DBI Techniques to Address Native Code Obfuscation

Romain Thomas <rthomas@quarkslab.com>

Quarkslab
www.quarkslab.com

About

- Security engineer at Quarkslab
- Working in a team that develops tools for reverse-engineering and vulnerabilities research
- Author of LIEF
- Enjoy Android and reverse-engineering

Romain Thomas
@rh0main

What this talk is about?

Obfuscators

- Commercial solutions: Arxan, Cloakware, Epona
- In-house LLVM based solution: Snapchat, TikTok, ...
- Open-source projects: O-LLVM, Hikari, Armariris, YANSOllvm, Goron, ...

Research

- Articles by R. Rolles on `https://www.msreverseengineering.com/research`
- Breaking Spotify DRM with PANDA
  Recon 14, B. Dolan-Gavitt & *al*
- **DRM obfuscation versus auxiliary attacks**
  Recon 14, M. Camille & F. Gabriel
- API Deobfuscator Resolving Obfuscated API Functions In Modern Packers
  BH15-USA, S. Choi

Research

- **Automation Techniques in C++ Reverse Engineering**
  Recon 19, R. Rolles
- Breaking Mobile Userland
  NowSecure 19, G. Rocca
- Symbolic Deobfuscation: From Virtualized Code Back to the Original
  Dimva 18, J. Salwan, S. Bardin & M. Potet
- **Differential Computation Analysis:Hiding your White-Box Designs is Not Enough**
  SSTIC 16, C. Hubain, P. Teuwen

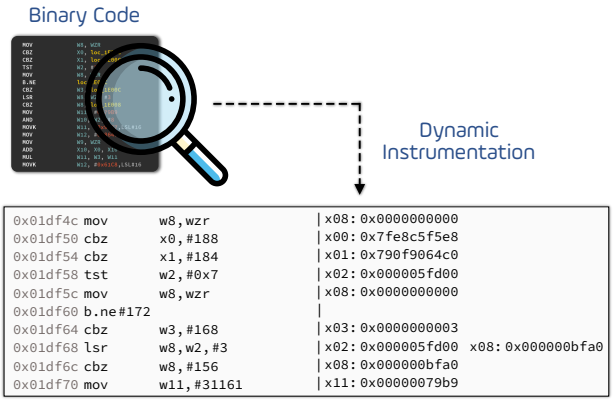
Research

TL;DR: A **DBI** seems a good **primitive** to address obfuscation.

What is a DBI

Dynamic **B**inary **I**nstrumentation is an analysis process that aims to dynamically observe binary behavior:

- Instructions
- Register values
- Basic blocks



DBI Frameworks

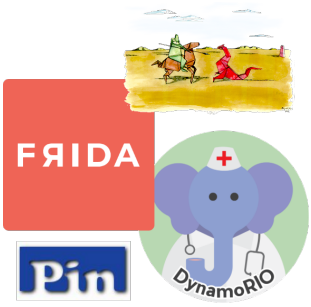
Intel PIN: `x86`, `x86-64`

Valgrind: `x86`, `x86-64`, `ARM`, `AArch64`

DynamoRIO: `x86`, `x86-64`, `ARM`, `AArch64`

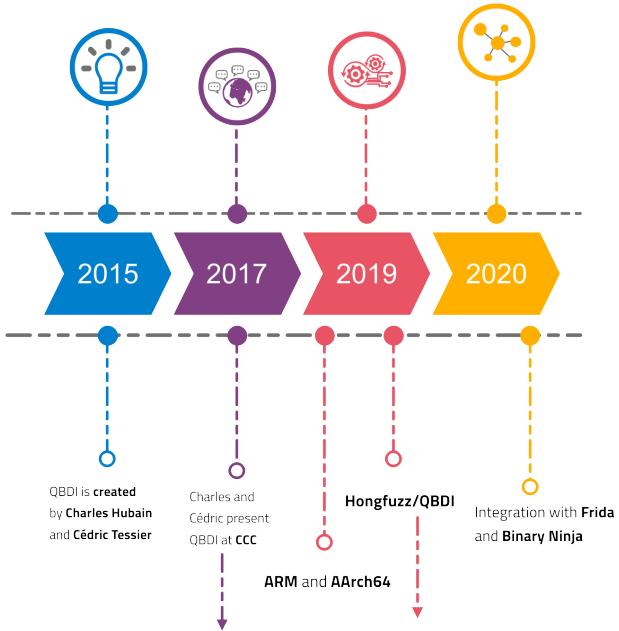
DynInst: `x86`, `x86-64`, `ARM?`, `AArch64?`

Frida: Function hooking and *stalker*



DBI Frameworks

+= QBDI


2015
2017
2019
2020
QBDI is **created** by **Charles Hubain** and **Cédric Tessier**
Charles and Cédric present QBDI at **CCC**
Hongfuzz/QBDI
Integration with **Frida** and **Binary Ninja**
ARM and **AArch64**
https://media.ccc.de/v/34c3-9006-implementing_an_llvm_based_dynamic_binary_instrumentation_framework
https://project.inria.fr/FranceJapanICST/files/2019/04/19-Kyoto-Fuzzing_Binaries_using_Dynamic_Instrumentation.pdf

QBDI

Modular DBI framework created by **Charles Hubain**[1] and **Cédric Tessier**:

Based on LLVM: Assembler, disassembler, instruction analysis and abstraction[2]

Architecture support: `x86`, `x86-64`, `ARM`[3], `AArch64`

Open-source: `https://github.com/quarkslab/QBDI`[4]

[1] Co-author of *Differential Computation Analysis:Hiding your White-Box Designs is Not Enough*
[2] Thanks to the `llvm::MCInst` interface
[3] **Full** support which includes Thumb/Thumb2
[4] Except for ARM and AArch64

QBDI Features

```
0×17505A LDR   R0, [R5]
0×17505C MOV   R8, R1
0×17505E LDR   R4, [R1, R6]
0×175060 LDR   R2, [R0, #0×18]
0×175062 MOV   R0, R5
0×175064 MOV   R1, R4
0×175066 BLX   R2
0×175068 MOV   R1, R0
0×175068 MOV   R1, R7
0×17506A MOV   R0, R7
0×17506C SVC   #0
0×17506E MOV   R1, R0
0×175070 AND.W R1, R0, R3
0×175074 EOR.W R0, R0, R3
0×175078 BLX   sub_175094
```

QBDI Features

```
0×17505A LDR   R0, [R5]
0×17505C MOV   R8, R1
0×17505E LDR   R4, [R1, R6]
0×175060 LDR   R2, [R0, #0×18]
0×175062 MOV   R0, R5
0×175064 MOV   R1, R4
0×175066 BLX   R2
0×175068 MOV   R1, R0
0×175068 MOV   R1, R7
0×17506A MOV   R0, R7
0×17506C SVC   #0
0×17506E MOV   R1, R0
0×175070 AND.W R1, R0, R3
0×175074 EOR.W R0, R0, R3
0×175078 BLX   sub_175094
```

```
1  VMAction callback(VMInstanceRef vm,
2                    GPRState *gprState, FPRState *fprState,
3                    /* user data */ void* ctx) {
4    ...
5    return VMAction::CONTINUE;
6  }
7
8  // ================================================= //
9
10 QBDI::VM vm;
11
12 vm.addCodeCB(QBDI::InstPosition::PREINST, callback,
13               /* user data */&ctx);
14
15 vm.call( ... , function, fnc_args);
```

QBDI Features

```
0×17505A LDR   R0, [R5]
0×17505C MOV   R8, R1
0×17505E LDR   R4, [R1, R6]
0×175060 LDR   R2, [R0, #0×18]
0×175062 MOV   R0, R5
0×175064 MOV   R1, R4
0×175066 BLX   R2
0×175068 MOV   R1, R0
0×175068 MOV   R1, R7
0×17506A MOV   R0, R7
0×17506C SVC   #0
0×17506E MOV   R1, R0
0×175070 AND.W R1, R0, R3
0×175074 EOR.W R0, R0, R3
0×175078 BLX   sub_175094
```

```
QBDI           Instruction Callback
.text:0×17505A LDR   R0, [R5]
QBDI           Instruction Callback
.text:0×17505C MOV   R8, R1
QBDI           Instruction Callback
.text:0×17505E LDR   R4, [R1, R6]
QBDI           Instruction Callback
.text:0×175060 LDR   R2, [R0, #0×18]
QBDI           Instruction Callback
.text:0×175062 MOV   R0, R5
QBDI           Instruction Callback
.text:0×175064 MOV   R1, R4
QBDI           Instruction Callback
.text:0×175066 BLX   R2
QBDI           Instruction Callback
.text:0×175068 MOV   R1, R0
QBDI           Instruction Callback
.text:0×175068 MOV   R1, R7
QBDI           Instruction Callback
.text:0×17506A MOV   R0, R7
QBDI           Instruction Callback
.text:0×17506C SVC   #0
QBDI           Instruction Callback
.text:0×17506E MOV   R1, R0
QBDI           Instruction Callback
.text:0×175070 AND.W R1, R0, R3
QBDI           Instruction Callback
.text:0×175074 EOR.W R0, R0, R3
QBDI           Instruction Callback
.text:0×175078 BLX   sub_175094
```

QBDI

Obfuscators tend to add **noisy instructions** that are not relevant to understand the function's logic.

QBDI

Obfuscators tend to add **noisy instructions** that are not relevant to understand the function's logic.

Moreover, instrumenting **all** the instructions can add a significant **overhead**!

QBDI

Obfuscators tend to add **noisy instructions** that are not relevant to understand the function's logic.

Moreover, instrumenting **all** the instructions can add a significant **overhead**!

