sudo screen -d /opt/qemu/bin/qemu-system-arm -n 2048 -M virt -cpu cortex-a15 -smp cpus=4,maxcpus=4 -kernel boot.stretch.armhf/vmlinuz-4.9.0-4.4-arm64-lpea -initrd boot.stretch.armhf/vmimage-4.9.0-4.4-arm64-lpea -device virtio-blk-device,devname=/dev/sda2 -netdev type=tap,devname=eth0 -device virtio-netdev,name=netdevice0 -macaddr=52:54:00:ea:00:00 -nodev -disp缺口no =no -graphic
sudo sysctl -w net.ipv4.ip_forward=1
```  
```
echo "Stopping firewall and allowing everyone..."
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT
```

**Script to boot mips**
```
#!/bin/bash
sudo tunctl -d tap0
sudo ifconfig tap0 10.253.253.254 netmask 255.255.255.0
sudo sysctl -w net.ipv4.ip_forward=1
```  
```
echo "Stopping firewall and allowing everyone..."
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT
```

---

**Script to boot arm**
```
sudo screen -d /opt/qemu/bin/qemu-system-mipsel -n 512 -M malta -kernel boot.stretch.mipsel/vmlinuz-4.9.0-4.4-kc-malta -initrd boot.stretch.mipsel/initrd.img-4.9.0-4.4-kc-malta -append "root=/dev/mmcblk0p1;bootchart=simple" -net nic -net tap,ifname=tap0,script=no,downscript=no -net nic -net tap,ifname=tap1,script=no,downscript=no -p "script=" -nographic
```  
```
sudo tuntctl -d tap0
```  
```
sudo ifconfig tap0 10.253.253.254 netmask 255.255.255.0
```  
```
sudo sysctl -w net.ipv4.ip_forward=1
```  
```
echo "Stopping firewall and allowing everyone..."
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT
```

**Script to boot mips**
```
sudo screen -d /opt/qemu/bin/qemu-system-mipsel -n 512 -M malta -kernel boot.stretch.mipsel/vmlinuz-4.9.0-4.4-kc-malta -initrd boot.stretch.mipsel/initrd.img-4.9.0-4.4-kc-malta -append "root=/dev/mmcblk0p1;bootchart=simple" -net nic -net tap,ifname=tap0,script=no,downscript=no -net nic -net tap,ifname=tap1,script=no,downscript=no -p "script=" -nographic
```  
```
sudo tuntctl -d tap0
```  
```
sudo ifconfig tap0 10.253.253.254 netmask 255.255.255.0
```  
```
sudo sysctl -w net.ipv4.ip_forward=1
```  
```
echo "Stopping firewall and allowing everyone..."
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT
```
chroot
chroot is easy (still hardware dependent), but we will have issue with tools
Stage 0 Issue: File Not Found
The File Missing Trick

We Missed You

```c
chdir("/")
execve("/bin/bash", ["/bin/bash", ",i"], 0xffffca14f650 /* 18 vars */) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/aarch64-linux-gnu/charset.alias", O_RDONLY|O_NOFOLLOW) = -1 ENOENT (No such file or directory)
write(2, "chroot: ", 8chroot: ) = 8
write(2, "failed to run command '/bin/bash', failed to run command '/bin/bash' = 33
write(2, ": No such file or directory", 27: No such file or directory) = 27
write(2, "\n", 1 ) = 1
close(1) = 0
close(2) = 0
exit_group(127) = ?
```

We found you

