Solana JIT
Lessons learned from fuzzing a smart-contract compiler
Kraken Security Labs
Thomas Roth

- Security research division of Kraken
Thanks!

• The Solana team
• secret.club's
• The AFL++ team
Solana

- "High performance" blockchain
- Runs smart-contracts
  - That are written in C/C++/Rust
- Proof-of-stake / proof-of-history
Solana

Validator  Validator  Validator  Validator  Validator
Solana
Solana

Leader

Validator  Validator  Validator  Validator

TX
Solana

Leader

Validator

Validator

Validator

Validator

TX

TX
Solana

Leader

Validator
Validator
Validator
Validator

TX
TX
TX
Solana
Solana
Solana
Solana

Leader

State 0x23432

TX  TX  TX

Validator  Validator  Validator  Validator
Solana
Solana

![Diagram showing Solana architecture with a leader and validators.]
Solana
Smart-contracts
Solana
Smart-contracts
Solana
Smart-contracts

Compiler
Solana
Smart-contracts
eBPF
extended Berkely Packet Filter

- eBPF
  - 64-bit RISC machine
  - Fixed-length instructions
  - Designed for packet-filtering
  - Virtual machine (with JIT) in the Linux kernel
eBPF
extended Berkely Packet Filter

- 10 registers + frame-pointer
- 64-bit wide instructions
  - Two different encodings (one with appended 64-bit immediate)
- 8 different instruction classes

<table>
<thead>
<tr>
<th>class</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPF_LD</td>
<td>0x00</td>
<td>non-standard load operations</td>
</tr>
<tr>
<td>BPF_LDX</td>
<td>0x01</td>
<td>load into register operations</td>
</tr>
<tr>
<td>BPF_ST</td>
<td>0x02</td>
<td>store from immediate operation</td>
</tr>
<tr>
<td>BPF_STX</td>
<td>0x03</td>
<td>store from register operation</td>
</tr>
<tr>
<td>BPF_ALU</td>
<td>0x04</td>
<td>32-bit arithmetic operations</td>
</tr>
<tr>
<td>BPF_JMP</td>
<td>0x05</td>
<td>64-bit jump operations</td>
</tr>
<tr>
<td>BPF_JMP32</td>
<td>0x06</td>
<td>32-bit jump operations</td>
</tr>
<tr>
<td>BPF_ALU64</td>
<td>0x07</td>
<td>64-bit arithmetic operations</td>
</tr>
</tbody>
</table>
eBPF extended

- 10 registers
- 64-bit wide instructions
- Two different encodings
- 8 different instruction classes

| CVE-2022-23222 | kernel/bpf/verifier.c in the Linux kernel through 5.15.14 allows local users to gain privileges because the `bpf_map_update_elem()` function did not check if the input `key` is null and therefore, arbitrary code execution. This issue was fixed via commit 0cb64c4f3714e ("bpf: Fix a verifier failure due to null key") (5.10-rc1). |
| CVE-2022-0500 | A flaw was found in unrestricted eBPF usage by the BPF_BTF_LOAD, leading to a possible out-of-bounds local user crash or privilege escalation on the system. |
| CVE-2022-0264 | A vulnerability was found in the Linux kernel's eBPF verifier when handling internal data structures of eBPF code to the kernel can use this to leak internal kernel memory details by defeating some of the store from immediate operations. |
| CVE-2021-4202 | A use-after-free flaw was found in nci_request in net/nfc/nci/core.c in NFC Controller Interface (NFC) while the device is getting removed, leading to a privilege escalation problem. |
| CVE-2021-41864 | prealloc elems and freelists in kernel/bpf/stackmap.c in the Linux kernel before 5.14.12 allows | 32-bit arithmetic operations |
| CVE-2021-4135 | A memory leak vulnerability was found in the Linux kernel's eBPF for the Simulated networking driver local user could use this flaw to leak unauthorized access to some data. |
| CVE-2021-4001 | A race condition was found in the Linux kernel's eBPF verifier between bpf_map_update_elem and (cap_sys_admin or cap_bpf) can modify the frozen mapped address space. This flaw affects kernels | 64-bit jump operations |
| CVE-2021-3490 | The eBPF ALU32 bounds tracking for bitwise ops (AND, OR and XOR) in the Linux kernel did not check and therefore, arbitrary code execution. This issue was fixed via commit 049c4e13714e ("bpf: Fix a verifier failure due to null key") (5.10-rc1). |
| CVE-2021-3489 | The eBPF RINGBUF bpf_ringbuf_reserve() function in the Linux kernel did not check that the allocate buffer and therefore, arbitrary code execution. This issue was fixed via commit 4bb81fcee5ae ("bpf: Implement BPF ring | 32-bit jump operations |
| CVE-2021-34866 | This vulnerability allows local attackers to escalate privileges on affected installations of Linux Kernel order to exploit this vulnerability. The specific flaw exists within the handling of eBPF programs. An attacker can leverage this vulnerability to escalate privileges and execute arbitrary code. |
| CVE-2021-31440 | This vulnerability allows local attackers to escalate privileges on affected installations of Linux Kernel order to exploit this vulnerability. The specific flaw exists within the handling of eBPF programs. An attacker can leverage this vulnerability to escalate privileges and execute arbitrary code in the | 64-bit arithmetic operations |
| CVE-2021-20320 | A flaw was found in s390 eBPF JIT in bpf_jit_insn in arch/s390/net/bpf_jit_comp.c in the Linux kernel to confidentiality problem. |
| CVE-2021-20268 | An out-of-bounds access was found in the Linux kernel's implementation of the eBPF code v | local user to crash the system or possibly escalate their privileges. The highest threat from this vulnerability is |
| CVE-2020-16-4557 | The replace_map_fd with_map_ptr function in kernel/bpf/verifier.c in the Linux kernel before 4.5 service (use-after-free) via crafted BPF instructions that reference an incorrect file descriptor. |
| CVE-2016-2383 | The adjust_branches function in kernel/bpf/verifier.c in the Linux kernel before 4.5 does not cons | memory by creating a packet filter and then loading crafted BPF instructions. |
Solana
Smart-contracts