$\Rightarrow$ Be *smart*

QBDI Features

```
0×17505A LDR   R0, [R5]
0×17505C MOV   R8, R1
0×17505E LDR   R4, [R1, R6]
0×175060 LDR   R2, [R0, #0×18]
0×175062 MOV   R0, R5
0×175064 MOV   R1, R4
0×175066 BLX   R2
0×175068 MOV   R1, R0
0×175068 MOV   R1, R7
0×17506A MOV   R0, R7
0×17506C SVC   #0
0×17506E MOV   R1, R0
0×175070 AND.W R1, R0, R3
0×175074 EOR.W R0, R0, R3
0×175078 BLX   sub_175094
```


```
0×17505A LDR   R0, [R5]
0×17505C MOV   R8, R1
0×17505E LDR   R4, [R1, R6]
0×175060 LDR   R2, [R0, #0×18]
0×175062 MOV   R0, R5
0×175064 MOV   R1, R4
0×175066 BLX   R2
0×175068 MOV   R1, R0
0×175068 MOV   R1, R7
0×17506A MOV   R0, R7
0×17506C SVC   #0
0×17506E MOV   R1, R0
0×175070 AND.W R1, R0, R3
0×175074 EOR.W R0, R0, R3
0×175078 BLX   sub_175094
```

QBDI Features

```
0x17505A LDR     R0,[R5]
0x17505C MOV     R8,R1
0x17505E LDR     R4,[R1,R6]
0x175060 LDR     R2,[R0,#0x18]
0x175062 MOV     R0,R5
0x175064 MOV     R1,R4
0x175066 BLX     R2
0x175068 MOV     R1,R0
0x175068 MOV     R1,R7
0x17506A MOV     R0,R7
0x17506C SVC     #0
0x17506E MOV     R1,R0
0x175070 AND.W R1,R0,R3
0x175074 EOR.W R0,R0,R3
0x175078 BLX     sub 175094
```

```
1  QBDI::VM vm;
2
3  //Callbacks before syscall instructions
4  vm.addSyscallCB(PRE_SYSCALL, syscall_cbk, &ctx);
5
6  //Callbacks before blx/bl instructions
7  vm.addCallCB(PRE_CALL, call_cbk, &ctx);
8
9  //Callbacks before memory load and store instructions
10 vm.addMemAccessCB(MEMORY_READ_WRITE, mem_cbk, &ctx);
11
12 vm.call(..., function, fnc_args);
```

QBDI Features

```
0×17505A LDR    R0, [R5]
0×17505C MOV    R8, R1
0×17505E LDR    R4, [R1, R6]
0×175060 LDR    R2, [R0, #0×18]
0×175062 MOV    R0, R5
0×175064 MOV    R1, R4
0×175066 BLX    R2
0×175068 MOV    R1, R0
0×175068 MOV    R1, R7
0×17506A MOV    R0, R7
0×17506C SVC    #0
0×17506E MOV    R1, R0
0×175070 AND.W  R1, R0, R3
0×175074 EOR.W  R0, R0, R3
0×175078 BLX    sub_175094
```

```
QBDI            Memory Callback
.text:0×17505A  LDR    R0, [R5]
.text:0×17505C  MOV    R8, R1
QBDI            Memory Callback
.text:0×17505E  LDR    R4, [R1, R6]
QBDI            Memory Callback
.text:0×175060  LDR    R2, [R0, #0×18]
.text:0×175062  MOV    R0, R5
.text:0×175064  MOV    R1, R4
QBDI            Call Callback
.text:0×175066  BLX    R2
.text:0×175068  MOV    R1, R0
.text:0×175068  MOV    R1, R7
.text:0×17506A  MOV    R0, R7
QBDI            Syscall Callback
.text:0×17506C  SVC    #0
.text:0×17506E  MOV    R1, R0
.text:0×175070  AND.W  R1, R0, R3
.text:0×175074  EOR.W  R0, R0, R3
QBDI            Call Callback
.text:0×175078  BLX    sub_175094
```

QBDI Features

In addition, QBDI is able to resolve the **effective memory address** for free!

QBDI Features

```
0×17505A LDR    R0, [R5]
0×17505C MOV    R8, R1
0×17505E LDR    R4, [R1, R6]
0×175060 LDR    R2, [R0, #0×18]
0×175062 MOV    R0, R5
0×175064 MOV    R1, R4
0×175066 BLX    R2
0×175068 MOV    R1, R0
0×175068 MOV    R1, R7
0×17506A MOV    R0, R7
0×17506C SVC    #0
0×17506E MOV    R1, R0
0×175070 AND.W  R1, R0, R3
0×175074 EOR.W  R0, R0, R3
0×175078 BLX    sub_175094
```

```
1  VMAction mem_cbk(VMInstanceRef vm, ...) {
2
3    // Memory info
4    MemoryAccess info = vm→getInstMemoryAccess().back();
5    // Provide:
6    // LDR  R0, [R5]              → R5
7    // LDR  R4, [R1, R6]          → R1 + R6
8    // LDR  R2, [R3, #0×18]       → R3 + 0×18
9    // STRB R8, [R1, -R2, LSL #4] → R1 - (R2 << 4)
10   uintptr_t addr = maccess.accessAddress;
11
12   // Value read or written (R0, R4, R2, R8)
13   uintptr_t value = maccess.value;
14   ...
15   return VMAction::CONTINUE;
16 }
```

QBDI Features

Same mechanism for `blx`, `bl`, `br`, `blr` instructions[1].

[1] not available in the public branch yet

QBDI Features

```
0×17505A LDR    R0, [R5]
0×17505C MOV    R8, R1
0×17505E LDR    R4, [R1, R6]
0×175060 LDR    R2, [R0, #0×18]
0×175062 MOV    R0, R5
0×175064 MOV    R1, R4
0×175066 BLX    R2
0×175068 MOV    R1, R0
0×175068 MOV    R1, R7
0×17506A MOV    R0, R7
0×17506C SVC    #0
0×17506E MOV    R1, R0
0×175070 AND.W  R1, R0, R3
0×175074 EOR.W  R0, R0, R3
0×175078 BLX    sub_175094
```

```
1  VMAction call_cbk(VMInstanceRef vm, ...) {
2
3    // Provide:
4    // BLX R2   → Value of R2
5    // BLX PC   → PC + CPU Mode shift + Alignment
6    // BLX #42  → PC + 42 + CPU Mode shift + Alignment
7    uintptr_t call_target = vm→getInstCallAccess().back();
8
9    ...
10   return VMAction::CONTINUE;
11 }
```

External Calls

What about external calls to `malloc()`, `printf()`, `env->FindClass()`, ...?

External Calls

It would not be relevant to instrument their **code** right?

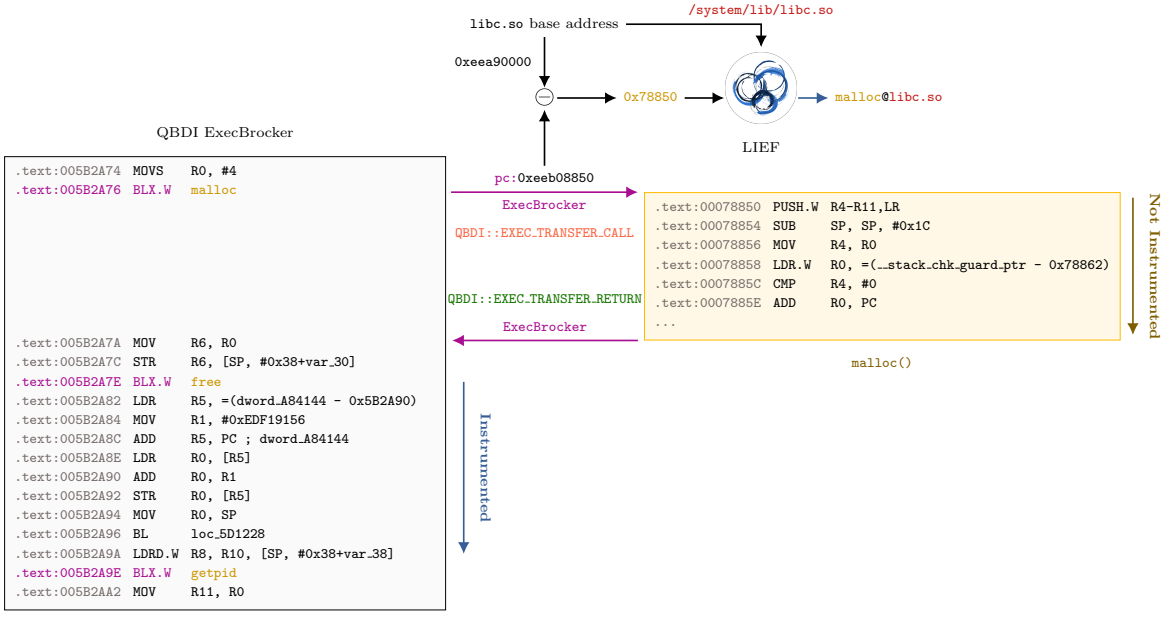
[1] By changing the return address (`lr`) to a dedicated stub that will resume the instrumentation process

External Calls

It would not be relevant to instrument their **code** right?

