QEMU emulation detection

thuxnder

Chaos Computer Club Cologne e.V.
http://koeln.ccc.de

Köln
29.01.2015
Content

- QEMU
- Emulator detection
- Emulation detection
  - Binary translation engine
  - Cache behavior
"QEMU is a generic and open source machine emulator and virtualizer."

- several architectures (x86, ARM, ppc, ...)
- different backends (binary translation, KVM)
- used for dynamic analysis systems
- Android emulator included in SDK
- dynamic Android app analysis
Motivation

Why do we want to detect ...

- malware wants to hide malicious behavior
- defeat dynamic analysis (hide IP, ...)
- academic research
- fun to look under the hood
We will focus on the **Android/ARM** use case, but it is also applicable to other use cases and architectures.
Emulator detection

Use your favorite search engine :-/
We are not going into details, because most do simple pattern matching.

- Hardware specific values (CPU, graphic card, ...)
- OS specific values
Emulation detection

With the following methods we are going for a more generic approach by not detecting the emulator itself but side effects of the emulation engine that is used.
Emulation detection

Qemu uses (in most cases) a **binary translation engine** to support architectures other than the host system. This technique translates instructions to be executed into an equivalent instruction sequence on the host architecture and executes it, including memory address rewriting, etc...
Emulation detection

Qemu Binary Translation Process

- VM Execution
- Reach Branch
- Cache Lookup [hit, miss]
- Execute Basic Block [last_ins]
- Translate BBL
- Generate Code
  - translateInsn(eip);
  - eip++;

thuxnder
http://koeln.ccc.de
Emulation detection

Diagram:
- Process connected to bt-detect process
- Thread1 and Thread2 connected to bt-detect process
- QEMU connected to bt-detect process
- Operating system connected to QEMU
- Operating system connected to CPU

Thuxnder
QEMU emulation detection
http://koeln.ccc.de
Emulation detection

Argumentation Chain

PC increase after instruction → Virtual CPU PC not always updated → Problem when restoring context

Detect binary translation in qemu

Measure scheduling behavior

Interrupt after end of current basic block

Optimization
Emulation detection

Scheduling Measurement Process

Thread 1

set 1

inc

inc

inc

Thread 2

get

histogram[value] + 1

[Iteration > 100,000]

variance(histogram)

variance > threshold

Emulator!

No Emulator

QEMU emulation detection

thuxnder

http://koeln.ccc.de
Emulation detection

Source code:

- http://dexlabs.org/files/detect_bt_x86.tar.gz
- http://dexlabs.org/files/detect_bt_amd64.c
- http://dexlabs.org/files/detect_bt_arm.c
Cache behavior on ARM

- different than x86
- I-Cache and D-Cache are not synchronized

QEMU does not implement two different caches for instructions and data. This leads to an anomaly that can be detected when comparing native execution and execution in QEMU.
Cache on ARM vs X86
Cache behavior on ARM

1. Instruction is written to I-Cache for $address → Instruction is written to Cache for $address
   - Write new instruction to $address

2. Instruction is written to D-Cache for $address → Instruction is written to Cache for $address
   - Execute instruction at $address

3. Instruction is read from I-Cache for $address → Instruction is read from Cache for $address
   - Old instruction will be executed → New instruction will be executed

Old vs. New instruction
- Real device
- Emulator
Cache behavior - POC

Perparation:

```c
#define PROT PROT_EXEC|PROT_WRITE|PROT_READ
#define FLAGS MAP_ANONYMOUS|MAP_FIXED|MAP_SHARED

void *exec = mmap((void*)0x10000000, (size_t)4096, PROT, FLAGS, -1,(off_t)0);
memcpy(exec, (&eval)-1, 2048);
```

Detection code:

```assembly
mov r0, #0;
ldr r1, =0x467a3080;

; code:
add r0, #1;
mov r2, PC;
add r2, #−6;
str r1, [r2];
cmp r0, #1;
BLE code;
lsr r0, r0, #7;
```
Questions ?!
References

- http://qemu.org
- http://dexlabs.org/blog/btdetect