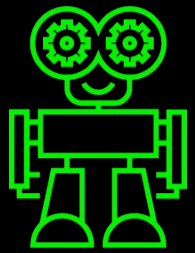
Codename: flip.re

Hypervisor-level debugging and beyond



Lars Haukli
@zutle
lars@0xepic.com

Intro

My reaction from r2con 2016:

“wow, a debugger that can do all that!”

- code analysis capabilities

- emulator + debugger

- scriptability

The malware landscape

350 000 new malware threats per day *

The challenge: Understanding the threats that matter to you

- Discover the malware threats and actors that can hurt you
- Handle the malware threats that end up causing security incidents

* source: AV-TEST

Malware analysis challenges

Even analyzing 1‰ (permyriad) of 350 000 would mean 35 per day!

A single threat can easily take 2 weeks to analyze

Sophisticated malware actively evade malware analysis solutions

The flip.re project

An interactive malware analysis system

- Analyze advanced threats faster
- Triage threats faster

You need to understand how a malware threat works before it’s too late!

Effective malware analysis

“For me, reverse engineering efficiently has always been about combining dynamic and static techniques”

Dynamic analysis is a lot faster, and almost always useful for deeper analysis

- Get an idea of what the code is doing
- Locate the code you want to analyze further

Evasive Malware (anti-analysis techniques)

Malware designed to frustrate reverse engineers by
actively making analysis challenging

Code obfuscation techniques typically combined with runtime checks to detect that it is being analyzed

Examples:

- VM/Sandbox detection
- Self-debugging to prevent analyst from attaching a debugger

Current malware sandboxes for analysis use

Often used as an initial (dynamic) analysis

Sometimes able to produce meaningful reports, but analysis is typically superficial

All fully automated analysis solutions will be vulnerable to evasive malware. Always. It is not possible to fix.

To solve the malware evasion problem, you need to solve the halting problem.

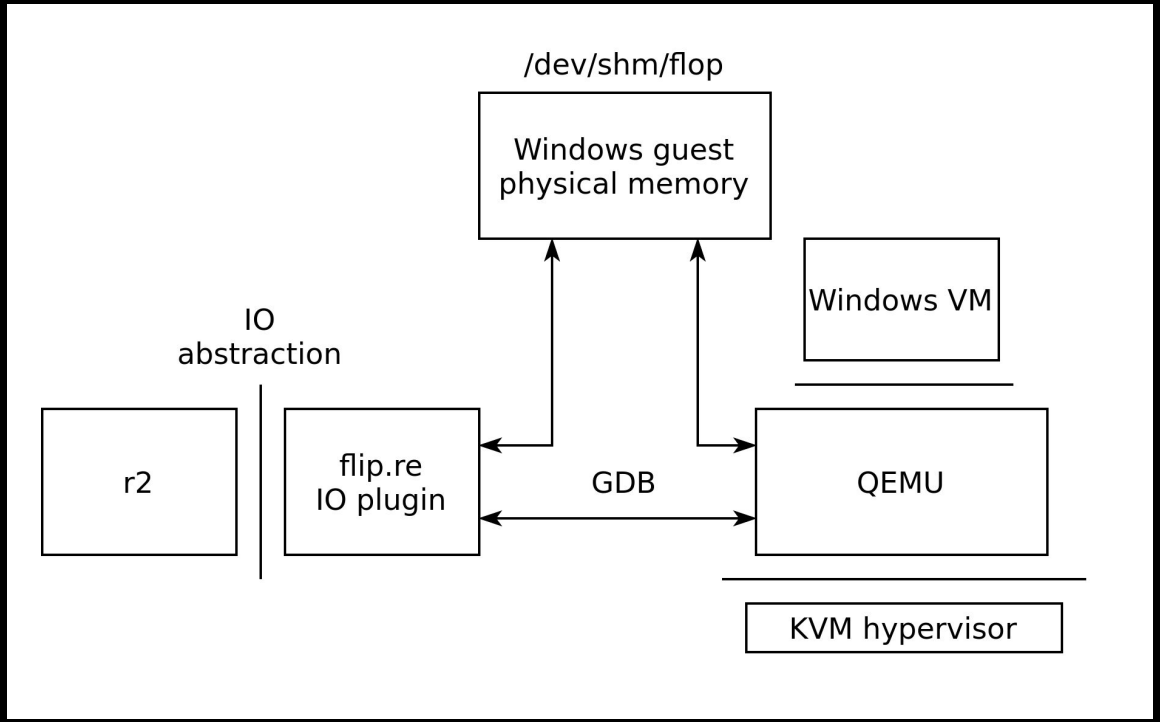
Core design principle

Less is more.

A stealthy approach for dynamic analysis will increase your chances of not having to worry about evasive malware causing incorrect results.