QBDI implements a mechanism to **stop** the instrumentation process, **leave** the function executing its own code (without instrumentation) and **catch** when it returns.[1]

[1] By changing the return address (`lr`) to a dedicated stub that will resume the instrumentation process


libc.so base address
/system/lib/libc.so
0xeea90000
0x78850
malloc@libc.so
LIEF
QBDI ExecBrocker
.text:005B2A74 MOVS R0, #4
.text:005B2A76 BLX.W malloc
pc:0xeeb08850
ExecBrocker
QBDI::EXEC_TRANSFER_CALL
.text:00078850 PUSH.W R4-R11,LR
.text:00078854 SUB SP, SP, #0x1C
.text:00078856 MOV R4, R0
.text:00078858 LDR.W R0, =(__stack_chk_guard_ptr - 0x78862)
.text:0007885C CMP R4, #0
.text:0007885E ADD R0, PC
...
Not Instrumented
QBDI::EXEC_TRANSFER_RETURN
ExecBrocker
malloc()
.text:005B2A7A MOV R6, R0
.text:005B2A7C STR R6, [SP, #0x38+var_30]
.text:005B2A7E BLX.W free
.text:005B2A82 LDR R5, =(dword_A84144 - 0x5B2A90)
.text:005B2A84 MOV R1, #0xEDF19156
.text:005B2A8C ADD R5, PC ; dword_A84144
.text:005B2A8E LDR R0, [R5]
.text:005B2A90 ADD R0, R1
.text:005B2A92 STR R0, [R5]
.text:005B2A94 MOV R0, SP
.text:005B2A96 BL loc_5D1228
.text:005B2A9A LDRD.W R8, R10, [SP, #0x38+var_38]
.text:005B2A9E BLX.W getpid
.text:005B2AA2 MOV R11, R0
Instrumented

JNI Functions

Regarding JNI functions, we can use the same mechanism with:

- `env->FindClass()`
- `env->RegisterNatives()`
- `env->NewStringUTF()`
- ...

JNI Functions

JNI function pointers are stored in the `JNIEnv* env` variable.

```
1  typedef const struct JNINativeInterface* JNIEnv;
2
3  struct JNINativeInterface {
4    ...
5    jclass (*DefineClass)(JNIEnv*, ...),
6    jclass (*FindClass)(JNIEnv*, ...);
7    jmethodID (*FromReflectedMethod)(JNIEnv*, ...);
8    ...
9  }
```

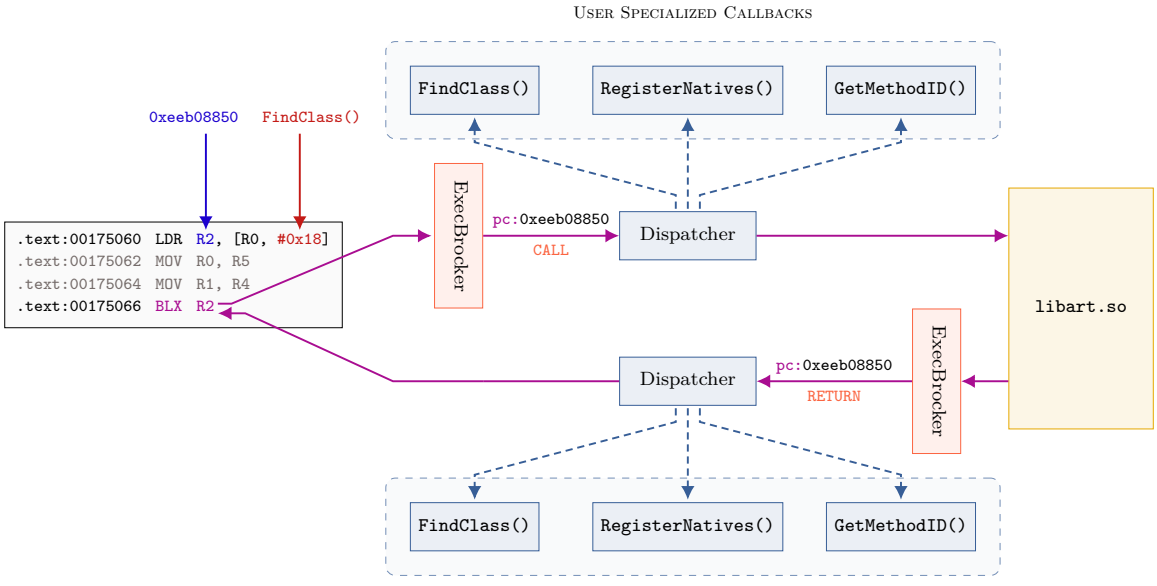
Figure – `NDK_PATH/sysroot/usr/include/jni.h`

JNI Functions

Consequently, given the `env` variable and using the **ExecBroker**, we can catch a call to `env->FindClass` by checking if `pc` matches `&(env->FindClass)`

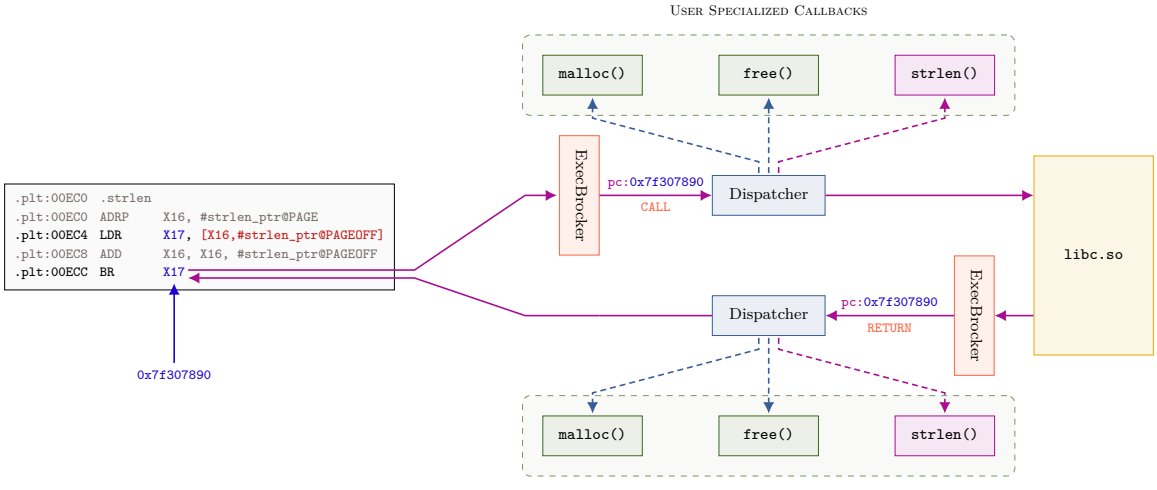
```
1  typedef const struct JNINativeInterface* JNIEnv;
2
3  struct JNINativeInterface {
4    ...
5    jclass (*DefineClass)(JNIEnv*, ...),
6    jclass (*FindClass)(JNIEnv*, ...);
7    jmethodID (*FromReflectedMethod)(JNIEnv*, ...);
8    ...
9  }
```

Figure – `NDK_PATH/sysroot/usr/include/jni.h`


User Specialized Callbacks
FindClass()
RegisterNatives()
GetMethodID()
0xeeb08850
FindClass()
.text:00175060 LDR R2, [R0, #0x18]
.text:00175062 MOV R0, R5
.text:00175064 MOV R1, R4
.text:00175066 BLX R2
ExecBrocker
pc:0xeeb08850
CALL
Dispatcher
libart.so
Dispatcher
pc:0xeeb08850
RETURN
ExecBrocker
FindClass()
RegisterNatives()
GetMethodID()
User Specialized Callbacks

External Functions

As we can resolve absolute addresses into symbols, one can implement **pre** and **post** callbacks to inspect function **parameters** and **return values**.


User Specialized Callbacks
malloc()
free()
strlen()
ExecBrocker
pc:0x7f307890
CALL
Dispatcher
libc.so
.plt:00EC0 .strlen
.plt:00EC0 ADRP X16, #strlen_ptr@PAGE
.plt:00EC4 LDR X17, [X16,#strlen_ptr@PAGEOFF]
.plt:00EC8 ADD X16, X16, #strlen_ptr@PAGEOFF
.plt:00ECC BR X17
0x7f307890
Dispatcher
pc:0x7f307890
RETURN
ExecBrocker
malloc()
free()
strlen()
User Specialized Callbacks

QBDI Assets

To summarize, QBDI enables:

- Instruction trace
- Memory trace
- Syscall trace
- Internal call trace
- Library call trace

```
QBDI            InstructionCallback
.text:0x17505A LDR     R0,[R5]
QBDI            InstructionCallback
.text:0x17505C MOV     R8,R1
QBDI            InstructionCallback
.text:0x17505E LDR     R4,[R1,R6]
QBDI            InstructionCallback
.text:0x175060 LDR     R2,[R0,#0x18]
QBDI            InstructionCallback
.text:0x175062 MOV     R0,R5
QBDI            InstructionCallback
.text:0x175064 MOV     R1,R4
QBDI            InstructionCallback
.text:0x175066 BLX     R2
QBDI            InstructionCallback
.text:0x175068 MOV     R1,R0
QBDI            InstructionCallback
.text:0x175068 MOV     R1,R7
QBDI            InstructionCallback
.text:0x17506A MOV     R0,R7
QBDI            InstructionCallback
.text:0x17506C SVC     #0
QBDI            InstructionCallback
.text:0x17506E MOV     R1,R0
QBDI            InstructionCallback
.text:0x175070 AND.W R1,R0,R3
QBDI            InstructionCallback
.text:0x175074 EOR.W R0,R0,R3
QBDI            InstructionCallback
.text:0x175078 BLX     sub_175094
```

```
QBDI            MemoryCallback
.text:0x17505A LDR     R0,[R5]
.text:0x17505C MOV     R8,R1
QBDI            MemoryCallback
.text:0x17505E LDR     R4,[R1,R6]
QBDI            MemoryCallback
.text:0x175060 LDR     R2,[R0,#0x18]
.text:0x175062 MOV     R0,R5
.text:0x175064 MOV     R1,R4
QBDI            CallCallback
.text:0x175066 BLX     R2
.text:0x175068 MOV     R1,R0
.text:0x175068 MOV     R1,R7
.text:0x17506A MOV     R0,R7
QBDI            SyscallCallback
.text:0x17506C SVC     #0
.text:0x17506E MOV     R1,R0
.text:0x175070 AND.W R1,R0,R3
.text:0x175074 EOR.W R0,R0,R3
QBDI            CallCallback
.text:0x175078 BLX     sub_175094
```

