Solana Core

Audit

Presented by:

OtterSec contact@osec.io

Harrison Green hgarrereyn@osec.io

Robert Chen notdeghost@osec.io

Contents

Appendices

01 | Executive Summary

Overview

OtterSec performed an assessment of the dependency-free ELF Parser introduced in the RBPF repository. This assessment was conducted between July 11th and July 22nd, 2022.

The audit scope consisted of a review of pull request #348.

Executive Summary

During the audit, we focused on the following scopes:

1. Issues which could lead to program misbehavior through normal usage
2. Issues which could affect the security of the Solana runtime (e.g. denial-of-service or excess resource allocation)
3. Issues which could break backward-compatibility with existing Solana programs

Areas that were of lesser concern were:

1. Inconsistencies with the ELF specification
2. Differential behavior between the two parser variants using specially crafted ELF files

02 | Architecture

In this section we analyze the architecture of the RBPF ELF Parser and discuss its usage in the Solana runtime.

ElfParser

For backwards compatibility, the legacy `GoblinParser` is supported alongside the new ELF parser. Both parsers implement the `ElfParser` trait which provides a shared interface for the `Executable` struct (02). At runtime, the `Executable` struct uses either the `GoblinParser` or `NewParser` based on the `new_elf_parser` configuration parameter.

Internally, the `NewParser` uses the `Elf64` struct to perform parsing.

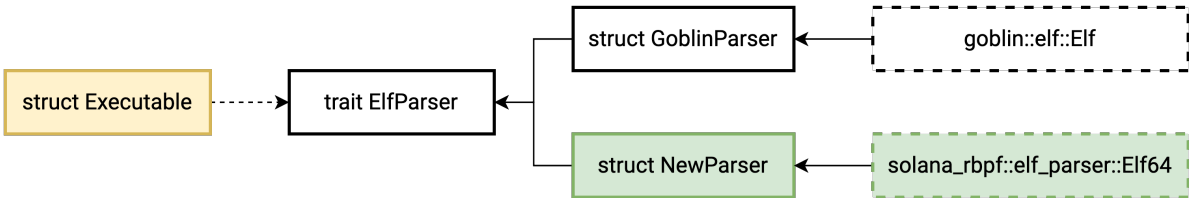

Figure 02.1: The ElfParser trait provides a shared interface for the legacy `GoblinParser` and new `NewParser`.

Usage in Solana Runtime

In Solana, ELF processing is handled by the `bpf_loader` program. This program is responsible for loading and executing BPF programs stored in program accounts. The first time each program is loaded into an `Executor`, the `bpf_loader` invokes `Executable::from_elf`, the main entrypoint for loading an ELF file.

During parsing, the ELF is first parsed with either the `GoblinParser` or `NewParser`. In Figure 02.2, we show the `NewParser` case (assuming the `new_elf_parser` configuration parameter is set).

The parser is responsible for extracting section metadata from the ELF file such as: the executable header, segment and section headers, and dynamic symbols and relocations.

With metadata extracted, `Executable::load_with_parser` performs validation, applies relocations and remaps the read-only sections into the memory layout that Solana expects.