Compiler

eBPF ELF
Solana
Smart-contracts
Solana
Smart-contracts
Solana

Smart-contracts

```solidity
contract Foo {
    string public text = "Hello";

    function bla() {
        text = "test";
    }
}
```
Solana
Smart-contracts

```solidity
contract Foo {
    string public text = "Hello";

    function bla() {
        text = "test";
    }
}
```
contract Foo {
    string public text = "Hello";

    function bla() {
        text = "test";
    }
}
contract Foo {
    string public text = "Hello";

    function bla() {
        text = "test";
    }
}
Solana
Smart-contracts
Solana
Smart-contracts

Smart Contract

Stateless
Solana

Smart-contracts
Solana
Smart-contracts

Accounts

Smart Contract

Accounts

Accounts

Accounts need to pay rent to not get deleted
Solana
Virtual machine

eBPF ELF
Solana
Virtual machine

eBPF ELF

Shared object
Solana
Virtual machine

- eBPF ELF
- Shared object
- No writable global variables
Solana
Virtual machine

Program code loaded at 0x100000000
Stack starts at 0x200000000
Heap starts at 0x300000000
Program input at 0x400000000
Solana
Virtual machine

Program code loaded at 0x100000000
Stack starts at 0x200000000
Heap starts at 0x300000000
Program input at 0x400000000
(No ASLR)
Solana
Virtual machine
Solana
Virtual machine

- eBPF ELF
- Solana VM
Solana
Virtual machine

eBPF ELF

Solana VM

Interpret the bytecode
Solana
Virtual machine

Interpret the bytecode
Or
Solana
Virtual machine

- eBPF ELF
  - Solana VM

Interpret the bytecode
Or
Just-in-time compilation
Solana
Virtual machine

- eBPF ELF
  - Solana VM

  Interpret the bytecode

  Or

  Just-in-time compilation
Solana

Virtual machine

eBPF ELF

Solana VM

 Interpret the bytecode

Or

Just-in-time compilation
(Compile the bytecode to native machine code at runtime)
Solana
Virtual machine

Just-in-time compilation

eBPF ELF

Solana VM
Solana
Virtual machine

Just-in-time compilation
Requires writable & (eventually) executable pages
Solana

Virtual machine

Just-in-time compilation

- Requires writable & (eventually) executable pages
- Needs to bring its own memory safety
Solana
Virtual machine

Just-in-time compilation
Requires writable & (eventually) executable pages
Needs to bring its own memory safety
Fairly difficult to implement securely
Solana

Virtual machine
Solana
Virtual machine

Rust (user-space) virtual machine for eBPF

Description

This is a fork of RBPF by Quentin Mornet.

This crate contains a virtual machine for eBPF program execution. BPF, as in Berkeley Packet Filter, is an assembly-like language initially developed for BSD systems, in order to filter packets in the kernel with tools such as tcpdump so as to avoid useless copies to user-space. It was ported to Linux, where it evolved into eBPF (extended BPF), a faster version with more features. While BPF programs are originally intended to run in the kernel, the virtual machine of this crate enables running it in user-space applications; it contains an interpreter, an x86_64 JIT-compiler for eBPF programs, as well as an assembler, disassembler and verifier.

The crate is supposed to compile and run on Linux, MacOS X, and Windows, although the JIT-compiler does not work with Windows at this time.
Solana

Virtual machine

This is a fork of RBPF by Quentin Mornet.

This crate contains a virtual machine for eBPF program execution. BPF, as in Berkeley Packet Filter, is an assembly-like language initially developed for BSD systems, in order to filter packets in the kernel with tools such as tcpdump so as to avoid useless copies to user-space. It was ported to Linux, where it evolved into eBPF (extended BPF), a faster version with more features. While BPF programs are originally intended to run in the kernel, the virtual machine of this crate enables running it in user-space applications; it contains an interpreter, an x86-64 JIT-compiler for eBPF programs, as well as an assembler, disassembler and verifier.