Dynamic Analysis

Setup N°1

1. Add **PRE** and **POST** syscall callbacks
2. Setup the ExecBroker to catch external calls
3. Resolve external functions and syscalls into names to forward their analysis to a dedicated callback

Dynamic Analysis

Setup N°1

1. Add **PRE** and **POST** syscall callbacks
2. Setup the ExecBroker to catch external calls
3. Resolve external functions and syscalls into names to forward their analysis to a dedicated callback

$\hookrightarrow$ Call trace

Dynamic Analysis

Hex-Rays
STATE-OF-THE-ART BINARY CODE ANALYSIS SOLUTIONS

Static Analysis

```
[30902:30902:672581000] 0x0b52ba .text!0x2a33e0 (#3) {
[30902:30902:674646000]      0x2a401c malloc(0x1b4): 0xf08c0280
[30902:30902:743507000]      READ:  ip_vti0
[30902:30902:743546000]      WRITE: ip_vti0
[30902:30902:743552000] }
[30902:30902:747740000] 0x0b52ba .text!0x2a33e0 (#4) {
[30902:30902:749373000]      0x2a401c malloc(0x1b4): 0xf08c0440
[30902:30902:865788000]      READ:  ip6_vti0
[30902:30902:865827000]      WRITE: ip6_vti0
[30902:30902:865833000] }
[30902:30902:868590000] 0x0b52ba .text!0x2a33e0 (#5) {
[30902:30902:870223000]      0x2a401c malloc(0x1b4): 0xf08c0600
[30902:30902:951361000]      READ:  sit0
[30902:30902:951396000]      WRITE: sit0
[30902:30902:951401000] }
```

FRIDA

Dynamic Trace

Call & Memory Callbacks

Setup N°2

1. Add **PRE** and **POST** call callbacks
2. Record byte memory accesses

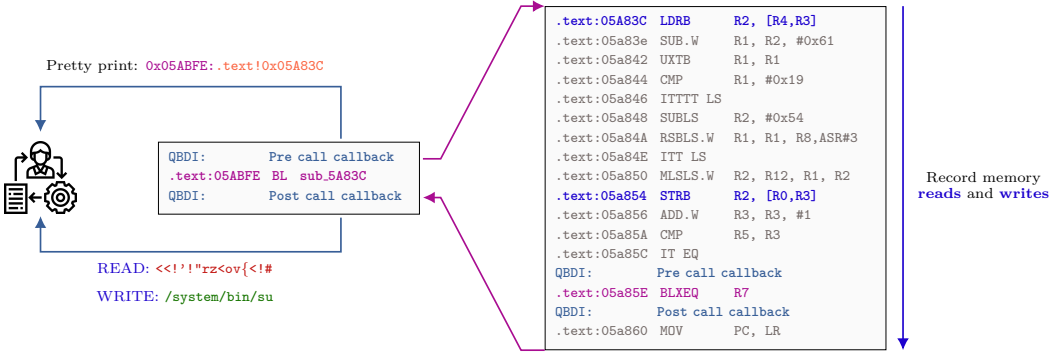
Call & Memory Callbacks

Setup N°2

1. Add **PRE** and **POST** call callbacks
2. Record byte memory accesses

↪ Suitable to observe (obfuscated) strings used by a function

Call & Memory Callbacks



Dynamic Trace: Memory Read & Write

```
19 @ 0000179c  w4 = zx.d([x2 + x5].b)
20 @ 000017a0  w4 = w4 ^ (w1 s>> 8)
21 @ 000017a4  w4 = w4 ^ w1
22 @ 000017a8  w4 = w4 ^ (w1 s>> 0x10)
23 @ 000017ac  [x0 + x5].b = (w4).b
24 @ 000017b0  x5 = x5 + 1
25 @ 000017b4  x1 = zx.q((x1 u>> 1) & 0xffffff)
26 @ 000017bc  if (x5 != x3) then 19 @ 0x179c else 27 @ 0x17c0
```

```
00029120  53 4d 66 fc 68 d2 b4 05  SMf.h...
00029128  c1 05 59 64 6a 66 67 01  ..Ydjfg.
00029130  3c 97 9d 62 00 00 00 00  <..b....
00029138  00 00 00 00 00 00 00 00  ........
00029140  66 c0 b7 1f 9b ab 08 5b  f......[
00029148  ee 12 56 7a 79 61 02 72  ..Vzya.r
00029150  24 5e fb a9 00 00 00 00  $^......
00029158  00 00 00 00 00 00 00 00  ........

00029160  char data_29160[18] = "Z\xde8\xd8\xf8\x18\xda9\xc63_:{`otd", 0
```

```
qbdi.trace                                                    buffers
 1
 2 0x014538: 0x143d0() {
 3     0x01442c: 0x177c() {
 4         READ: Z893_:{`otd | WRITE: /dev/socket/qemud
 5     } // 0x177c (#75)
 6
 7     0x014440: 0x16c80() {
 8         0x016ca0: syscall: openat(0xffffff9c, '/dev/socket/qemud')
 9     } // 0x16c80 (#242)
10
11     0x014460: 0x13d90() {
12     } // 0x13d90 (#12060)
13
14     0x01442c: 0x177c() {
15         READ: [Vzyar$^ | WRITE: /dev/qemu_pipes$^
16     } // 0x177c (#76)
17
18     0x014440: 0x16c80() {
19         0x016ca0: syscall: openat(0xffffff9c, '/dev/qemu_pipe')
20     } // 0x16c80 (#243)
NORMAL  /tmp/qbdi.trace                          5%  1/20 ㏑ :  1
"qbdi.trace" 20L, 507C written
(8) romain 4:tmp*                                2.24 1.43 0.86 13:45
```

```
00029160  char data_29160[18] = "Z\xde8\xd8\xf8\x18\xda9\xc63_:{`otd", 0
```

```python
import string

def decode(encoded: bytes, key: int) -> str:
    k = key
    out = []
    for a in encoded:
        b = ((a ^ (k >> 8)) ^ k) ^ (k >> 0x10)
        k = (k >> 1) & 0xffffff
        out.append(b & 0xFF)
    return bytes(out).decode("utf8")

buff = b"Z\xde8\xd8\xf8\x18\xda9\xc63_:{`otd"
key  = 0x9bd20000add8
printable = "".join(chr(c) for c in buff if chr(c) in string.printable)
print(printable, "--->", decode(buff, key))
```

```
➜  script python ./decode_str.py
Z893_:{`otd ---> /dev/socket/qemud
➜  script


(8) romain4:script*                0.51 1.09 0.83 13:48
```

qbdi.trace

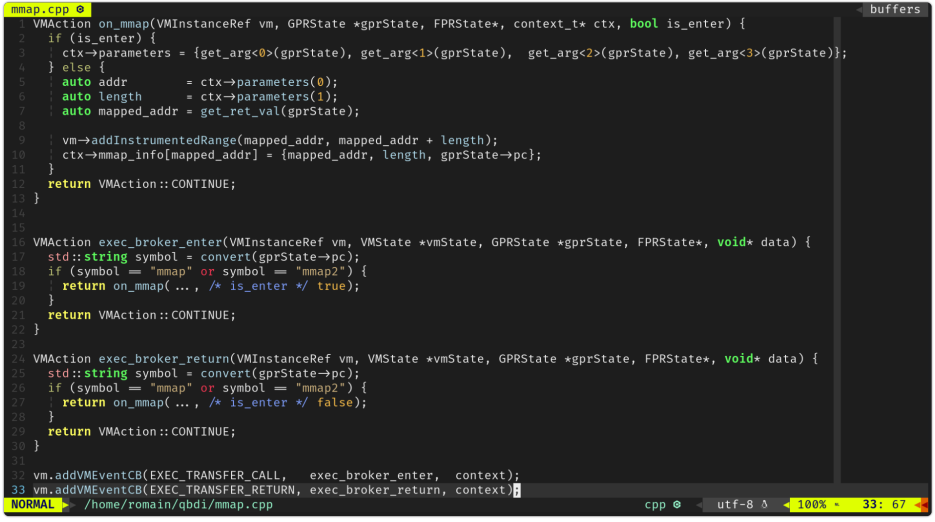
```
0x014538: 0x143d0() {
    0x01442c: 0x177c() {
        READ: Z893_:{`otd | WRITE: /dev/socket/qemud
    } // 0x177c (#75)

    0x014440: 0x16c80() {
        0x016ca0: syscall: openat(0xffffff9c, '/dev/socket/qemud')
    } // 0x16c80 (#242)
```

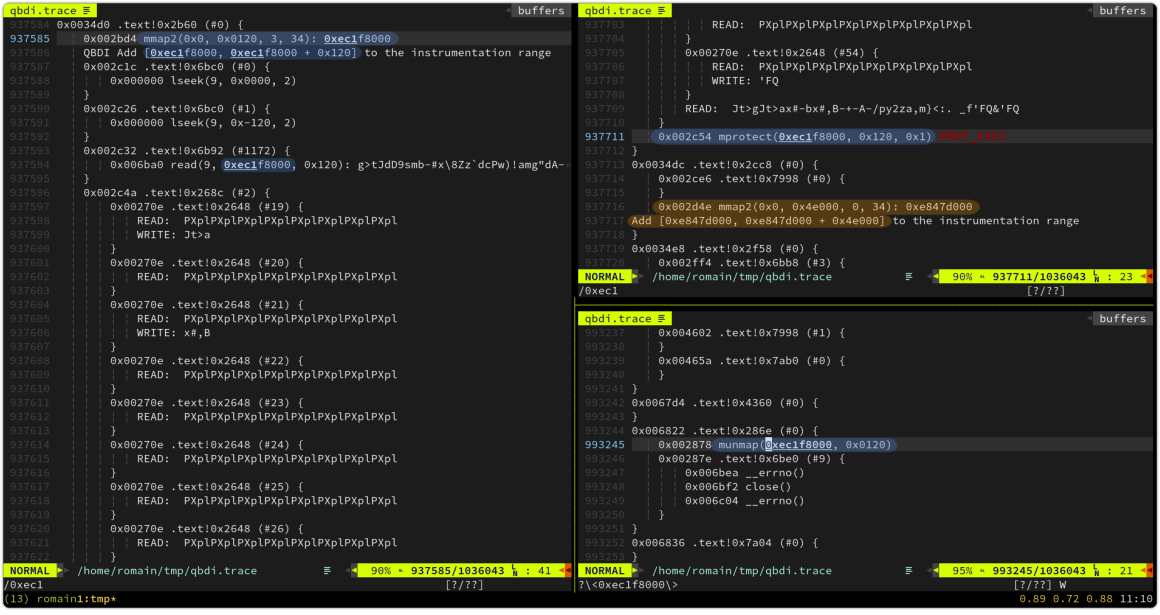
Dynamic analysis

Setup N°3

1. Track memory allocations: `operator new()`, `malloc`, `calloc`, `...`
2. When a call to `mmap()` is caught ⇒ update the QBDI's instrumented range
3. When a memory area is deallocated, inspect the bytes being freed. It might contain strings.