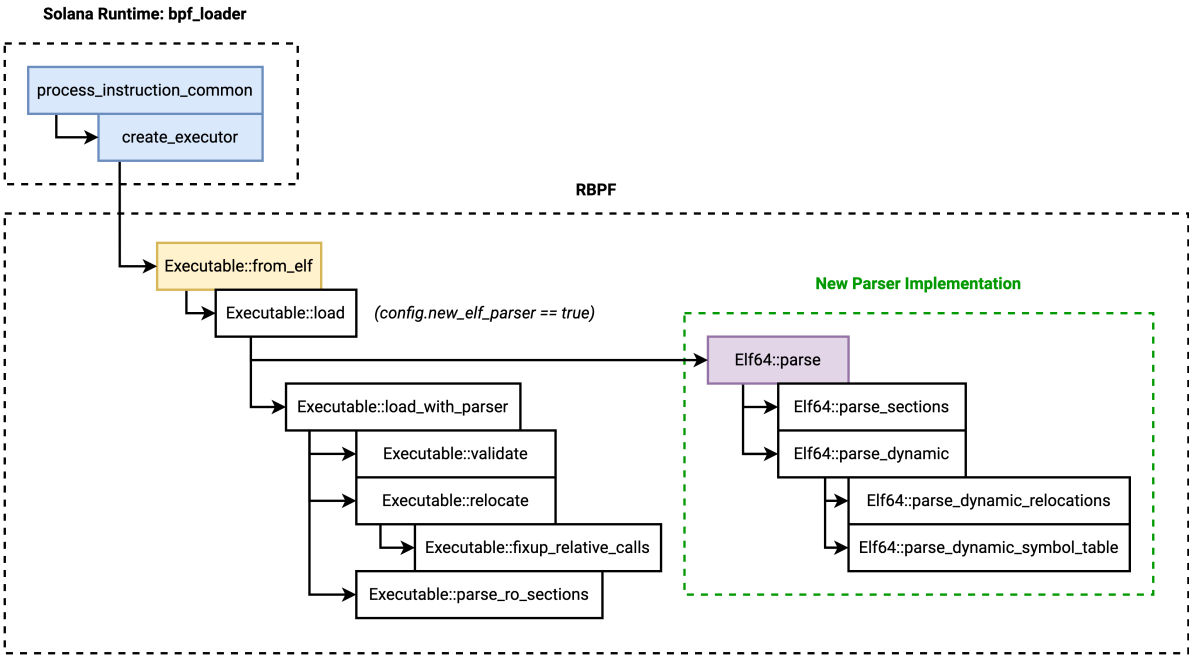

Figure 02.2: ELF parsing and loading within the larger Solana runtime.

Security Considerations

Metadata validation

ELF metadata validation (for example ensuring sections do not overlap and do not specify ranges outside of the file data) is primarily handled by the `Elf64` library.

Additionally, out of bounds indexing into the `elf_bytes` backing data automatically throws a runtime error.

Metadata corruption

The `Executable` struct performs dynamic relocations which can effectively rewrite any data in the ELF file. However, the `Elf64` struct contains references to immutable `elf_bytes` slices while relocations (and call target fixups) apply to a separate, mutable copy of the backing `elf_bytes`. This separation ensures that metadata extracted during the first stage is *immutable* for the lifetime of the executable.

Executable runtime

The `Executable` struct is only responsible for loading the ELF bytes into the memory layout expected by the Solana runtime, for example ensuring the read-only section is in the `0x100000000-0x200000000` region.

The `Executable` struct is *not* responsible for validating that the text section contains valid or legal BPF instructions. Code validation is performed separately under the assumption that *any* sequence of bytes may be provided to the BPF interpreter or JIT compiler.

Resource allocation

Care must be taken that the resources used during parsing and loading (memory, time, etc...) should not far exceed the cost of invoking the runtime code. For example, a small ELF file which could cause a large memory allocation during parsing could be used as a denial-of-service attack on the Solana runtime.

03 | Findings

Overall, we report 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.

The below chart displays the findings by severity.

Severity	Count
Critical	0
High	0
Medium	1
Low	1
Informational	4

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have **immediate** security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-ELF-ADV-00	Medium	Resolved	Off-by-one error in section size check
OS-ELF-ADV-01	Low	Resolved	Read-only region can overlap stack

OS-ELF-ADV-00 [med] [resolved] | Off-by-one error in section size check

Description

The ELF Parser incorrectly reports sections as `OutOfBounds` if the section ends at the end of the ELF file.

rbpf@7f801c2:src/elf_parser/mod.rs RUST

```
170 if section_range.end >= elf_bytes.len() {
171     return Err(ElfParserError::OutOfBounds);
172 }
```

The impact of this bug is that a valid Solana ELF may be rejected by the Solana runtime.

Remediation

Modify the range check to use > instead of >=.

Patch

Issue 365; fixed in 30e2c96.

OS-ELF-ADV-01 [low] [resolved] | Read-only region can overlap stack

Description

The readonly region (`0x1xxxxxxxx`) can extend into the stack region (`0x2xxxxxxxx`) due to faulty bounds when initializing the `Executable`.

During loading, the read-only sections in the ELF file (e.g. `.text`, `.rodata`, ...) are accumulated and merged into a single byte sequence which is stored as a region in the runtime `MemoryMap` used by the BPF virtual machine.

This region *should* fall in the `MM_PROGRAM_START` region (`0x100000000-0x200000000`).

However, sections that specify a virtual address just below the region boundary of `0x200000000` (e.g. `0x1fffffff0`) can extend into the next region which happens to be the stack.

The incorrect checks happen in two places. First in `Executable::load_with_parser` while validating the `.text` section:

rbpf@7f801c2:src/elf.rs RUST

```
448 if (config.reject_broken_elfs
449     && !config.optimize_rodata
450     && text_section.sh_addr() != text_section.sh_offset())
451     || text_section_info.vaddr > ebpf::MM_STACK_START
452 {
453     return Err(ElfError::ValueOutOfBounds);
454 }
```

Second in `Executable::parse_ro_sections` while validating other sections:

rbpf@7f801c2:src/elf.rs RUST

```
766 if (config.reject_broken_elfs && invalid_offsets) || vaddr >
    ↪  ebpf::MM_STACK_START {
767     return Err(ElfError::ValueOutOfBounds);
768 }
```

Since memory map lookup uses the high 32 bits as an index, the overlapping part is “shadowed” by the stack and is effectively un-addressable. Note however that memcpy/memset/... syscalls can address this memory as long as the base address is less than `0x200000000`.

