Fun (or not) with C ABIs

(with some free ad for DragonFFI)

Pass the salt 2018 - Adrien Guinet (@adriengnt)
2018/02/04

Content of this talk

- whoami
- What's in a C ABI?
- Problems for reverse engineers
- Solutions?

Whoami

Adrien Guinet (@adriengnt)

- Working at Quarkslab on an LLVM-based obfuscator
- Some open source contributions: Cython/Pythran, DragonFFI, faup (furl at the time) ...

What's in a C ABI

Lots of things that are not defined by the C standard itself:

- **Calling conventions**: how to pass arguments / get the return value of a function
- C **structures layout** / padding
- **Bit fields** implementation
- Executable format
- Example for the x86-64 SystemV ABI: https://www.uclibc.org/docs/psABI-x86_64.pdf

Example of various calling convention

Let's take this C function:

```
typedef struct {
  float a;
  float b;
} Point;

Point mul2(Point x) {
  Point ret;
  ret.a = x.a*2;
  ret.b = x.b*2;
  return ret;
}
```

Example of various calling convention

Compiled for x86-64 Linux:

```
$ clang-6.0 -S -emit-llvm -o - a.c -O2
define <2 x float> @mul2(<2 x float>) local_unnamed_addr #0
  %2 = fmul <2 x float> %0, <float 2.000000e+00,="" float=""
  ret <2 x float> %2
}

$ clang-6.0 -S -mllvm -x86-asm-syntax=intel -o - a.c -O2
mul2:
  addps   xmm0, xmm0
  ret
</float>
```

Example of various calling convention

Compiled for x86-32 Linux:

```
$ clang-6.0 -S -emit-llvm -o - a.c -O2 -m32
define void @mul2(%struct.Point* noalias nocapture sret, flo
  %4 = fmul float %1, 2.000000e+00
  %5 = fmul float %2, 2.000000e+00
  %6 = getelementptr inbounds %struct.Point, %struct.Point*
  store float %4, float* %6, align 4
  %7 = getelementptr inbounds %struct.Point, %struct.Point*
  store float %5, float* %7, align 4
  ret void
}

$ clang-6.0 -S -mllvm -x86-asm-syntax=intel -o - a.c -O2  -m
mul2:
  movss   xmm0, dword ptr [esp + 12] # xmm0 = mem[0],zero,ze
  movss   xmm1, dword ptr [esp + 8] # xmm1 = mem[0],zero,zer
  mov     eax, dword ptr [esp + 4]
  addss   xmm1, xmm1
  addss   xmm0, xmm0
  movss   dword ptr [eax], xmm1
```

```
movss   dword ptr [eax + 4], xmm0
ret     4
```

Problems when reversing/for forensics

- Lots of ABIs gotchas to know for reversers
- **Dumping structures** and analyzing them: parsing code must follow all these rules
- For **automatic decompilers**: hard to guess original C function signatures
- Calling C function from **foreign language/virtual machines**

Issues for automatic decompilers

Example: these 2 functions:

```
typedef struct {
  float a;
  float b;
} Point;

Point mul2(Point x) {
  Point ret;
  ret.a = x.a*2;
  ret.b = x.b*2;
  return ret;
}

void mul2_ref(Point* ret, Point x) {
  ret->a = x.a*2;
  ret->b = x.b*2;
}
```

give the **same code** when compiled for **x86-32/Linux**, but not for x86-64/Linux

Foreign function calls

- From a VM (qemu/unicorn, Miasm): setting the proper stack/registers, jumping into the function, getting the return value. **ABI dependent**
- From a **foreign language** (i.e. python)

Foreign function calls: example

Calling a C function from Python:

```
import pydffi
CU = pydffi.FFI().cdef('''
typedef struct {
  float x;
  float y;
} Point;
Point mul_2(Point x);
''')

P = CU.funcs.mul_2(CU.types.Point(x=1.,y=2.))
print(P.x, P.y)
```

You can also do this with cppyy and (at some point) cffi!

Solutions?

- **libffi**: reference FFI library, generates ASM wrappers to abstract various C ABIs. **cffi** gives Python bindings.
- **dragonffi**: implementation of a FFI based on Clang/LLVM, with Python bindings
- **cppyy**: automatic bindings for C++ libraries from Python (really impressive, based on cling)

They all have pros and cons, don't have time to list them all here!

What's missing?

- "Simple" library for FFI within a VM (i.e.: call a compiled complex C ARM32 function within a Miasm VM running on x86)
- Library to parse C structure definition and get parsers for a fixed architecture, with high-level language bindings
- Offline FFI compilers for shared libraries
- Clear parsable definition of various ABIs rules

If people are interested in this, let's talk about it!

Thanks for your attention!

https://github.com/aguinet/dragonffi

```
pip install pydffi
```

For Linux x86/64 users only

Twitter: @adriengnt
Mail: adrien@guinet.me