Dynamic analysis




qbdi.trace
buffers
0x0034d0 .text!0x2b60 (#0) {
937585 0x002bd4 mmap2(0x0, 0x0120, 3, 34): 0xec1f8000
QBDI Add [0xec1f8000, 0xec1f8000 + 0x120] to the instrumentation range
0x002c1c .text!0x6bc0 (#0) {
0x000000 lseek(9, 0x0000, 2)
}
0x002c26 .text!0x6bc0 (#1) {
0x000000 lseek(9, 0x-120, 2)
}
0x002c32 .text!0x6b92 (#1172) {
0x006ba0 read(9, 0xec1f8000, 0x120): g>tJdD9smb-#x\8Zz`dcPw)!amg"dA~
}
0x002c4a .text!0x268c (#2) {
0x00270e .text!0x2648 (#19) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
WRITE: Jt>a
}
0x00270e .text!0x2648 (#20) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
}
0x00270e .text!0x2648 (#21) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
WRITE: x#,B
}
0x00270e .text!0x2648 (#22) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
}
0x00270e .text!0x2648 (#23) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
}
0x00270e .text!0x2648 (#24) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
}
0x00270e .text!0x2648 (#25) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
}
0x00270e .text!0x2648 (#26) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
}
NORMAL /home/romain/tmp/qbdi.trace 90% ≈ 937585/1036043 ℔ : 41
/0xec1 [?/??]
qbdi.trace
buffers
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
}
0x00270e .text!0x2648 (#54) {
READ: PXplPXplPXplPXplPXplPXplPXplPXpl
WRITE: 'FQ
}
READ: Jt>gJt>ax#-bx#,B-+-A~/py2za,m}<:. _f'FQ&'FQ
}
937711 0x002c54 mprotect(0xec1f8000, 0x120, 0x1) PROT_EXEC
}
0x0034dc .text!0x2cc8 (#0) {
0x002ce6 .text!0x7998 (#0) {
}
0x002d4e mmap2(0x0, 0x4e000, 0, 34): 0xe847d000
Add [0xe847d000, 0xe847d000 + 0x4e000] to the instrumentation range
}
0x0034e8 .text!0x2f58 (#0) {
0x002ff4 .text!0x6bb8 (#3) {
NORMAL /home/romain/tmp/qbdi.trace 90% ≈ 937711/1036043 ℔ : 23
/0xec1 [?/??]
qbdi.trace
buffers
0x004602 .text!0x7998 (#1) {
}
0x00465a .text!0x7ab0 (#0) {
}
}
0x0067d4 .text!0x4360 (#0) {
}
0x006822 .text!0x286e (#0) {
993245 0x002878 munmap(0xec1f8000, 0x0120)
0x00287e .text!0x6be0 (#9) {
0x006bea __errno()
0x006bf2 close()
0x006c04 __errno()
}
}
0x006836 .text!0x7a04 (#0) {
}
NORMAL /home/romain/tmp/qbdi.trace 95% ≈ 993245/1036043 ℔ : 21
?\<0xec1f8000\> [?/??] W
(13) romain1:tmp* 0.89 0.72 0.88 11:10

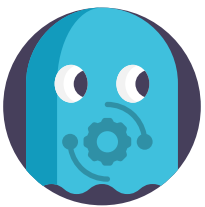
Use Cases

Use Cases

Use Cases

The following examples are based on two commercial obfuscators (Blue and Green).

JNI_OnLoad()



JNI_OnLoad Obfuscation

JNI_OnLoad()

```java
package gh;

import android.support.annotation.Keep;

public class wer {
    @Keep
    private static final byte[] e = {41, 82, -31, 109, 9, 85, 95, 77, 57, 121, -53, Byte.MAX_VALUE,

    public static native String a(String[] strArr, String[] strArr2, String str, byte[] bArr);

    public static native String b(String str);

    public static native String c(String str);

    public static native int d(byte[] bArr, byte[] bArr2);
}
```

JNI_OnLoad()

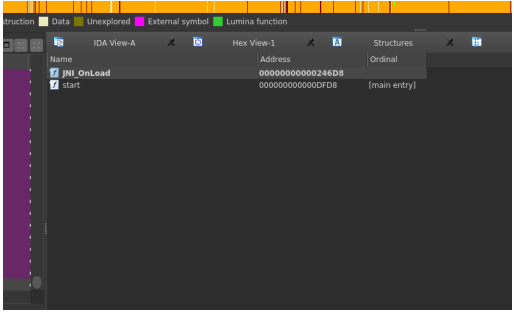


Figure – Library exports

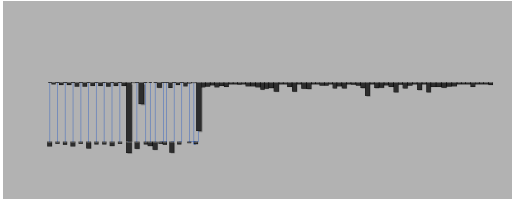
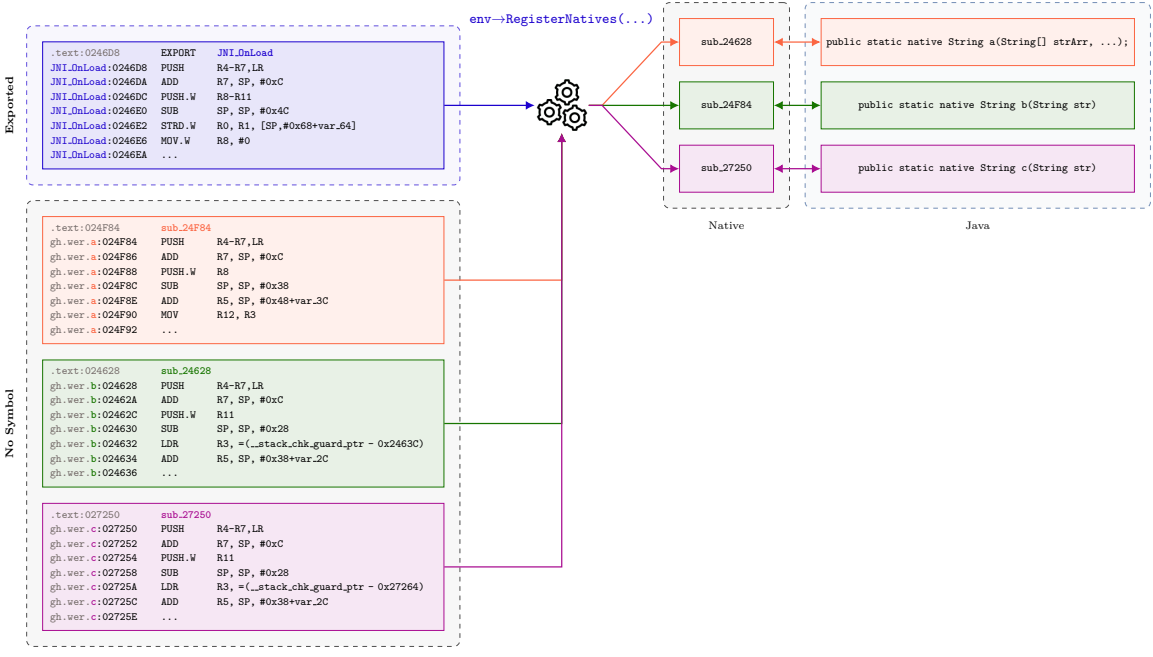


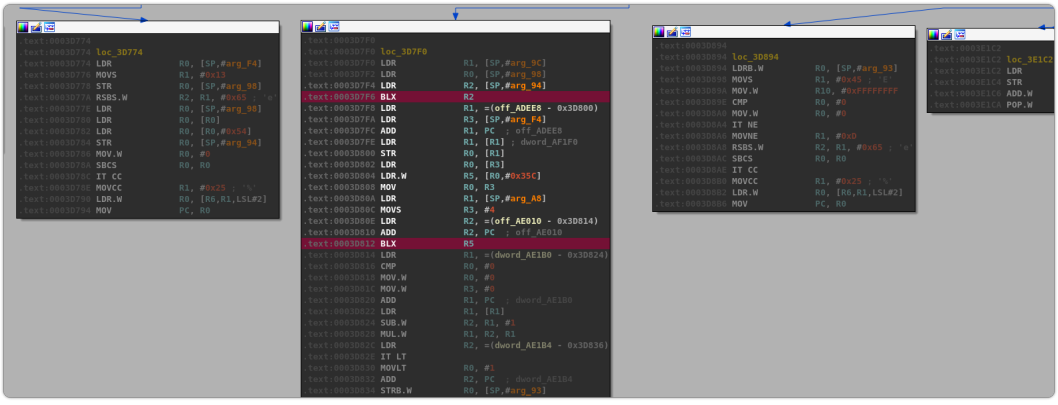
Figure – `JNI_OnLoad()` protected by Green