```
root@rpi3:/opt/
```

```bash
ln -s /lib64/ld-linux-aarch64.so.1 /lib64
```

```
lib64/ld-linux-aarch64.so.1: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter 
/lib64/ld-linux-aarch64.so.1, for GNU/Linux 3.14.0, BuildID[sha1]=22e2854c58b1814825b95c9b103ac658cd371f5b0, stripped
```
Stage 1 Issue: .SO Not Found
Out from chroot, we need feeding

Feeding all the required so and binary with "ln –s"
Out from chroot, we need feeding

“segfault” without clear error. strace come to rescue
NVram
Dark side of NVRAM

Relationship between main binary is so intimate, but in actual fact. Is just a hit and run.
Dark Side of NVRAM

Relationship between main binary is so intimate, but in actual fact, it's just a hit and run.

Dark Side of the main process, we ignore and can't move to the next step.
A fake NVRAM

IF interactor is the medium, can we fake it?

reply with nvram info
A fake NVRAM

IF interactor is the medium, can we fake it?

main process

ask for nvram info

reply with nvram info

Custom Interactor
The bridge trick

The switch looking device
Wireless Devices
Faking wpa_supplicant

making eth0 looks like wlan0 works too
Every Thing Else Fail
Argument: To Patch or To Fulfill Firmware Needs
Motivations
More Resources = More Power

Processor
Normally 1-2 Core

MAX RAM
Normally 256MB/512MB

MAX Space

Or We Can Just X86 IT

FLASH
Normally 8MB/16MB/32MB/256MB
What is Required

Debugger or Disassembler

*BSD  Linux  MacOS  Windows

MIPS  ARM  AARCH64  X86
Why Not Off The Shelf Emulator

1960s - IBM released hypervisors on CP-40 and CP-67 operating systems [2]

2001 - IBM launched a hypervisor for midrange UNIX systems [2]

1999 - VMware introduced virtualization to the x86 platform with VMware Workstation 1.0 [3][4]

2001 - VMware released ESX Server 1.0 [5]


Many more vendors and enhanced virtualization solutions are released.

More Emulate = Higher Chances Being Detected
Unicorn Emulator framework

- Multi-architectures: Arm, Arm64, M68K, Mips, Sparc, & X86 (include X86_64)
- Native support for Windows & *nix (with Mac OSX, Linux, *BSD & Solaris confirmed)
- Clean/simple/lightweight/intuitive architecture-neutral API
- Implemented in pure C language, with multiple bindings
- High performance by using Just-In-Time compiler technique
- Support fine-grained instrumentation at various levels

Limitation

- Just emulator for low level instructions + memory access
- No higher level concepts of Operating System
  - File format
  - Library
  - Filesystem
  - Systemcall
  - OS structures
Features

- Cross platform: Windows, MacOS, Linux, BSD
- Cross architecture: X86, X86_64, Arm, Arm64, Mips
- Multiple file formats: PE, MachO, ELF, UEFI(PE)
- 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
- Full GDB/IDA/r2 Support
- OS profiling support
## User Mode Emulation

<table>
<thead>
<tr>
<th>TOOL</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 (No *NIX)  
- 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 |
| WSL/2     | - Limited ARCH Support  
- Only Linux and run in Windows  
- Not Sandboxed, It linked to /mnt/c  
- No Instrumentation (maybe) |
| Zelos     | - Very good project!  
- It’s a Framework!  
- Linux based only (No Windows)  
- Incomplete support for Linux multi arch |
How Qiling Works
How Does It Work

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

```python
class ELFParser:
    def __init__(self, path, ql):
        self.path = path
        self.ql = ql

        with open(path, "rb") as f:
            self.elfdata = f.read()

        self.elfident = self.getident()

        if self.elfident[ : 4] != b'\x7f\x45\x46\x47':
            ql.errprint("\x7f\x45\x46\x47: ERROR: NOT a ELF")
            exit()

        if self.elfident[ : 4] == b'\x00\x00\x00\x00':
            self.is32bit = True
        else:
            self.is32bit = False

        if self.elfident[ : 4] == b'\x00\x00\x00\x08':
            self.ise64bit = True
        else:
            self.ise64bit = False

        if self.elfident[ : 4] == b'\x00\x00\x00\x08':
            self.endian = 1
        elif self.elfident[ : 4] == b'\x00\x00\x00\x02':
            self.endian = 2

        def load00shellecode(self, dls):
            self.initElf()
            self.initElfPE()
            self.initElfData()
            for each in dls:
                self.loadDLEach()

        def loadPE32(self):
            self.pe = profile.PE(self.path, fast_load=True)