The crate is supposed to compile and run on Linux, MacOS X, and Windows, although the JIT-compiler does not work with Windows at this time.
Solana
Virtual machine

Testnet - v1.6.0
This is a Testnet release. It is not recommended for Mainnet Beta

New Features
- BPF Rust toolchain upgraded to Rust 1.50
- BPF JIT is now enabled by default (add the --no-bpf-jit flag to disable the JIT)
- New solana-validator subcommands to improve quality of life for node operators

Breaking Changes
- solana program deploy commands now only return json when specified via --output parameter
- solana vote-account's output is tweaked for easier understanding
- solana catchup displays more precise ETA by computing overall average progress (contributed by @diman-io)

Known Issues
- 1.6.0 Rust crates have been yanked due to source backwards compatibility issues with the 1.5.x Rust crates

Assets 12
Solana
Virtual machine

Testnet - v1.6.0

This is a Testnet release. It is not recommended for Mainnet Beta

New Features
- BPF Rust toolchain upgraded to Rust 1.50
- BPF JIT is now enabled by default (add the --no-bpf-jit flag to disable the JIT)
- New `solana-validator` subcommands to improve quality of life for node operators

Breaking Changes
- `solana program deploy` commands now only return json when specified via `--output` parameter
- `solana vote-account`'s output is tweaked for easier understanding
- `solana catchup` displays more precise ETA by computing overall average progress (contributed by @diman-io)

Known Issues
- 1.6.0 Rust crates have been yanked due to source backwards compatibility issues with the 1.5.x Rust crates

Testnet - v1.5.6
Solana

Virtual machine

This is a fork of RBPF by Quentin Monnet.

This crate contains a virtual machine for eBPF program execution. BPF, as in Berkeley Packet Filter, is an assembly-like language initially developed for BSD systems, in order to filter packets in the kernel with tools such as tcpdump so as to avoid useless copies to user-space. It was ported to Linux, where it evolved into eBPF (extended BPF), a faster version with more features. While BPF programs are originally intended to run in the kernel, the virtual machine of this crate enables running it in user-space applications; it contains an interpreter, an x86_64 JIT-compiler for eBPF programs, as well as an assembler, disassembler and verifier.

The crate is supposed to compile and run on Linux, MacOS X, and Windows, although the JIT-compiler does not work with Windows at this time.
Solana

Virtual machine

This is a fork of RBPF by Quentin Monnet.

This crate contains a virtual machine for eBPF program execution. BPF, as in Berkeley Packet Filter, is an assembly-like language initially developed for BSD systems, in order to filter packets in the kernel with tools such as tcpdump so as to avoid useless copies to user-space. It was ported to Linux, where it evolved into eBPF (extended BPF), a faster version with more features. While BPF programs are originally intended to run in the kernel, the virtual machine of this crate enables running it in user-space applications; it contains an interpreter, an x86_64 JIT-compiler for eBPF programs, as well as an assembler, disassembler and verifier.

The crate is supposed to compile and run on Linux, MacOS X, and Windows, although the JIT-compiler does not work with Windows at this time.
Solana

Virtual machine

To do list

- Implement some traits (Clone, Drop, Debug are good candidates).
- Provide built-in support for user-space array and hash BPF maps.
- Improve safety of JIT-compiled programs with runtime memory checks.
- Add helpers (some of those supported in the kernel, such as checksum update, could be helpful).
- Improve verifier. Could we find a way to directly support programs compiled with clang?
- Maybe one day, tail calls?
- JIT-compilers for other architectures?
- ...

License

Following the effort of the Rust language project itself in order to ease integration with other projects, the rbpf crate is distributed under the terms of both the MIT license and the Apache License (Version 2.0).

See LICENSE-APACHE and LICENSE-MIT for details.

Inspired by

- uBPF, a C user-space implementation of an eBPF virtual machine, with a JIT-compiler and disassembler (and also including the assembler from the human-readable form of the instructions, such as in Nov 16, Briefly.), by Rich Lane for Big Switch Networks (2015)

- Building a simple JIT in Rust, by Jonathan Turner (2015)
Solana

Virtual machine

---

**To do list**

- Implement some traits (Clone, Drop, Debug are good candidates).
- Provide built-in support for user-space array and hash BPF maps.
- **Improve safety of JIT-compiled programs with runtime memory checks.**
- Add helpers (some of those supported in the kernel, such as checksum update, could be helpful).
- Improve verifier. Could we find a way to directly support programs compiled with clang?
- Maybe one day, tail calls?
- JIT compilers for other architectures?
- ...

**License**

Following the effort of the Rust language project itself in order to ease integration with other projects, the rbpf crate is distributed under the terms of both the MIT license and the Apache License (Version 2.0).

See LICENSE-APACHE and LICENSE-MIT for details.

**Inspired by**

- **uBPF**, a C user-space implementation of an eBPF virtual machine, with a JIT-compiler and disassembler (and also including the assembler from the human-readable form of the instructions, such as in `nov 16, 8x1337`), by Rich Lane for Big Switch Networks (2015)

  **Building a simple JIT in Rust**, by Jonathan Turner (2015)
What about program validation?

The “verifier” of this crate is very short and has nothing to do with the kernel verifier, which means that it accepts programs that may not be safe. On the other hand, you probably do not run this in a kernel here, so it will not crash your system. Implementing a verifier similar to the one in the kernel is not trivial, and we cannot “copy” it since it is under GPL license.

What about safety then?

Rust has a strong emphasis on safety. Yet to have the eBPF VM work, some `unsafe` blocks of code are used. The VM, taken as an eBPF interpreter, can return an error but should not crash. Please file an issue otherwise.

As for the JIT-compiler, it is a different story, since runtime memory checks are more complicated to implement in assembly. It will crash if your JIT-compiled program tries to perform unauthorized memory accesses. Usually, it could be a good idea to test your program with the interpreter first.