Exported
.text:0246D8 EXPORT JNI_OnLoad
JNI_OnLoad:0246D8 PUSH R4-R7,LR
JNI_OnLoad:0246DA ADD R7, SP, #0xC
JNI_OnLoad:0246DC PUSH.W R8-R11
JNI_OnLoad:0246E0 SUB SP, SP, #0x4C
JNI_OnLoad:0246E2 STRD.W R0, R1, [SP,#0x68+var_64]
JNI_OnLoad:0246E6 MOV.W R8, #0
JNI_OnLoad:0246EA ...
env→RegisterNatives(...)
sub_24628
sub_24F84
sub_27250
public static native String a(String[] strArr, ...);
public static native String b(String str)
public static native String c(String str)
Native
Java
No Symbol
.text:024F84 sub_24F84
gh.wer.a:024F84 PUSH R4-R7,LR
gh.wer.a:024F86 ADD R7, SP, #0xC
gh.wer.a:024F88 PUSH.W R8
gh.wer.a:024F8C SUB SP, SP, #0x38
gh.wer.a:024F8E ADD R5, SP, #0x48+var_3C
gh.wer.a:024F90 MOV R12, R3
gh.wer.a:024F92 ...
.text:024628 sub_24628
gh.wer.b:024628 PUSH R4-R7,LR
gh.wer.b:02462A ADD R7, SP, #0xC
gh.wer.b:02462C PUSH.W R11
gh.wer.b:024630 SUB SP, SP, #0x28
gh.wer.b:024632 LDR R3, =(__stack_chk_guard_ptr - 0x2463C)
gh.wer.b:024634 ADD R5, SP, #0x38+var_2C
gh.wer.b:024636 ...
.text:027250 sub_27250
gh.wer.c:027250 PUSH R4-R7,LR
gh.wer.c:027252 ADD R7, SP, #0xC
gh.wer.c:027254 PUSH.W R11
gh.wer.c:027258 SUB SP, SP, #0x28
gh.wer.c:02725A LDR R3, =(__stack_chk_guard_ptr - 0x27264)
gh.wer.c:02725C ADD R5, SP, #0x38+var_2C
gh.wer.c:02725E ...

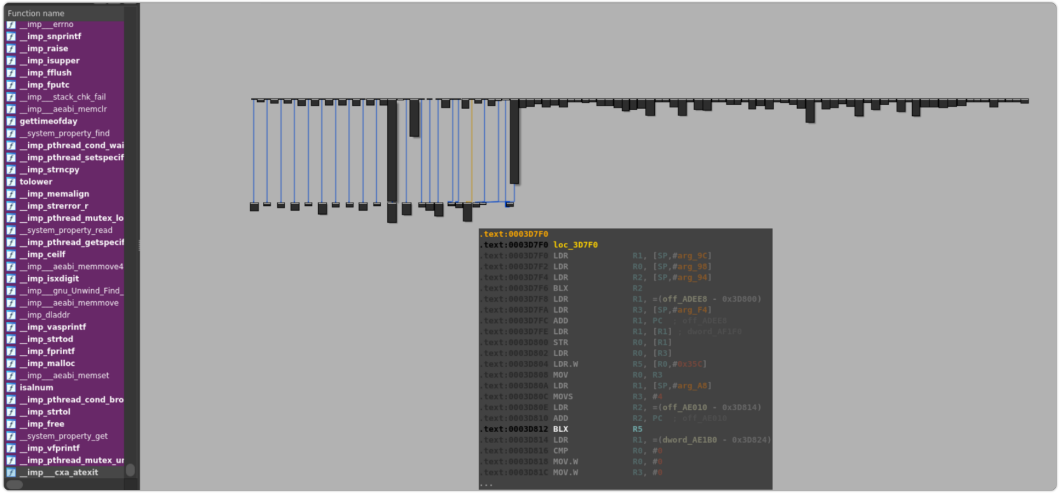
JNI_OnLoad()

```
0x024852 .text!0x8955d (#1) {
    0x0895fe .text!0x6ce8e (#4) {
        0x06ceb8 .text!0x6d120 (#4) {}
    }
}
0x0248fe .text!0x3c769 (#0) {
    0x03d5b4 jvm->GetEnv(...)
    0x03d640 env->FindClass('gh/wer'): 0x5
    0x03d748 env->FindClass('java/lang/RuntimeException'): 0x19
    0x03d7f8 env->NewGlobalRef(0x19): 0xee6
    0x03d812 env->RegisterNatives('gh.wer', ..., nb_methods=4)
        a -> /data/local/tmp/libXYplugin.10.59.0.0.so@0x24f85
        b -> /data/local/tmp/libXYplugin.10.59.0.0.so@0x24629
        c -> /data/local/tmp/libXYplugin.10.59.0.0.so@0x27251
        d -> /data/local/tmp/libXYplugin.10.59.0.0.so@0x84e15
    0x03d8ce .text!0x37695 (#0) {}
    0x03d900 .text!0x4c629 (#0) {
        0x04c878 .text!0x42b9d (#0) {
            0x047744 .text!0x52bb9 (#0) {
                0x054a90 env->FindClass('com/snapchat/android/framework/misc/AppContext'): 0x21
            }
            0x047756 env->NewGlobalRef(0x21): 0xef6
            0x047744 .text!0x52bb9 (#1) {
                0x054a90 env->FindClass('android/content/Context'): 0x25
            }
```

JNI_OnLoad()


.text:0003D774 loc_3D774
.text:0003D774 LDR R0, [SP,#arg_F4]
.text:0003D776 MOVS R1, #0x13
.text:0003D778 STR R0, [SP,#arg_98]
.text:0003D77A RSBS.W R2, R1, #0x65 ; 'e'
.text:0003D77E LDR R0, [SP,#arg_98]
.text:0003D780 LDR R0, [R0]
.text:0003D782 LDR R0, [R0,#0x54]
.text:0003D784 STR R0, [SP,#arg_94]
.text:0003D786 MOV.W R0, #0
.text:0003D78A SBCS R0, R0
.text:0003D78C IT CC
.text:0003D78E MOVCC R1, #0x25 ; '%'
.text:0003D792 LDR.W R0, [R6,R1,LSL#2]
.text:0003D796 MOV PC, R0
.text:0003D7F0
.text:0003D7F0 loc_3D7F0
.text:0003D7F0 LDR R1, [SP,#arg_9C]
.text:0003D7F2 LDR R0, [SP,#arg_98]
.text:0003D7F4 LDR R2, [SP,#arg_94]
.text:0003D7F6 BLX R2
.text:0003D7F8 LDR R1, =(off_ADEE8 - 0x3D800)
.text:0003D7FA LDR R3, [SP,#arg_F4]
.text:0003D7FC ADD R1, PC ; off_ADEE8
.text:0003D7FE LDR R1, [R1] ; dword_AF1F0
.text:0003D800 STR R0, [R1]
.text:0003D802 LDR R0, [R3]
.text:0003D804 LDR.W R5, [R0,#0x35C]
.text:0003D808 MOV R0, R3
.text:0003D80A LDR R1, [SP,#arg_A8]
.text:0003D80C MOVS R3, #4
.text:0003D80E LDR R2, =(off_AE010 - 0x3D814)
.text:0003D810 ADD R2, PC ; off_AE010
.text:0003D812 BLX R5
.text:0003D814 LDR R1, =(dword_AE1B0 - 0x3D824)
.text:0003D816 CMP R0, #0
.text:0003D818 MOV.W R0, #0
.text:0003D81C MOV.W R3, #0
.text:0003D820 ADD R1, PC ; dword_AE1B0
.text:0003D822 LDR R1, [R1]
.text:0003D824 SUB.W R2, R1, #1
.text:0003D828 MUL.W R1, R2, R1
.text:0003D82C LDR R2, =(dword_AE1B4 - 0x3D836)
.text:0003D82E IT LT
.text:0003D830 MOVLT R0, #1
.text:0003D832 ADD R2, PC ; dword_AE1B4
.text:0003D834 STRB.W R0, [SP,#arg_93]
.text:0003D894
.text:0003D894 loc_3D894
.text:0003D894 LDRB.W R0, [SP,#arg_93]
.text:0003D898 MOVS R1, #0x45 ; 'E'
.text:0003D89A MOV.W R10, #0xFFFFFFFF
.text:0003D89E CMP R0, #0
.text:0003D8A0 MOV.W R0, #0
.text:0003D8A4 IT NE
.text:0003D8A6 MOVNE R1, #0xD
.text:0003D8A8 RSBS.W R2, R1, #0x65 ; 'e'
.text:0003D8AC SBCS R0, R0
.text:0003D8AE IT CC
.text:0003D8B0 MOVCC R1, #0x25 ; '%'
.text:0003D8B2 LDR.W R0, [R6,R1,LSL#2]
.text:0003D8B6 MOV PC, R0
.text:0003E1C2
.text:0003E1C2 loc_3E1C2
.text:0003E1C2 LDR
.text:0003E1C4 STR
.text:0003E1C6 ADD.W
.text:0003E1CA POP.W

JNI_OnLoad()