We still expect malware to attempt evasion, so we enable a human analyst to interact with the system to counter it.

Architecture



Advantage: stealthy/sneaky

radare2 IO/debug plugin

Written in Rust

- Amazing language with an excellent modern toolchain
- Great replacement for C as a safer and more reliable systems programming language

- Sometimes frustrating when types get complicated, and (as a C coder) the compiler can feel overly strict
- Unsafe Rust has a very similar feeling to C with lots of type casting

build.rs

Using bindgen, you can generate Rust FFI bindings to use the radare2 C plugin API

```
let bindings = bindgen::Builder::default()
    .rustfmt_bindings(true)
    .derive_default(true)
    .header("r2_wrapper.h")
    .clang_arg(format!("-I{}", r2_include_dir.trim()))
    .clang_arg(format!("-I{}/sdb", r2_include_dir.trim()))
    .whitelist_var("R2_VERSION")
    .whitelist_var("R_LIB_TYPE_.*")
    .whitelist_var("R_SYS_BITS_32|R_SYS_BITS_64")
    .whitelist_var("R_IO_SEEK_SET|R_IO_SEEK_CUR|R_IO_SEEK_END")
    .whitelist_type("RLibStruct|RIOPlugin|RDebugPlugin|RDebugPid|RListIter")
    .generate()
    .unwrap_or_else(|()| {
        println!("Whoops! Something went wrong when trying to generate Rust bindings
        for radare2.");
        process::exit(1);
    });
```

build.rs

You can also compile C code with the Rust build system (cargo), and link it into your target executable (using the cc crate)

```
let cfg = cc::Build::new()
    .include(r2_include_dir.trim())
    .include(format!("{}/sdb", r2_include_dir.trim()))
    .include("gdb/include")
    .file("gdb/src/libgdbr.c")
    .file("gdb/src/core.c")
    .file("gdb/src/messages.c")
    .file("gdb/src/packet.c")
    .file("gdb/src/utils.c")
    .compile("gdb");
```

Developing radare2 plugins in Rust

We are releasing an example Rust plugin for r2con 2020 for anyone interested in learning more about developing radare2 plugins in Rust

https://github.com/flipreversing/r2con2020

GDB

‘-s’ param in QEMU enables gdbstub (GDB server) on localhost:1234

Effectively enables us to debug things that run inside the VM

- Kernel
- Any user-mode process

Features:

- Query CPU registers
- Single Step
- Trap on Breakpoints

Physical memory access

QEMU can share the physical memory of the VM as a `memory-backend-file`

-object memory-backend-file,[...],mem-path=/dev/shm/flop,share=on

Enables accessing guest physical memory (Windows) from the host (Linux)

Virtual memory access (Windows VM)

Each process has its own CR3 values

- two values each (user/kernel) after the Meltdown/Spectre discoveries

x86 architecture defines address translation logic

- Windows has additional mechanisms on top
- Outside scope of this presentation (r2con 2017, Windows Internals)

Our plugin translates guest virtual addresses into host physical addresses, so we can effectively work with virtual addresses inside the Windows VM

PDB symbols

Our "API" to the Windows kernel
- Details on location and layout of structures (used internally by our plugin)
- Also useful to load as flags for the analyst

Currently you can feed our plugin the path to ntoskrnl.exe from the Windows VM the the first time you attach

We require the GUID: r2 -c “i~guid[1]” -q ntoskrnl.exe
and the pdb name: r2 -c “i~dbg_file[1]” -q ntoskrnl.exe

With this info, we can download the correct PDB from the Microsoft,
and parse the PDB file using the pdb crate (see https://crates.io/crates/pdb)

Live demo time!

- Attaching to a Windows VM
- Downloading PDB files from the Microsoft symbol server
- Load kernel symbols as flags in r2
- Debug a program running inside the Windows VM