Oh, and if your program has infinite loops, even with the interpreter, you’re on your own.

Caveats

- This crate is under development and the API may be subject to change.
- The JIT compiler produces an unsafe program: memory access are not tested at runtime (yet). Use with caution.
- Contrary to the interpreter, if a division by 0 is attempted, the JIT program returns `xxffffffffffffffff` and exits cleanly (no `Error` returned). This is because the author has not found how to return something more informative in the current assembly framework.
What about program validation?

The “verifier” of this crate is very short and has nothing to do with the kernel verifier, which means that it accepts programs that may not be safe. On the other hand, you probably do not run this in a kernel here, so it will not crash your system. Implementing a verifier similar to the one in the kernel is not trivial, and we cannot “copy” it since it is under GPL license.

What about safety then?

Rust has a strong emphasis on safety. Yet to have the eBPF VM work, some `unsafe` blocks of code are used. The VM, taken as an eBPF interpreter, can return an error but should not crash. Please file an issue otherwise.

As for the JIT-compiler, it is a different story, since runtime memory checks are more complicated to implement in assembly. If your JIT-compiled program tries to perform unauthorized memory accesses, it will crash. Usually, it could be a good idea to test your program with the interpreter first.

Oh, and if your program has infinite loops, even with the interpreter, you’re on your own.

Caveats

- This crate is under development and the API may be subject to change.
- The JIT compiler produces an unsafe program: memory access are not tested at runtime (yet). Use with caution.
- Contrary to the interpreter, if a division by 0 is attempted, the JIT program returns `0xfffffffffffffff` and exits cleanly (no `Error` returned). This is because the author has not found how to return something more meaningful in the context of assembly for.
What about program validation?

The “verifier” of this crate is very short and has nothing to do with the kernel verifier, which means that it accepts programs that may not be safe. On the other hand, you probably do not run this in a kernel here, so it will not crash your system. Implementing a verifier similar to the one in the kernel is not trivial, and we cannot “copy” it since it is under GPL license.

What about safety then?

Rust has a strong emphasis on safety. Yet to have the eBPF VM work, some `unsafe` blocks of code are used. The VM, taken as an eBPF interpreter, can return an error but should not crash. Please file an issue otherwise.

As for the JIT-compiler, it’s a different story, since runtime memory checks are more complicated to implement in assembly. **It will crash if your JIT-compiled program tries to perform unauthorized memory accesses.** Usually, it could be a good idea to test your program with the interpreter first.

Oh, and if your program has infinite loops, even with the interpreter, you’re on your own.

Caveats

- This crate is under development and the API may be subject to change.

- **The JIT compiler produces an unsafe program: memory access are not tested at runtime (yet). Use with caution.**

- Contrary to the interpreter, if a division by 0 is attempted, the JIT program returns `0xffffffffffffffff` and exits cleanly (no `Error` returned). This is because the author has not found how to return something more informative to the external assembler yet.
Let's get fuzzing!
Fuzzing the Solana VM
Fuzzing the Solana VM

Simpe eBPF program → Mutate it semi-randomly
Fuzzing the Solana VM

1. Simple eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
Fuzzing the Solana VM

1. Simple eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
4. Collect crashes
Fuzzing the Solana VM

1. Simpe eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
4. Collect crashes (Hopefully)
Fuzzing the Solana VM

1. Simpe eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
4. Collect crashes
5. Analyze crashes

(Hopefully)
Fuzzing the Solana VM

Simpe eBPF program
Fuzzing the Solana VM

This branch is 366 commits ahead, 3 commits behind qmonnet:master.
Fuzzing the Solana VM

Simple eBPF program
Fuzzing the Solana VM

Simpe eBPF program

How can we analyze them?
Fuzzing the Solana VM

How can we analyze them?

No good reverse-engineering tooling for Solana binaries...
Analyzing Solana binaries
Analyzing Solana binaries
Analyzing Solana binaries
helloworld.so has not been analyzed. Would you like to analyze it now?
Verifiable Builds
Verifiable builds
Verifiable builds

Contract Source Code Verified (Exact Match)

Contract Name: BoredApeYachtClub

Compiler Version: v0.7.0+commit.9e61f92b
Verifiable builds

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>cndy3Z4yapfJhml3SHjip5exZXqR3z33tTHzeNM2gKZ</td>
</tr>
<tr>
<td>Address Label</td>
<td>NFT Candy Machine Program V2</td>
</tr>
<tr>
<td>Balance (sOL)</td>
<td>0.0014144</td>
</tr>
<tr>
<td>Executable</td>
<td>Yes</td>
</tr>
<tr>
<td>Executable Data</td>
<td>8dgyV6ZzrHl:pN1fMe3Zxr5lEQuE9qz3N8ho6eAN9ka</td>
</tr>
<tr>
<td>Upgradeable</td>
<td>Yes</td>
</tr>
<tr>
<td>Verifiable Build Status</td>
<td>Anchor: Unverified</td>
</tr>
<tr>
<td>Security.txt</td>
<td>Program has no security.txt</td>
</tr>
<tr>
<td>Last Deployed Slot</td>
<td>135, 858, 186</td>
</tr>
<tr>
<td>Upgrade Authority</td>
<td>CandyWZGOKRekMTExtzC7M4XLUBz8x3Ksa0xWj6EMUVu</td>
</tr>
</tbody>
</table>
### Verifiable builds