            # for simplicity, no image base relocation
            self.ql.PE_IMAGE_BASE = self.pe.OPTIONAL_HEADER.ImageBase
            self.ql.PE_IMAGE_SIZE = self.pe.OPTIONAL_HEADER.SizeOfImage
            self.ql.entry_point = self.pe.OPTIONAL_HEADER.AddressOfEntryPoint
            self.ql.print("loading %s to 0x%x" % (self.path, self.PE_IMAGE_BASE))
```

Parse != Loader
Posix Series - Syscall Emulator

```python
def q1 syscall read(ql, uc, read_fd, read_buf, read_len, null0, null1, null2):
    path = (ql read_string(ql, uc, read_buf))
    if read_fd < 256 and ql file desc[read fd] != 0:
        try:
            if isinstance(ql file desc[read fd], socket.socket):
                data = ql file desc[read fd] recv(read_len)
            else:
                data = ql file desc[read fd] read(read_len)
            uc mem write(read_buf, data)
            ql print(">>> Read Completed %d x % path")
            return len(data)
        except:
            return -1
    ql print(">>> Read FD, %0x, %0x) = %d x % (read_fd, read_buf, read_len, return)
    return -1

def q1 syscall lseek(ql, uc, lseek_fd, lseek_offset, lseek_origin, null0, null1, null2):
    if lseek_fd >= 256:
        return -1
    ql print(">>> lseek(%, %0x, %0x) = %d x % (lseek_fd, lseek_offset, lseek_origin, return)
    return 0

def q1 syscall brk(ql, uc, brk_input, null0, null1, null2):
    if brk_input == 0:
        if brk_input >= ql brk address:
            ql mem access(brk address, (int((brk input + 0xff) // 0x1000) * 0x1000 - brk address))
        else:
            brk_input = ql brk address
        return 0
    return 0

def q1 syscall mprotect(ql, uc, mprotect_start, mprotect_len, mprotect_prot, null0, null1, null2):
    return 0
```

Syscall almost the same for OSX/Linux/*BSD
Kernel Programming 101
Emulate Syscall
Skip/Forward or Emulate Code
Prepare Execution Report
It took us some time to fix the GDT and Set Thread Area.
ARM/64 Series

ARM/Thumb and ARM64

Making Sure Loader is compatible

ARM MCR instruction for Set TLS

ARM Kernel Initialization

ARM and ARM64 Enable VFP
MIPS32EL Series

MIPS Comes with CO Processor

Configuration needed for CO Processor

Unicorn does not support Floating Point

Patch Unicorn to Support CO Processors

Custom Binary Injected for Set Thread Area
Applications of Qiling
Build dynamic analysis tools – Basic ++

- Let Qiling loads the binary (loading + dynamic linking)
- Syscall & system API logging available, provided by default
- Program callbacks with Qiling hook capabilities: hook memory access, hook address range
- Repeat in a loop: run() → analysis → resume()

```python
from unicorn import *
from capstone import *
from qiling import *

md = Cs(CS_ARCH_X86, CS_MODE_64)

def print_asm(ql, address, size):
    buf = ql.uc.mem_read(address, size)
    for i in md.disasm(buf, address):
        print("{:0x}:	{}	{}	{}
```

