RANDORISEC

MOBILE HACKING

PWN – Debugging & Fuzzing

PWN
CHEATSHEET
V0.1

android

MAIN STEPS

- Local / remote debugging
- Fuzzing
- Modern protections

USEFUL RESOURCES

- AFL++ fuzzing project with tutorials
https://github.com/AFLplusplus/AFLplusplus
- QEMU emulator
https://gitlab.com/qemu-project/qemu
- ARM resources
https://developer.arm.com/

TOOLS

- GDB
- LLDB
- AFL++
- ADB
- QEMU
- PatchELF

GDB – Basics

Show memory

QWORD hex format
```
gdb # x/ngx <address|variable>
```
DWORD hex format
```
gdb # x/nwx <address|variable>
```
BYTE hex format
```
gdb # x/nbx <address|variable>
```
8-bit string format
```
gdb # x/ns <address|variable>
```
16-bit string format
```
gdb # x/nhs <address|variable>
```

Show instructions

Print n instructions
```
gdb # x/ni <address|variable>
```
Disassemble n bytes of memory
```
gdb # disass <address|variable>,+n
```

Process

Attach to a process with PID
```
gdb # attach <pid>
```
Attach to a remote target
```
gdb # target remote <ip>:<port>
```
Change root folder location
```
gdb # set sysroot /path/to/root
```

Watchpoints

List watchpoints
```
gdb # info watchpoints
```
Enabled when an address is read or written
```
gdb # awatch <address|variable>
```
Enabled when an address content is modified
```
gdb # watch <address|variable>
```
Enabled when an address is read
```
gdb # rwatch <address|variable>
```

Breakpoints

Add a breakpoint
```
gdb # b *address
```
Enable hardware breakpoints if available
```
gdb # set can-use-hw-watchpoints
```
List all breakpoints
```
gdb # info b
```
Disable a breakpoint
```
gdb # dis <bp_number>
```
Delete a breakpoint
```
gdb # del <bp_number>
```
Continue execution after a breakpoint hit
```
gdb # continue
```

Threads

List all threads
```
gdb # info thread
```
Change the current thread
```
gdb # thread <thread_num>
```

Fork

Follow child or parent process execution
```
gdb # set follow-fork-mode child|parent
```
List all inferiors
```
gdb # info inferiors
```
Change the current inferior
```
gdb # inferior <inferior_num>
```

Mapping

Show /proc/self/maps content
```
gdb # info proc maps
```
Show sections detail
```
gdb # maintenance info sections
```
Show target informations
```
gdb # info files
```

Remote ADB debugging

Push a debugging server on the device. Either lldb-server for recent version of the Android NDK, or a gdbserver
```
host$> adb push $NDK/toolchains/llvm/prebuilt/linux-x86_64/lib64/clang/14.0.7/lib/linux/x86_64/lldb-server /data/local/tmp
host$> adb shell chmod +x /data/local/tmp/lldb-server
```
or
```
host$> adb push $NDK/prebuilt/android-arm64/gdbserver/gdbserver /data/local/tmp
host$> adb shell chmod +x /data/local/tmp/gdbserver
```
Attach the debugging server to a running process
```
host$ adb shell /data/local/tmp/lldb-server g --attach <PID> :31337
```
or
```
host$ adb shell /data/local/tmp/gdbserver --attach :31337 <PID>
```
Forward the port onto the host
```
host$> adb forward tcp:31337 tcp:31337
```
Connect to the remote target
```
host$> lldb <binary> --one-line "gdb-remote 127.0.0.1:31337"
```
or
```
host$ gdb <binary> -ex "target remote :31337"
```

GDB – Record execution

Enable process instructions record if available
```
gdb # record full
```
Stop process record
```
gdb # record stop
```
Run program backward
```
gdb # rc
```
Run program backward until last instruction
```
gdb # reverse-step
```
Change manually direction of execution
```
gdb # set exec-direction reverse|forward
```

PatchELF

Replace a library by a new one
```
$ patchelf --replace-needed /lib/old.so /lib/new.so <binary>
```
Change the interpreter used
```
$ patchelf --set-interpreter /path/to/desired/ld.so <binary>
```
Add a new library
```
$ patchelf --add-needed <lib.so> <binary>
```

RANDORISEC

MOBILE HACKING

PWN – Debugging & Fuzzing

PWN CHEATSHEET V0.1

android

MAIN STEPS

- Local / remote debugging
- Fuzzing
- Modern protections

USEFUL RESOURCES

- AFL++ fuzzing project with tutorials
https://github.com/AFLplusplus/AFLplusplus
- QEMU emulator
https://gitlab.com/qemu-project/qemu
- ARM resources
https://developer.arm.com/