#### Account

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>cn3yZ4yapfJ8ml13hi05exZkgK8k33thTzeNMe29kZ</td>
</tr>
<tr>
<td>Address Label</td>
<td>NFT Candy Machine Program V2</td>
</tr>
<tr>
<td>Balance (SOL)</td>
<td>0.00114144</td>
</tr>
<tr>
<td>Executable</td>
<td>Yes</td>
</tr>
<tr>
<td>Executable Data</td>
<td>Bdgy2KZhrhpN1BMcZxxr5r1QttE9pztN8ho6eNA9kz</td>
</tr>
<tr>
<td>Upgradeable</td>
<td>Yes</td>
</tr>
<tr>
<td>Verifiable Build Status (experimental)</td>
<td>Anchor: Unverified</td>
</tr>
<tr>
<td>Security.txt</td>
<td>Program has no security.txt</td>
</tr>
<tr>
<td>Last Deployed Slot</td>
<td>135, 858, 186</td>
</tr>
<tr>
<td>Upgrade Authority</td>
<td>CandyMcZGCKPmIMETExh2cXTM4XLUbz8xJKsreWyjEMVU</td>
</tr>
</tbody>
</table>
Verifiable builds
Fuzzing the Solana VM

- Simpe eBPF program
- Mutate it semi-randomly
- Execute it in the Solana VM
Fuzzing the Solana VM

1. Simple eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
Fuzzing the Solana VM

1. Simpe eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
fn main() {}
fn main() {
    let args: Vec<String> = env::args().collect;
    let query = &args[1];
    let mut file = File::open(query).unwrap();
    let mut elf = Vec::new();
    file.read_to_end(&mut elf).unwrap();
    let data: &[u8] = &elf[..];
}
fn main() {
    let args: Vec<String> = env::args().collect();
    let query = &args[1];
    let mut file = File::open(query).unwrap();
    let mut elf = Vec::new();
    file.read_to_end(&mut elf).unwrap();
    let data: &[u8] = &elf[..];

    let mut executable = Executable::<UserError, TestInstructionMeter>::from_elf(
        &data,
        Config::default(),
        SyscallRegistry::default(),
    ).unwrap();
fn main() {
    let args: Vec<String> = env::args().collect();
    let query = &args[1];
    let mut file = File::open(query).unwrap();
    let mut elf = Vec::new();
    file.read_to_end(&mut elf).unwrap();
    let data: &[u8] = &elf[..];

    let mut executable = Executable::<UserError, TestInstructionMeter>::from_elf(  
        &data,
        Config::default(),
        SyscallRegistry::default(),
    ).unwrap();

    Executable::<UserError, TestInstructionMeter>::jit_compile(&mut executable).unwrap();
}
fn main() {
    let args: Vec<String> = env::args().collect();
    let query = &args[1];
    let mut file = File::open(query).unwrap();
    let mut elf = Vec::new();
    file.read_to_end(&mut elf).unwrap();
    let data: &[u8] = &elf[..];

    let mut executable = Executable::<UserError, TestInstructionMeter>::from_elf(
        &data,
        Config::default(),
        SyscallRegistry::default(),
    ).unwrap();

    Executable::<UserError, TestInstructionMeter>::jit_compile(&mut executable).unwrap();

    let verified_executable = VerifiedExecutable::<EmptyVerified, UserError, TestInstructionMeter>::
    from_executable(executable).unwrap();
fn main() {
    let args: Vec<String> = env::args().collect();
    let query = &args[1];
    let mut file = File::open(query).unwrap();
    let mut elf = Vec::new();
    file.read_to_end(&mut elf).unwrap();
    let data: &[u8] = &elf[..];

    let mut executable = Executable::<UserError, TestInstructionMeter>::from_elf(  
        &data,
        Config::default(),
        SyscallRegistry::default(),
    ).unwrap();

    Executable::<UserError, TestInstructionMeter>::jit_compile(&mut executable).unwrap();

    let verified_executable = VerifiedExecutable::<EmptyVerified, UserError, TestInstructionMeter>::from_executable(executable).unwrap();

    let mut vm = EbpfVm::new(&verified_executable, &mut [], vec![]).unwrap();
fn main() {
    let args: Vec<String> = env::args().collect();
    let query = &args[1];
    let mut file = File::open(query).unwrap();
    let mut elf = Vec::new();
    file.read_to_end(&mut elf).unwrap();
    let data: &[u8] = &elf[..];

    let mut executable = Executable::<UserError, TestInstructionMeter>::from_elf(
        &data,
        Config::default(),
        SyscallRegistry::default(),
    ).unwrap();

    Executable::<UserError, TestInstructionMeter>::jit_compile(&mut executable).unwrap();

    let verified_executable = VerifiedExecutable::<EmptyVerified, UserError, TestInstructionMeter>::from_executable(executable).unwrap();

    let mut vm = EbpfVm::new(&verified_executable, &mut [], vec![]).unwrap();

    vm.execute_program_jit(&mut TestInstructionMeter { remaining: 90000 });
}
Fuzzing the Solana VM

1. Simpe eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
Fuzzing the Solana VM

1. Simple eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
Fuzzing the Solana VM

1. Simpe eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
Fuzzing the Solana VM

- Simpe eBPF program
- Mutate it semi-randomly
- Execute it in the Solana VM
Fuzzing the Solana VM

