Static Configuration Variables

May 20, 2018

Contents

1 Reading the INI file

First, we need to tell Return-Oriented Programming with ROPER: The Secret of the Ooze (ROPER2) where to find the config file. This is set through the environment variable, `ROPER_INI_PATH`.

```
<<bring dependencies into scope>>
    lazy_static! {
        pub static ref ROPER_INI_PATH: String
            = match env::var("ROPER_INI_PATH") {
                    Err(_) => ".roper_config/roper.ini".to_string(),
                    Ok(d)  => d.to_string(),
              };
    }
<<read in the config file>>
<<load and parse the RNG seed>>
<<load the code buffer>>
<<other static variables>>
```

The config file is expected to be in the `ini` file format. See `https://github.com/zonyitoo/rust-ini` for the details on the specification and implementation of the crate used to parse the file. An example of a valid configuration file (using the current defaults) is as follows:

```
# Don't change the encoding
encoding=utf-8

[Binary]
path=/home/oblivia/roper2/elfs/x86/systemd

[RNG]
seed=de ad f0 0d 12 34 56 78 ba ad ba be c0 de fa ce
```

Now, we want to load the entire configuration in as a static variable. To do this, we use the `lazy_static!` macro, which we'll be leaning on pretty heavily in this module.

```
lazy_static! {
    pub static ref INI: Ini = Ini::load_from_file(&*ROPER_INI_PATH)
        .expect(&format!("Failed to load init file from {}",ROPER_INI_PATH);
}
```

Now, let's read and parse the PRNG seed from the `ini` file. The PRNG that we're currently using is Isaac64, from version `0.5.0-pre.1` of the `rand` crate. The seed needs to be an array containing exactly 32 `u8` integers (32 bytes). In the `ini` file, it should appear as a sequence of space-separated hexidecimal octets, as shown above, in the example. It's okay if fewer than 32 bytes are specified in the `ini` file – the remaining bytes will just be padded with zeroes. Any excess bytes will be ignored, so if you can't count, no sweat.

```
pub type RngSeed = [u8; 32];

lazy_static! {
    pub static ref RNG_SEED: RngSeed /* for Isaac64Rng */
        = {
            let rand_sec = INI.section(Some("Random".to_owned()))
                .expect("couldn't find [Random] section in ini file");
            let seed_txt = rand_sec.get("seed")
                .expect("couldn't get seed field from [Random] section");
            let mut seed_vec = [0u8; 32];
            let mut i = 0;
            for octet in seed_txt.split_whitespace() {
                seed_vec[i] = u8::from_str_radix(octet,16).expect("Failed to parse seed");
                i += 1;
                if i == 32 { break };
            }
            while i < 32 { seed_vec[i] = 0 };
            seed_vec
        };
}
```

1.1 Loading the CODE_BUFFER

There's no reason to read the machine code we're targetting into memory more than once, and no reason to tangle up the code with passed pointers to this shared and immutable resource, so we might as well load that as a global as well.

```
lazy_static! {
    pub static ref CODE_BUFFER: Vec<u8>
        = {
            /* first, read the config */
            let bp = match env::var("ROPER_BINARY") {
                Ok(s)  => s,
                Err(_) => {
                  INI.section(Some("Binary"))
                     .expect("Couldn't find Binary section in INI")
                     .get("path")
                     .expect("Couldn't find path field in Binary section of INI")
                }
            };
            //println!("[*] Read binary path as {:?}",bp);
            let path = Path::new(&bp);
            let mut fd = File::open(path).expect(&format!("Can't read binary at {:?}",bp));
            let mut buffer = Vec::new();
            fd.read_to_end(&mut buffer).unwrap();
            buffer
        };
}
```

A more structured view on this same data is supplied by the static global, `MEM_IMAGE`, which is set in the emu::loader module.

2 Other Static Variables

A bit too lazy to document these right now.

```
// set addr size here too. dispense with risc_width() calls, which are confused
lazy_static! {
    pub static ref ARCHITECTURE: Arch
        = {
            let arch_magic = match Object::parse(&CODE_BUFFER).unwrap() {
                Object::Elf(e) => machine_to_str(e.header.e_machine),
                _ => panic!("Binary format unimplemented."),
            };
            match arch_magic {
                "ARM" => Arch::Arm(Mode::Arm),
                "MIPS"  => Arch::Mips(Mode::Be),
                "MIPS_RS3_LE" => Arch::Mips(Mode::Le),
```

```
                "X86_64" => Arch::X86(Mode::Bits64),
                "386" => Arch::X86(Mode::Bits32),
                _  => panic!("arch_magic not recognized!"),
            }
        };
}

lazy_static! {
    pub static ref ADDR_WIDTH: usize
        = {
            match *ARCHITECTURE {
                Arch::X86(_) => 8,
                _ => 4,
            }
        };
}

/// A tiny machine word formatter
#[inline]
pub fn wf<T: PrimInt + fmt::LowerHex>(w: T) -> String {
    match *ARCHITECTURE {
        Arch::X86(Mode::Bits64) => format!("{:016x}", w),
        Arch::X86(Mode::Bits16) => format!("{:04x}", w),
        _ => format!("{:08x}", w),
    }
}

lazy_static! {
    pub static ref KILL_SWITCH: Arc<RwLock<bool>>
        = Arc::new(RwLock::new(false));
}

pub const INPUT_SLOT_FREQ: f32 = 0.1;

lazy_static! {
    pub static ref CROSSOVER_DEGREE: f32 = 0.5;
    /* TODO: Read this from a config file */
}

lazy_static! {
    /* if true, then homologous xbit crossover selects only those slots
     * for which mbit ^ pbit == 1. if false, it selects only those slots
     * for which mbit ^ pbit == 0.
     */
    pub static ref CROSSOVER_XBIT: bool = true;
    /* TODO read from config file */
}

#[derive(Copy, Clone, PartialEq, Eq, Debug)]
```

```
pub enum MaskOp {
    Xor,
    Nand,
    OnePt,
    Uniform,
    And,
    Or,
}

lazy_static! {
    /* TODO read from config file */
    pub static ref CROSSOVER_MASK_COMBINER: MaskOp = MaskOp::And;
}

lazy_static! {
    /* TODO read */
    pub static ref CROSSOVER_MASK_INHERITANCE: MaskOp = MaskOp::Uniform;
}

lazy_static! {
    /* TODO read */
    pub static ref CROSSOVER_MASK_MUT_RATE: f32 = 0.2;
}
```

3 Dependencies

```
extern crate goblin;
extern crate num;
extern crate rand;

use std::fs::File;
use std::sync::{Arc, RwLock};
use std::io::Read;
use std::path::Path;
use std::env;
use std::fmt;

use self::num::PrimInt;
use self::goblin::Object;
use self::goblin::elf::header::machine_to_str;

use emu::loader::{Arch, Mode};
```