Function name
__imp___errno
__imp_snprintf
__imp_raise
__imp_isupper
__imp_fflush
__imp_fputc
__imp___stack_chk_fail
__imp___aeabi_memclr
gettimeofday
__system_property_find
__imp_pthread_cond_wai
__imp_pthread_setspecif
__imp_strncpy
tolower
__imp_memalign
__imp_strerror_r
__imp_pthread_mutex_lo
__system_property_read
__imp_pthread_getspecif
__imp_ceilf
__imp___aeabi_memmove4
__imp_isxdigit
__imp___gnu_Unwind_Find_
__imp___aeabi_memmove
__imp_dladdr
__imp_vasprintf
__imp_strtod
__imp_fprintf
__imp_malloc
__imp___aeabi_memset
isalnum
__imp_pthread_cond_bro
__imp_strtol
__imp_free
__system_property_get
__imp_vfprintf
__imp_pthread_mutex_ur
__imp___cxa_atexit
.text:0003D7F0
.text:0003D7F0 loc_3D7F0
.text:0003D7F0 LDR R1, [SP,#arg_9C]
.text:0003D7F2 LDR R0, [SP,#arg_98]
.text:0003D7F4 LDR R2, [SP,#arg_94]
.text:0003D7F6 BLX R2
.text:0003D7F8 LDR R1, =(off_ADEE8 - 0x3D800)
.text:0003D7FA LDR R3, [SP,#arg_F4]
.text:0003D7FC ADD R1, PC ; off_ADEE8
.text:0003D7FE LDR R1, [R1] ; dword_AE1F8
.text:0003D800 STR R0, [R1]
.text:0003D802 LDR R0, [R3]
.text:0003D804 LDR.W R5, [R0,#0x35C]
.text:0003D808 MOV R0, R3
.text:0003D80A LDR R1, [SP,#arg_A8]
.text:0003D80C MOVS R3, #1
.text:0003D80E LDR R2, =(off_AE010 - 0x3D814)
.text:0003D810 ADD R2, PC ; off_AE010
.text:0003D812 BLX R5
.text:0003D814 LDR R1, =(dword_AE1B0 - 0x3D824)
.text:0003D816 CMP R0, #0
.text:0003D818 MOV.W R0, #0
.text:0003D81C MOV.W R3, #0

JNI_OnLoad()

Demo:

https://www.romainthomas.fr/publication/20-bh-asia-dbi/#demo1

Video Game Protection

Video Game Protections

Video Game Protection

Android video games have to face similar threats as desktop video games. It includes cheats by memory tampering or dynamic modifications.

Video Game Protection

A famous video game implements Frida detection routines, the whole protected by Blue.

Video Game Protection

A famous video game implements Frida detection routines, the whole protected by Blue.

To avoid that Frida bypass `anti-frida()`, they inline syscall instructions.

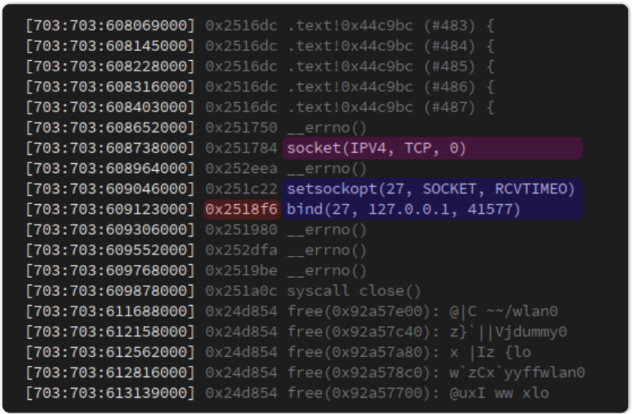
Video Game Protection

A famous video game implements Frida detection routines, the whole protected by Blue.

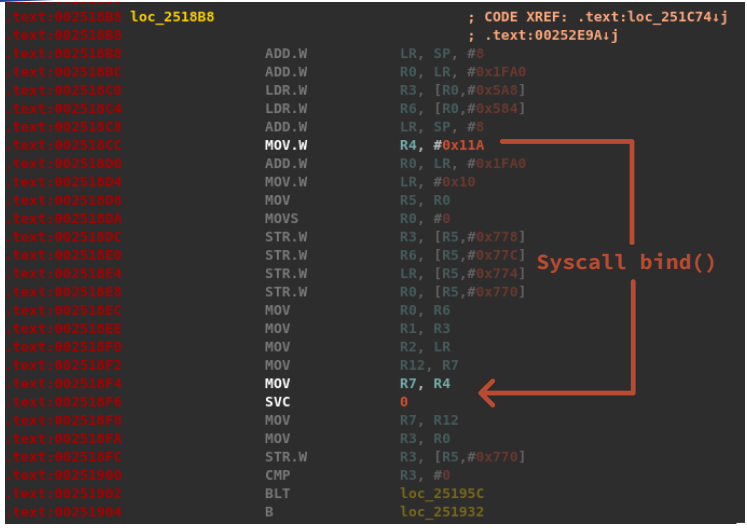
To avoid that Frida bypass `anti-frida()`, they inline syscall instructions.

Fortunately, QBDI can trace **syscalls**!

Video Game Protection

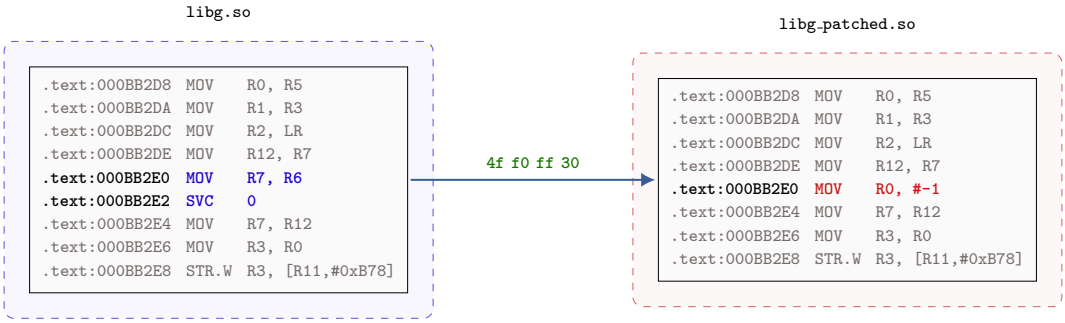


Video Game Protection



Video Game Protection

Thanks to **QBDI** we are able to locate a subset of the syscalls used ⇒ we can patch the instructions with **LIEF**[1]

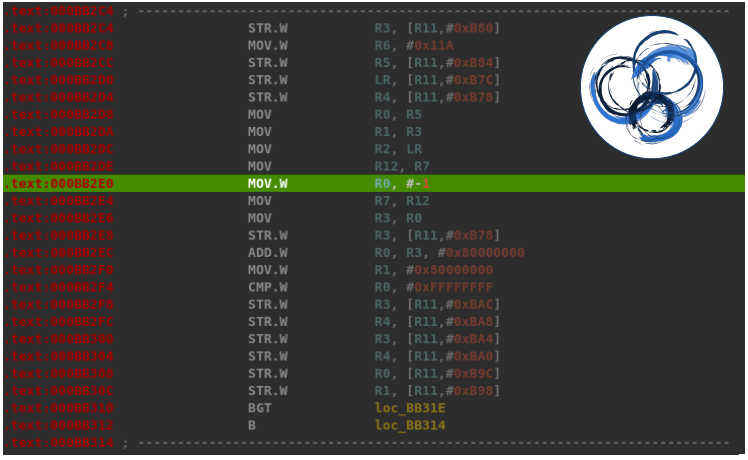


[1] https://gist.github.com/romainthomas/f25b0377d8f0f37601c9a223e2105f32

LIEF Patching

```
.text:000BB2C4 loc_BB2C4                               ; DATA XREF: sub_BAA84+82E↑o
.text:000BB2C4                                         ; sub_BAA84:off_BB2C0↑o
.text:000BB2C4                 STR.W           R3, [R11,#0xB80]
.text:000BB2C8                 MOV.W           R6, #0x11A
.text:000BB2CC                 STR.W           R5, [R11,#0xB84]
.text:000BB2D0                 STR.W           LR, [R11,#0xB7C]
.text:000BB2D4                 STR.W           R4, [R11,#0xB78]
.text:000BB2D8                 MOV             R0, R5
.text:000BB2DA                 MOV             R1, R3
.text:000BB2DC                 MOV             R2, LR
.text:000BB2DE                 MOV             R12, R7
.text:000BB2E0                 MOV             R7, R6
.text:000BB2E2                 SVC             0
.text:000BB2E4                 MOV             R7, R12
.text:000BB2E6                 MOV             R3, R0
.text:000BB2E8                 STR.W           R3, [R11,#0xB78]
.text:000BB2EC                 ADD.W           R0, R3, #0x80000000
.text:000BB2F0                 MOV.W           R1, #0x80000000
.text:000BB2F4                 CMP.W           R0, #0xFFFFFFFF
.text:000BB2F8                 STR.W           R3, [R11,#0xBAC]
.text:000BB2FC                 STR.W           R4, [R11,#0xBA8]
.text:000BB300                 STR.W           R3, [R11,#0xBA4]
.text:000BB304                 STR.W           R4, [R11,#0xBA0]
.text:000BB308                 STR.W           R0, [R11,#0xB9C]
.text:000BB30C                 STR.W           R1, [R11,#0xB98]
.text:000BB310                 BGT             loc_BB31E
.text:000BB312                 B               loc_BB314
.text:000BB314 ; ---------------------------------------------------------------------------
```

LIEF Patching

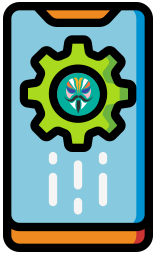


LIEF Patching

The game does not crash anymore when Frida server is running![1]

[1] Several `SVC` have to be patched

Root Detection in a MDM Application



Root detection in a MDM solution

Root Detection in MDM Application

Mobile **D**evice **M**anagement (MDM) solutions aim to provide an enhanced control over the devices deployed in a company.

Root Detection in MDM Application

One of the *must-have* features in these solutions is to be able to efficiently detect if the device on which the MDM agent is deployed is **rooted** or **jailbroken**.

Root Detection in MDM Application

One of the *must-have* features in these solutions is to be able to efficiently detect if the device on which the MDM agent is deployed is **rooted** or **jailbroken**.