The impact of this bug is that programs with overlapping read-only/stack regions may behave incorrectly when operating on data read from the overlapping region (since this data will actually come from the stack region).

Remediation

These checks should be (`vaddr + size > MM_STACK_START`) instead of (`vaddr > MM_STACK_START`).

Patch

Fixed in 371.

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they do represent antipatterns and could introduce a vulnerability in the future.

ID	Status	Description
OS-ELF-SUG-00	Resolved	Use checked arithmetic instead of saturating
OS-ELF-SUG-01	Resolved	ELF Section 0 should be SHT_NULL
OS-ELF-SUG-02	Resolved	Some dynamic attributes are ignored
OS-ELF-SUG-03	Resolved	Dynamic section metadata is retrieved incorrectly

OS-ELF-SUG-00 [resolved] | Use checked arithmetic instead of saturating

Description

Use of saturating arithmetic (`saturating_add`, `saturating_sub`, `saturating_mul`, etc...) causes values to "saturate" at the maximum or minimum possible representations.

During ELF parsing, saturating arithmetic is used to perform arithmetic on offsets, virtual address, and sizes. In these cases, it rarely makes sense to continue parsing. Rather, saturating events should manifest as errors caught during parsing.

During our analysis, we did not find any vulnerabilities related to the use of saturating arithmetic. In all cases explored, the bounds of each operation were enforced by auxiliary code before or after the arithmetic. For example, out-of-bounds ranges caused by saturating events will throw runtime errors when used as a slice index. In other cases, the auxiliary constraints are less obvious; for example, performed as part of a subroutine call prior to arithmetic.

As a defensive coding measure, the use of checked arithmetic could help prevent introducing future bugs by accidentally changing or removing a bounds check.

For example, in `parse_dynamic_relocations`, the file offset for the dynamic table is computed (in one path) by identifying the enclosing segment and converting the virtual address to a physical offset. The first `saturating_sub` will not saturate because `program_header_for_vaddr` will not return a program header with a `p_vaddr` greater than `vaddr`. The second `saturating_sub` can saturate, however the resulting offset will produce an out-of-bounds slice later in the function.

rbpf@7f801c2:src/elf_parser/mod.rs RUST

```
332 let offset = if let Some(program_header) =
        ↪ self.program_header_for_vaddr(vaddr) {
333     vaddr
334         .saturating_sub(program_header.p_vaddr)
335         .saturating_add(program_header.p_offset)
336 } else {
```

Patch

Issue 366; fixed in f6c483d.

OS-ELF-SUG-01 [resolved] | ELF Section 0 should be SHT_NULL

Description

Section number 0 in the ELF section header table is required to be a null sentinel section with type `SHT_NULL` according to the ELF specification. (https://refspecs.linuxfoundation.org/elf/elf.pdf, 1-13). Runtime loaders should ignore this section.

Currently in the RBPF ELF Parser implementation, this section index is used like any other section and can be non-null.

Patch

Issue 364; fixed in 79209e1.

OS-ELF-SUG-02 [resolved] | Some dynamic attributes are ignored

Description

The ELF parser allows ELFs that set a wide range (`DT_{x}`) attributes in the dynamic table (0-35). However, it silently ignores all except: `DT_REL`, `DT_RELENT`, `DT_RELSZ`, and `DT_SYMTAB`. For example, attributes that are ignored include `DT_INIT_ARRAY` which may specify a list of initialization functions.

ELFs built with a non-standard toolchain may run into hard-to-debug errors with unexpected behavior if these dynamic attributes are silently ignored.

This issue is likely not a factor when using the official Solana build toolchain.

Note also that this issue may be hard to fix while maintaining backwards-compatability with potentially invalid legacy programs.

Patch

Wontfix: not a general purpose ELF loader; intended behavior is to only parse dynamic entries that are used.

OS-ELF-SUG-03 [resolved] | Dynamic section metadata is retrieved incorrectly

Description

In `Elf64::parse_sections`, section lookup is performed by matching special section names, e.g. `.dynsym`, `.symtab`, `.strtab`.

These sections *should* be retrieved via the metadata in the dynamic table such as the `DT_STRTAB` and `DT_SYMTAB` pointers. This retrieval method is also how the legacy Goblin ELF parser finds these sections.

In normal cases, the dynamic table metadata should be in agreement with the specially named sections; therefore, this retrieval method will not cause inconsistencies.

Similar to OS-ELF-SUG-02 this issue may be hard to fix while maintaining backwards-compatability with potentially invalid legacy programs.

Patch

Tracked in 367.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities which immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority/token account validation • Rounding errors on token transfers
High	Vulnerabilities which could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities which could lead to denial of service scenarios or degraded usability. Examples: • Malicious input cause computation limit exhaustion • Forced exceptions preventing normal use
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation • Uncaught Rust errors (vector out of bounds indexing)