```python
if __name__ == "__main__":
    ql = Qiling(["rootfs/x8664_linux/bin/x8664_hello"], "rootfs/x8664_linux")
    ql.hook_code(print_asm)
    ql.run()
```
Debugger – GDB / IDAPro/ r2
Guided fuzzer – cross platform/architecture

- Cross platform/architecture: Windows, MacOS, Linux, BSD on X86, Arm, Arm64, Mips
- https://github.com/qilingframework/qiling/tree/dev/examples/fuzzing
Firmware analysis

- Emulation offers a chance to move analysis to a much more powerful platform
- Emulate a single binary is better than whole firmware
  - Hardware emulation is tough without hardware specs
  - Series of different firmware can share the same target binary
- Challenges
  - Dump firmware, or extract firmware from binary blob
  - Extract the target binary
  - NVRAM emulation
  - Dependency libraries
  - Presence of other devices: wireless interface
Demo Setup

VirtualBox or VMware
from qiling import *

def run_sandbox(path, rootfs, ostype, output):
    ql = Qiling(path, rootfs, ostype=ostype, output=output)
    ql.run()

if __name__ == '__main__':
    run_sandbox(["rootfs/arm_linux/bin/arm32-hello-static"], "rootfs/arm_linux", "linux", "debug")
```python
def run_one_round(payload):
    stdin = MyPipe()
    q = Qiling(["/rootfs/x86_linux/bin/crackme_linux", "/rootfs/x86_linux", output = "off"], stdin = stdin, stdout = sys.stdout, ins_count = [0]
    q.hook_code(instruction_count, ins_count)
    stdin.write(payload)
    q.run()
    del stdin
    del q
    return ins_count[0]

def solve():
    idx_list = [1, 4, 2, 0, 3]
    flag = b'\x00\x00\x00\x00\x00\x00\n'
    old_count = run_one_round(flag)
    for idx in idx_list:
        for i in b'0123456789abcdefgijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abedefghijklmph
        if tmp > old_count:
            old_count = tmp
        break
        # if idx == 2:
        #    break
    print(flag)

if __name__ == "__main__":
    solve()
```

Brute Forcer
Qiling: Hands On Time
Training Setup

- **Required OS**
  - Ubuntu 18.04 / 20.04
  - WSL2

- **Installation**
  - `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 libpixman1-dev clang python3-setuptools llvm`
  - `pip3 install qiling OR git clone git@github.com:qilingframework/qiling.git`

- **Install AFL++**
  - `git clone https://github.com/AFLplusplus/AFLplusplus.git`
  - `cd AFLplusplus`
  - `make`
  - `cd unicorn_mode`
  - `./build_unicorn_support.sh`
Emulate a Router

https://github.com/zsjevilhex/iot/tree/master/route/tenda
Device Emulation

- Read and write emulation for /dev/<devices>
- Able to input custom feedback towards Qiling

Third Party NVRAM

- Emulate Unix Domain Socket Connections
- Emulate ENV Input
Firmware Fuzzing

- Fuzzing DIR-815
- https://www.exploit-db.com/exploits/33863
- https://drive.google.com/file/d/10f3cqObsyZ_GHFy0DM-9d1VdsKCVhYjS/view?usp=sharing
What Else
A sanitized heap recently got merged into @qiling.io. Enable it to detect pool overflows, underflows, out-of-bounds reads, use-after-free bugs, and double or totally invalid free. Special thanks to @domenuk, @prôme, and the rest of the BaseSAFE guys for the ideas and motivation.

Qiling Framework
@qiling.io

After hijacking syscall and Windows API, Qiling now can hijack ".so" functions! You can set callback on function enter/exit, or replace entire function.

Checkout: docs.qiling.io/en/latest/hija...

11:51 AM · Jun 12, 2020 · Twitter Web App

Step after step, we are almost done @qiling.io

Utilizing symbolic execution in UEFI drivers to get to a vulnerable path. Using @qb_triton and @qiling.io Thanks to @assaf_carlsbad and @JonathanSalwan for their help.

12:28 AM · Apr 23, 2020 · Twitter Web App
Call for sponsor for development of Unicorn 2

- Current Unicorn is based on Qemu 2.1.2, from 2015
- Planning for **Unicorn 2**, based on new Qemu (5+)
- Some new exciting APIs in planning
- [https://github.com/unicorn-engine/unicorn/issues/1217](https://github.com/unicorn-engine/unicorn/issues/1217)

NGUYEN Anh Quynh, aquynh -at- gmail.com, @unicorn_engine