Case Study: Anti-Anti analysis

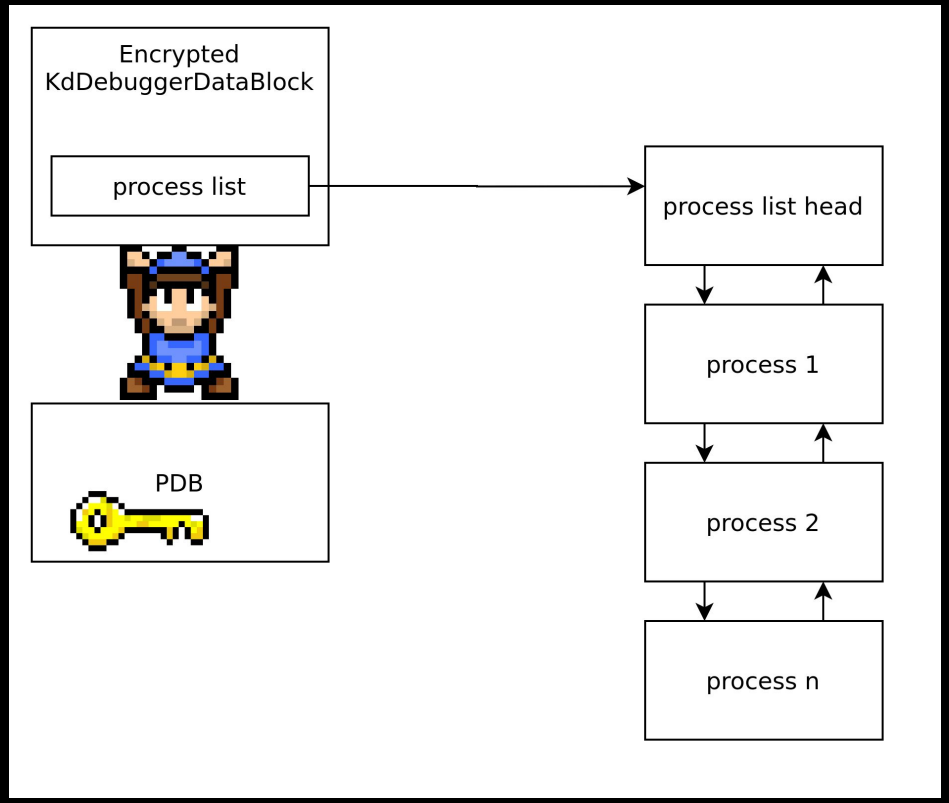
Tracing can be an effective technique to locate interesting code

Frida is a great tool for this

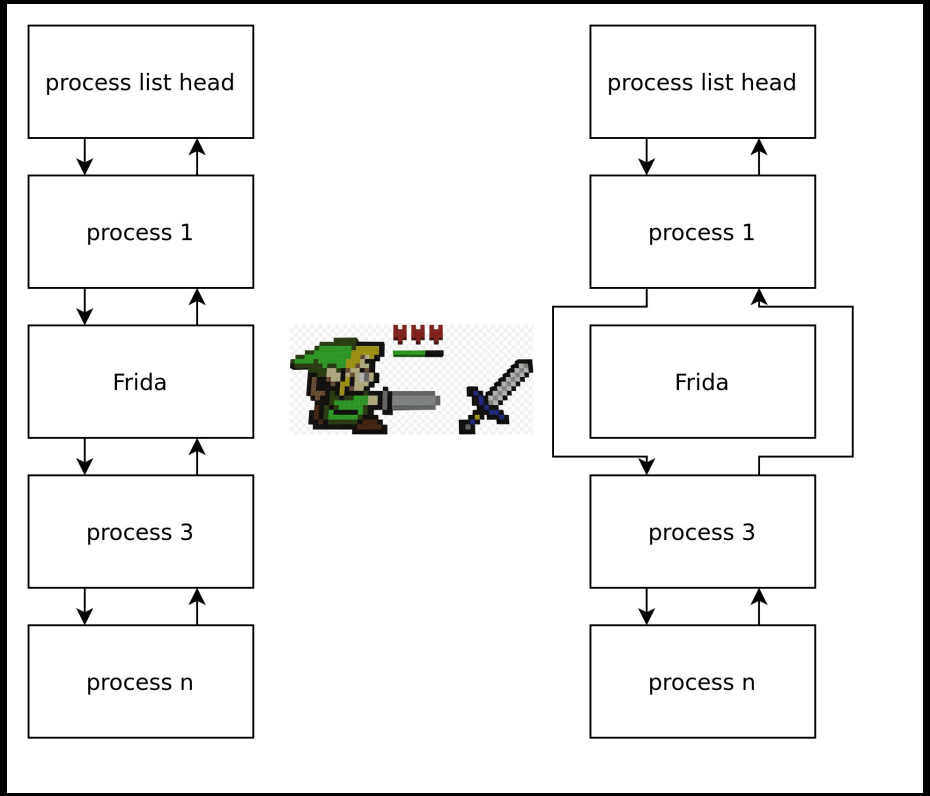
Malware can check for Frida’s processes on the system, but our system can use rootkit techniques to hide analysis components

In this way we can fool the malware into running

Retrieving the process list



Hiding the Frida process



Live demo time!

- Using rootkit techniques to hide the Frida process

Participate!

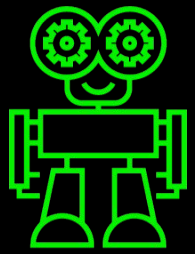
We will soon be looking for alpha testers and early adopters!

Follow us on Twitter: @flipreversing

Join our Telegram: https://t.me/flipreversing

Send us an email: hello@0xepic.com

Thank you, and enjoy chiptunes!



Lars Haukli
@zutle
lars@0xepic.com