1. Simpe eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM

AFL++
AFL++

- AFL++ is a fork of Google's AFL
  - Faster
  - Better instrumentation
  - Rust support!
AFL++

- Instruments the target binary
- Automatically mutates input to generate as much coverage as possible
- Comes with nice UI
- Integrated test-case minimization
fn main() {
  let args: Vec<String> = env::args().collect();
  let query = &args[1];
  let mut file = File::open(query).unwrap();
  let mut elf = Vec::new();
  file.read_to_end(&mut elf).unwrap();
  let data: &[u8] = &elf[..];

  let mut executable = Executable::<UserError, TestInstructionMeter>::from_elf(
    &data,
    Config::default(),
    SyscallRegistry::default(),
  ).unwrap();

  Executable::<UserError, TestInstructionMeter>::jit_compile(&mut executable).unwrap();

  let verifiedExecutable = VerifiedExecutable::<EmptyVerified, UserError, TestInstructionMeter>::from_executable(executable).unwrap();

  let mut vm = EbpFm::new(&verifiedExecutable, &mut [], vec![]).unwrap();

  vm.execute_program_jit(&mut TestInstructionMeter { remaining: 90000 });
}
fn main() {
    fuzz![data: &u8] { 

        let mut executable = Executable::<UserError, TestInstructionMeter>::from_elf(
            &data,
            Config::default(),
            SyscallRegistry::default(),
        ).unwrap();

        Executable::<UserError, TestInstructionMeter>::jit_compile(&mut executable).unwrap();
        let verified_executable = VerifiedExecutable::<EmptyVerifier, UserError, TestInstructionMeter>::from_executable(executable).unwrap();

        let mut vm = EbpfVm::new(&verified_executable, &mut [], vec![]).unwrap();

        vm.execute_program_jit(&mut TestInstructionMeter { remaining: 90000 });
    }
}
fn main() {
    fuzz!([data: &[u8]] | {
        panic::set_hook(Box::new(||) {
            unsafe {
                libc::_exit(0);
            }
        })
    });

    let mut executable = Executable::::<UserError, TestInstructionMeter>::from_elf(
        &data,
        Config::default(),
        SyscallRegistry::default(),
    ).unwrap();

    Executable::::<UserError, TestInstructionMeter>::jit_compile(&mut executable).unwrap();
    let verified_executable = VerifiedExecutable::<EmptyVerifier, UserError, TestInstructionMeter>::from_executable(executable).unwrap();

    let mut vm = EbpfVm::<&verified_executable, &mut [], vec![]].unwrap();

    vm.execute_program_jit(&mut TestInstructionMeter { remaining: 90000 });
}
Fuzzing the Solana VM

1. Simpe eBPF program
2. Mutate it semi-randomly
3. Execute it in the Solana VM
Fuzzing the Solana VM

Simpe eBPF program

Mutate it semi-randomly

Execute it in the Solana VM
Ready to fuzz!
root@localhost: ~

root@localhost:~/solana_fuzz


root@localhost:~

root@localhost:~/solana_fuzz# cargo afl fuzz
root@localhost:~/solana_fuzz# cargo afl fuzz -i inputs
root@localhost:~/solana_fuzz# cargo afl fuzz -i inputs -o out_demo
root@localhost:~/solana_fuzz# cargo afl fuzz -i inputs -o out_demo -- ./target/d
root@localhost:~/solana_fuzz# cargo afl fuzz -i inputs -o out_demo ./target/debug/solana_fuzz
root@localhost:~

