Rizin: refactoring of the elf plugin

Introduction

Introduction

My name is Alexis Ehret. I am a student at EPITA. And I have done a GSoC with Rizin.

Rizin

Rizin, a lovecraftian monster

Rizin is a fork of the well-known tool Radare2, that “focus on usability, stability, and working features” and that provide “welcoming environment for developers and users alike”

Rizin, a lovecraftian monster

“Radare2 was created in February 2006”. So more than fifteen year of development without any proper tooling…

Rizin, a lovecraftian monster

```
42sh$ git clone https://github.com/rizinorg/rizin.git
42sh$ cloc rizin
```

Rizin, a lovecraftian monster

- 594682 lines of C
- 92279 lines of C headers

Elf plugin refactoring

Scope

The majority of my work was on librz/bin/p/ and librz/bin/format/elf/.

Solid foundation

The first thing to do was to refactor every function in librz/bin/format/elf/ and to split the elf.c file.

- use Rizin annotations
- use rz_assert_val_if_fail
- refactor or “rewrite” the function if necessary

Solid foundation

- fix sections generated from the dynamic section
- used of the DT_HASH and DT_GNU_HASH
- change the source of trust when parsing symbol versions

DT_HASH

```
struct elf_hash_table { // DT_HASH
        Elf_(Word) nbuckets;
        Elf_(Word) nchains;
        //      Elf_(Word) buckets[nbuckets];
        //      Elf_(Word) chains[nchains];
};
```

DT_GNU_HASH

```c
struct gnu_hash_table { // DT_GNU_HASH
        Elf_(Word) nbuckets;
        Elf_(Word) symoffset;
        Elf_(Word) bloom_size;
        Elf_(Word) bloom_shift;
        //      Elf_(Addr) boom[bloom_size];
        //      Elf_(Word) buckets[nbuckets];
        //      Elf_(Word) chains[];
};
```

Sources of trust

- Sections information shouldn’t be trusted in an executable (EXEC / DYN)
- Sections information should be trusted with relocatable file

Dynamic section

```c
struct rz_bin_elf_dt_dynamic_t {
        HtUU *info;
        RzVector *dt_needed;
};
```

Better RzBuffer

DEMO

How to store segment information?

```c
typedef struct Elf_(rz_bin_elf_segment_t) {
        Elf_(Phdr) data;
        bool is_valid;
}
RzBinElfSegment;
```

How to store section information?

```
typedef struct rz_bin_elf_section_t {
        ut32 flags;
        ut32 info;
        ut32 link;
        ut32 type;
        ut64 align;
        ut64 offset;
        ut64 rva;
        ut64 size;
        char *name;
        bool is_valid;
} RzBinElfSection;
```

String table

The string table are now correctly parsed and checked before any string can’t be used. Which helped removed some hard coded limit.

Symbols and import

DEMO

Configuration variables

- elf.load.sections
- elf.checks.segments
- elf.checks.sections

Thumb addresses

DEMO

Sources

Sources

- GSoC sources
- GSoC submission

Conclusion

Conclusion

The GSoC was an incredible source of motivation to contribute to the Open-Source community. And it helped me improve my knowledge of elf internals. I would like to thank my mentors Anton Kochkov and Florian Märkl for their help during the GSoC.