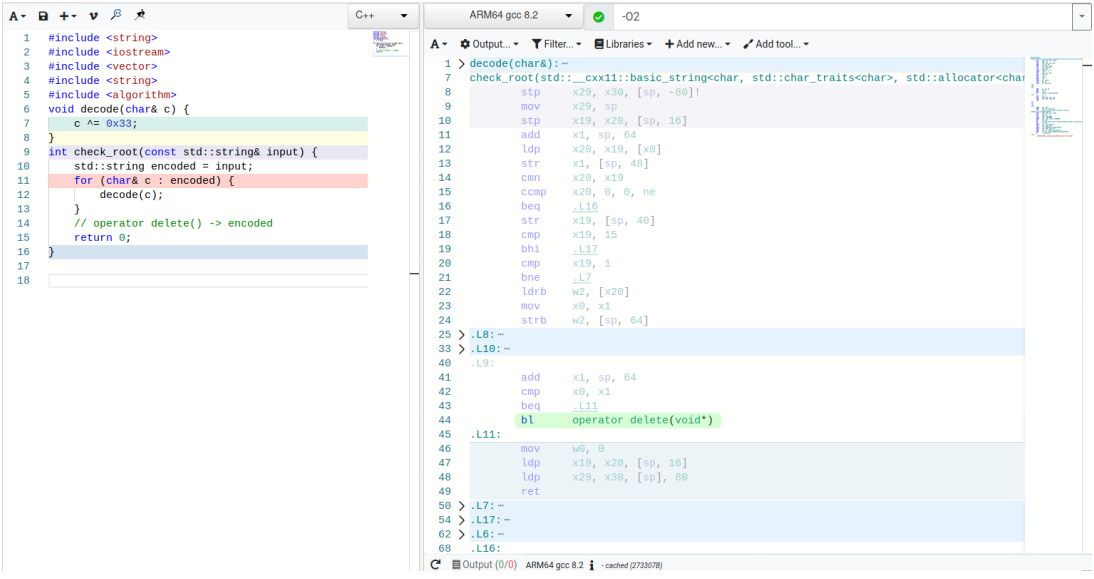
$\Rightarrow$ somehow the code needs to interact with the operating system.

Root Detection in MDM Application

The MDM solution analyzed with QBDI is written in C++, the whole being obfuscated by Blue.

Root Detection in MDM Application

C++ involves destructors that can be implicitly defined in the code.


C++
ARM64 gcc 8.2
-O2
Output...
Filter...
Libraries
Add new...
Add tool...
#include <string>
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
void decode(char& c) {
c ^= 0x33;
}
int check_root(const std::string& input) {
std::string encoded = input;
for (char& c : encoded) {
decode(c);
}
// operator delete() -> encoded
return 0;
}
decode(char&):
check_root(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char
stp x29, x30, [sp, -80]!
mov x29, sp
stp x19, x20, [sp, 16]
add x1, sp, 64
ldp x20, x19, [x0]
str x1, [sp, 48]
cmn x20, x19
ccmp x20, 0, 0, ne
beq .L16
str x19, [sp, 40]
cmp x19, 15
bhi .L17
cmp x19, 1
bne .L7
ldrb w2, [x20]
mov x0, x1
strb w2, [sp, 64]
.L8:
.L10:
.L9:
add x1, sp, 64
cmp x0, x1
beq .L11
bl operator delete(void*)
.L11:
mov w0, 0
ldp x19, x20, [sp, 16]
ldp x29, x30, [sp], 80
ret
.L7:
.L17:
.L6:
.L16:
Output (0/0)
ARM64 gcc 8.2
- cached (2733078)

Root Detection in a MDM Application

Caught by QBDI's ExecBrocker

```
[21595:21595:416839000]  0x0586c8 delete(0x5df4320): eu.chainfire.supersu
[21595:21595:417040000]  0x0586c8 delete(0x5df4348): com.koushikdutta.superuser
[21595:21595:417267000]  0x0586c8 delete(0x5dac148): /sbin/su
[21595:21595:417444000]  0x0586c8 delete(0x5dac190): /system/su
[21595:21595:417632000]  0x0586c8 delete(0x5df4370): /system/bin/.ext/.su
[21595:21595:417835000]  0x0586c8 delete(0x5c36630): ""/system/usr/we-need-root/su-backup
[21595:21595:418027000]  0x0586c8 delete(0x5df4398): com.android.settings
[21595:21595:418228000]  0x0586c8 delete(0x5df43c0): cyanogenmod.superuser
[21595:21595:418454000]  0x0586c8 delete(0x5d75830): $$com.zachspong.temprootremovIsRoot.jb
[21595:21595:418717000]  0x0586c8 delete(0x5df43e8): com.ramdroid.appquarantine
[21595:21595:418924000]  0x0586c8 delete(0x5df4410): /etc/security/otacerts.zip
[21595:21595:419105000]  0x0586c8 delete(0x5d92640): eu.chainfire.adbd
[21595:21595:419287000]  0x0586c8 delete(0x5d925c0): /system/xbin/mu
[21595:21595:419547000]  0x0586c8 delete(0x5d92600): /system/etc/hosts
[21595:21595:419764000]  0x0586c8 delete(0x5cd34e0): 30
[21595:21595:420015000]  0x0586c8 delete(0x5dac550): Micromax
[21595:21595:420213000]  0x0586c8 delete(0x5dac5b0): ls -l
[21595:21595:420411000]  0x0586c8 delete(0x5dac5f8): /data/data
[21595:21595:420636000]  0x0586c8 delete(0x5dac640): HP Slate 7
[21595:21595:420847000]  0x0586c8 delete(0x5cd31e0): ZTE
[21595:21595:420982000]  0x0586c8 delete(0x5c366f0): ""/system/framework/XposedBridge.jar
[21595:21595:421181000]  0x0586c8 delete(0x4c64860): 99grep --binary-files=text "Xposed" /system/bin/app_process
[21595:21595:421399000]  0x0586c8 delete(0x4c64950): BBgrep --binary-files=text "Xposed" /system/bin/app_process32_xposed
[21595:21595:421603000]  0x0586c8 delete(0x4c64900): BBgrep --binary-files=text "Xposed" /system/bin/app_process64_xposed
[21595:21595:421740000]  0x0586c8 delete(0x5df4438): /system/bin/app_process
[21595:21595:421918000]  0x0586c8 delete(0x5df4460): /system/bin/app_process32
[21595:21595:422094000]  0x0586c8 delete(0x5df4488): /system/bin/app_process64
[21595:21595:422212000]  0x0586c8 delete(0x5dacb20): Xposed
```

Packer

Android Packer

Packer

A Glimpse Into Tencent's Legu Packer

Date Tue 26 November 2019 By Romain Thomas Category Android Tags android reverse engineering packer

Analysis of Tencent Legu: a packer for Android applications.

Introduction

This blog post deals with the Legu packer, an Android protector developed by Tencent that is currently one of the state-of-the-art solutions to protect APK DEX files. The packer is updated frequently and this blog post focuses on versions `4.1.0.15` and `4.1.0.18`.

Overview

An application protected with Legu is composed of two native libraries: `libshell-super.2019.so` and `libshella-4.1.0.XY.so` as well as raw binary files embedded in the resources of the APK:

- `tosversion`
- `0OO0Ol111lll`
- `0OO0OooOl11ll`
- `oOooOO0ooOo.dat`

The main logic of the packer is located in the native library `libshell-super.2019.so` which basically unpacks and loads the protected DEX files from the resources.

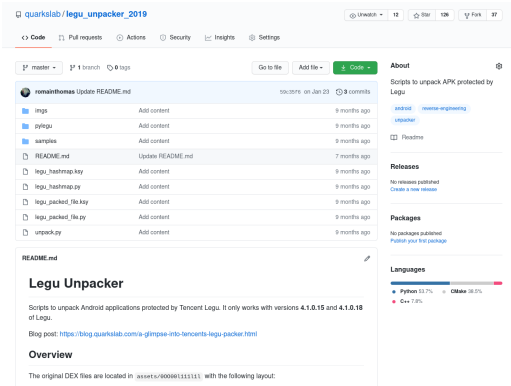
Some functions of the library are obfuscated but thanks to Frida/QBDI their analysis is not a big deal.

Internals

Basically, the original DEX files are located in the `assets/0OO0Ol111lll` file along with the information required to unpack them.

The following figure lays out the structure of this file.

https://blog.quarkslab.com/a-glimpse-into-tencents-legu-packer.html



https://github.com/quarkslab/legu_unpacker_2019

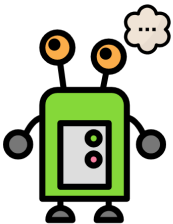
Packer

An insight of the analysis with QBDI:

`https://www.romainthomas.fr/publication/20-bh-asia-dbi/#demo2`

Conclusion

3 points to keep in mind when dealing with obfuscation



Conclusion

- Obfuscators can't break all **program semantics**: syscalls, memory accesses, external calls, etc
- A DBI enables to recover some of them: *"You have primitives, make your recipe"*
- A drawback of commercial obfuscators compared to in-house obfuscators is that they don't clearly know beforehand which asset aims to be protected $\Longrightarrow$ developers can fail in applying suitable obfuscation passes.

Acknowledgements

This talk would not have been possible without the initial work of

Charles Hubain and **Cédric Tessier**!

Acknowledgements

Many thanks to the **LLVM** community that provides such a powerful framework that is widely used in the security field:

- Rellic & Remill from Trailofbits
- QBDI
- Arybo[1] [2]
- RetDec from Avast
- O-LLVM & cie
- ...

[1] https://github.com/quarkslab/arybo
[2] https://triton.quarkslab.com/files/DIMVA2018-deobfuscation-salwan-bardin-potet.pdf

Acknowledgements

Thanks to my colleagues and Quarkslab to provide such an environment to explore these topics!

Questions?

Quarkslab

SECURING EVERY BIT OF YOUR DATA