cargo afl fuzz -i inputs -o out_demo -- ./target/debug/solana_fuzz
<table>
<thead>
<tr>
<th>process timing</th>
<th>overall results</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time: 0 days, 0 hrs, 0 min, 1 sec</td>
<td>cycles done: 0</td>
</tr>
<tr>
<td>last new find: 0 days, 0 hrs, 0 min, 0 sec</td>
<td>corpus count: 127</td>
</tr>
<tr>
<td>last saved crash: none seen yet</td>
<td>saved crashes: 0</td>
</tr>
<tr>
<td>last saved hang: none seen yet</td>
<td>saved hangs: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cycle progress</th>
<th>map coverage</th>
<th>findings in depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>now processing: 0.0 (0.0%)</td>
<td>map density: 1.48% / 8.22%</td>
<td>favored items: 4 (3.15%)</td>
</tr>
<tr>
<td>runs timed out: 0 (0.00%)</td>
<td>count coverage: 2.95 bits/tuple</td>
<td>new edges on: 84 (66.14%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>stage progress</th>
<th>item geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>now trying: havoc</td>
<td>levels: 2</td>
</tr>
<tr>
<td>stage execs: 2213/32.8k (6.75%)</td>
<td>pending: 127</td>
</tr>
<tr>
<td>total execs: 4243</td>
<td>pend fav: 4</td>
</tr>
<tr>
<td>exec speed: 3420/sec</td>
<td>own finds: 122</td>
</tr>
<tr>
<td>favored items: 4 (3.15%)</td>
<td>imported: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fuzzing strategy yields</th>
<th>stability: 77.36%</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit flips: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>byte flips: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>arithmetics: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>known ints: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>dictionary: n/a</td>
<td></td>
</tr>
<tr>
<td>havoc/splice: 0/0, 0/0</td>
<td></td>
</tr>
<tr>
<td>py/custom/rq: unused, unused, unused, unused</td>
<td></td>
</tr>
<tr>
<td>trim/eff: 0.00%/917, disabled</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Process timing</td>
<td></td>
</tr>
<tr>
<td>Run time</td>
<td>0 days, 0 hrs, 0 min, 1 sec</td>
</tr>
<tr>
<td>Last new find</td>
<td>0 days, 0 hrs, 0 min, 0 sec</td>
</tr>
<tr>
<td>Last saved crash</td>
<td>None seen yet</td>
</tr>
<tr>
<td>Last saved hang</td>
<td>None seen yet</td>
</tr>
<tr>
<td>Cycle progress</td>
<td></td>
</tr>
<tr>
<td>Now processing</td>
<td>0.0 (0.0%)</td>
</tr>
<tr>
<td>Runs timed out</td>
<td>0 (0.00%)</td>
</tr>
<tr>
<td>Stage progress</td>
<td></td>
</tr>
<tr>
<td>Now trying</td>
<td>Havoc</td>
</tr>
<tr>
<td>Stage execs</td>
<td>2213/32.8k (6.75%)</td>
</tr>
<tr>
<td>Total execs</td>
<td>4243</td>
</tr>
<tr>
<td>Exec speed</td>
<td>3420/sec</td>
</tr>
<tr>
<td>Fuzzing strategy yields</td>
<td></td>
</tr>
<tr>
<td>Bit flips</td>
<td>Disabled (default, enable with -D)</td>
</tr>
<tr>
<td>Byte flips</td>
<td>Disabled (default, enable with -D)</td>
</tr>
<tr>
<td>Arithmetics</td>
<td>Disabled (default, enable with -D)</td>
</tr>
<tr>
<td>Known ints</td>
<td>Disabled (default, enable with -D)</td>
</tr>
<tr>
<td>Dictionary</td>
<td>N/A</td>
</tr>
<tr>
<td>Havoc/splice</td>
<td>0/0, 0/0</td>
</tr>
<tr>
<td>Py/custom/rq</td>
<td>Unused, unused, unused, unused, unused</td>
</tr>
<tr>
<td>Trim/eff</td>
<td>0.00%/917, disabled</td>
</tr>
<tr>
<td>Overall results</td>
<td></td>
</tr>
<tr>
<td>Cycles done</td>
<td>0</td>
</tr>
<tr>
<td>Corpus count</td>
<td>127</td>
</tr>
<tr>
<td>Saved crashes</td>
<td>0</td>
</tr>
<tr>
<td>Saved hangs</td>
<td>0</td>
</tr>
<tr>
<td>Map coverage</td>
<td></td>
</tr>
<tr>
<td>Map density</td>
<td>1.48% / 8.22%</td>
</tr>
<tr>
<td>Count coverage</td>
<td>2.95 bits/tuple</td>
</tr>
<tr>
<td>Findings in depth</td>
<td></td>
</tr>
<tr>
<td>Favored items</td>
<td>4 (3.15%)</td>
</tr>
<tr>
<td>New edges on</td>
<td>84 (66.14%)</td>
</tr>
<tr>
<td>Total crashes</td>
<td>0 (0 saved)</td>
</tr>
<tr>
<td>Total tmouts</td>
<td>0 (0 saved)</td>
</tr>
<tr>
<td>Item geometry</td>
<td></td>
</tr>
<tr>
<td>Levels</td>
<td>2</td>
</tr>
<tr>
<td>Pending</td>
<td>127</td>
</tr>
<tr>
<td>Pend fav</td>
<td>4</td>
</tr>
<tr>
<td>Own finds</td>
<td>122</td>
</tr>
<tr>
<td>Imported</td>
<td>0</td>
</tr>
<tr>
<td>Stability</td>
<td>77.36%</td>
</tr>
<tr>
<td>process timing</td>
<td>overall results</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>run time</td>
<td>cycles done: 0</td>
</tr>
<tr>
<td>last new find</td>
<td>corpus count: 127</td>
</tr>
<tr>
<td>last saved crash</td>
<td>saved crashes: 0</td>
</tr>
<tr>
<td>last saved hang</td>
<td>saved hangs: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cycle progress</th>
<th>map coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>now processing: 0.0 (0.80%)</td>
<td>map density: 1.48% / 8.22%</td>
</tr>
<tr>
<td>runs timed out: 0 (0.00%)</td>
<td>count coverage: 2.95 bits/tuple</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>stage progress</th>
<th>findings in depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>now trying: havoc</td>
<td>favored items: 4 (3.15%)</td>
</tr>
<tr>
<td>stage execs: 2213/32.8k (6.75%)</td>
<td>new edges on: 84 (66.14%)</td>
</tr>
<tr>
<td>total execs: 4243</td>
<td>total crashes: 0 (0 saved)</td>
</tr>
<tr>
<td>exec speed: 3420/sec</td>
<td>total tmouts: 0 (0 saved)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fuzzing strategy yields</th>
<th>item geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit flips: disabled (default, enable with -D)</td>
<td>levels: 2</td>
</tr>
<tr>
<td>byte flips: disabled (default, enable with -D)</td>
<td>pending: 127</td>
</tr>
<tr>
<td>arithmetics: disabled (default, enable with -D)</td>
<td>pend fav: 4</td>
</tr>
<tr>
<td>known ints: disabled (default, enable with -D)</td>
<td>own finds: 122</td>
</tr>
<tr>
<td>dictionary: n/a</td>
<td>imported: 0</td>
</tr>
<tr>
<td>havoc/splice: 0/0, 0/0</td>
<td>stability: 77.36%</td>
</tr>
</tbody>
</table>