TOOLS

- GDB
- LLDB
- AFL++
- ADB
- QEMU
- PatchELF

Fuzzing closed source binary with AFL++ in QEMU mode

AFL++ can work in QEMU mode. It allows to fuzz target built for other architecture on an x86_64 host. AFL++ uses a customized QEMU *afl-qemu-ltrace* which can be built from the qemuafl repo through build_qemu_support script:

```
$ git clone https://github.com/AFLplusplus/AFLplusplus
$ cd $AFL && make
$ cd qemu_mode
$ CPU_TARGET=aarch64 ./build_qemu_support.sh
$ cd .. && sudo make install
```

The code coverage is directly obtained from the QEMU runtime, no need to instrument the target yourself.
You can combine the QEMU mode with:

- **libqasan** : QEMU Address Sanitizer for detecting memory corruption
- **libcompcov** : Instrument memory comparison function

They can be easily cross-compiled:

```
$ cd $AFL/qemu_mode/libqasan
$ CC=/path/to/toolchain/bin/aarch64-linux-xxxx-clang make
```

Debugging issues

- `AFL_DEBUG=1` : AFL++ debug message and child output
- `AFL_DEBUG_CHILD=1` : Only child output

Improve bug detection and coverage

Use QEMU Address Sanitizer QASAN
`AFL_USE_QASAN=1` : libqasan.so will be automatically preloaded

Fine tune code coverage

- For a specific targeted library export `AFL_QEMU_INST_RANGES=libfuzzed.so`
- For a specific address range export `AFL_QEMU_INST_RANGES=0x8B000600-0x8C048EA0`
- For all libraries export `AFL_INST_LIBS=1`

Use QEMU libcompcov

- AFL_PRELOAD env var can be used to load any library in targeted child process.
- `export AFL_PRELOAD=/usr/local/lib/libcompcov.so AFL_COMPCOV_LEVEL=2`

Improve your performance by using the QEMU Persistent mode

- It allow you to loop between an address ranges, this avoid creating a child process for a fuzz iteration.
- `AFL_QEMU_PERSISTENT_ADDR=0x55...` : Set the start address of your fuzzing loop.
- `AFL_QEMU_PERSISTENT_RET=0x55...` : Set the end address of your fuzzing loop.
- `AFL_QEMU_PERSISTNT_GPR=1` : Restore the CPU context at each fuzzing loop iteration.

You can limit the number of calibration stage needed, resulting in a tiny speed improvement

```
export AFL_FAST_CAL=1
```

For better performance always use small input sample, limit the maximum number of bytes for your input.

Launching AFL++ in QEMU mode

Launch AFL++ on your harness, `-Q` specify QEMU mode which will use *afl-qemu-ltrace* automatically

```
$ afl-fuzz -D -Q -c 0 -i input_corpus -o fuzz_output -- ./harness @@
```

Deploy several instances, one per CPU core through the uses of -M/-S

- Launch the main instance :

```
$ afl-fuzz -M main0 -D -Q -c 0 -i input_corpus -o fuzz_output -- ./harness @@
```

- Launch the worker instances :

```
$ afl-fuzz -S wrk1 -D -Q -c 0 -i input_corpus -o fuzz_output -- ./harness @@
$ afl-fuzz -S wrkN -D -Q -c 0 -i input_corpus -o fuzz_output -- ./harness @@
```

Use a terminal multiplexer like *tmux* and launch AFL++ process in different pane inside the same session.

Memory corruption vulnerabilities mitigations

Memory tagging – Introduced in ARMv9

Each memory allocation is tagged with metadata. The tag is associated with pointers when dereferencing and checked at runtime at each load or store. It helps detect UAF and buffer overflow bugs.

Control-Flow Integrity / PAC

Protects the control-flow by authenticating function calls (on some implementations, also function returns).

- CFI: Software-implemented, can be enabled at compile time.

Example of a protected function call with CFI in pseudocode:

```
// Checks if the function pointers falls in a given range.
if (0x30 < ((ulong)(function_ptr - 0x12c060) >> 3 | (long)function_ptr << 0x3d)) {
    __cfi_slowpath_diag(0xdf33000600225b0d, function_ptr);
}
res = (*function_ptr)(arg);
```

- PAC: Hardware-supported since ARMv8, uses specialized instructions, authenticates both the caller, and return address.

Example of a protected function with PAC:

```
  0x400c24:  paciasp
  0x400c28:  stp     x29, x30, [sp, #-0x10]!
  0x400c2c:  mov     x29, sp
  0x400c30:  adrp    x0, 0x400000
  0x400c34:  add     x0, x0, #0xf28
  0x400c38:  bl      printf
  0x400c40:  nop
  0x400c44:  ldp     x29, x30, [sp], #0x10
  0x400c48:  retaa
```