py/custom/rq: unused, unused, unused, unused
trim/eff: 0.00%/917, disabled
<table>
<thead>
<tr>
<th>process timing</th>
<th>overall results</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time: 0 days, 0 hrs, 1 min, 12 sec</td>
<td>cycles done: 0</td>
</tr>
<tr>
<td>last new find: 0 days, 0 hrs, 0 min, 0 sec</td>
<td>corpus count: 512</td>
</tr>
<tr>
<td>last saved crash: 0 days, 0 hrs, 0 min, 4 sec</td>
<td>saved crashes: 1</td>
</tr>
<tr>
<td>last saved hang: none seen yet</td>
<td>saved hangs: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cycle progress</th>
<th>map coverage</th>
<th>findings in depth</th>
<th>item geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>now processing: 426.0 (83.2%)</td>
<td>map density: 5.77% / 13.55%</td>
<td>favored items: 237 (46.29%)</td>
<td>levels: 5</td>
</tr>
<tr>
<td>runs timed out: 0 (0.00%)</td>
<td>count coverage: 3.71 bits/tuple</td>
<td>new edges on: 328 (64.06%)</td>
<td>pending: 477</td>
</tr>
<tr>
<td>stage progress</td>
<td>findings in depth</td>
<td>total crashes: 1 (1 saved)</td>
<td>pend fav: 208</td>
</tr>
<tr>
<td>now trying: havoc</td>
<td>favored items: 237 (46.29%)</td>
<td>total tmouts: 0 (0 saved)</td>
<td>own finds: 508</td>
</tr>
<tr>
<td>stage execs: 16.2k/32.8k (49.44%)</td>
<td>new edges on: 328 (64.06%)</td>
<td>imported: 0</td>
<td>stability: 67.93%</td>
</tr>
<tr>
<td>total execs: 262k</td>
<td>total crashes: 1 (1 saved)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exec speed: 3605/sec</td>
<td>total tmouts: 0 (0 saved)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fuzzing strategy yields</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bit flips: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>byte flips: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>arithmetics: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>known ints: disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>dictionary: n/a</td>
<td></td>
</tr>
<tr>
<td>havoc/splice: 482/191k, 9/16.7k</td>
<td></td>
</tr>
<tr>
<td>py/custom/rq: unused, unused, unused, unused, unused</td>
<td></td>
</tr>
<tr>
<td>trim/eff: 1.95%/32.1k, disabled</td>
<td></td>
</tr>
<tr>
<td>Process Timing</td>
<td>Overall Results</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Run time: 0 days, 0 hrs, 1 min, 12 sec</td>
<td>Cycles done: 0</td>
</tr>
<tr>
<td>Last new find: 0 days, 0 hrs, 0 min, 0 sec</td>
<td>Corpus count: 512</td>
</tr>
<tr>
<td>Last saved crash: 0 days, 0 hrs, 0 min, 4 sec</td>
<td><strong>Saved crashes: 1</strong></td>
</tr>
<tr>
<td>Last saved hang: none seen yet</td>
<td><strong>Saved hangs: 0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now processing: 426.0 (83.2%)</td>
<td>Map density: 5.77% / 13.55%</td>
</tr>
<tr>
<td>Runs timed out: 0 (0.00%)</td>
<td>Count coverage: 3.71 bits/tuple</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage Progress</th>
<th>Findings in Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now trying: havoc</td>
<td>Favored items: 237 (46.29%)</td>
</tr>
<tr>
<td>Stage execs: 16.2k/32.8k (49.44%)</td>
<td>New edges: 328 (64.06%)</td>
</tr>
<tr>
<td>Total execs: 262k</td>
<td>Total crashes: 1 (1 saved)</td>
</tr>
<tr>
<td>Exec speed: 3605/sec</td>
<td>Total t timeouts: 0 (0 saved)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuzzing Strategy Yields</th>
<th>Item Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit flips: disabled (default, enable with -D)</td>
<td>Levels: 5</td>
</tr>
<tr>
<td>Byte flips: disabled (default, enable with -D)</td>
<td>Pending: 477</td>
</tr>
<tr>
<td>Arithmetics: disabled (default, enable with -D)</td>
<td>Pending fav: 203</td>
</tr>
<tr>
<td>Known ints: disabled (default, enable with -D)</td>
<td>Own finds: 508</td>
</tr>
<tr>
<td>Dictionary: n/a</td>
<td>Imported: 0</td>
</tr>
<tr>
<td>Havoc/splice: 482/191k, 9/16.7k</td>
<td>Stability: 67.93%</td>
</tr>
<tr>
<td>Py/custom/rq: unused, unused, unused, unused</td>
<td><strong>Trim/eff:</strong> 1.95%/32.1k, disabled</td>
</tr>
<tr>
<td>process timing</td>
<td>overall results</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>run time</td>
<td>cycles done: 0</td>
</tr>
<tr>
<td>last new find</td>
<td>corpus count: 512</td>
</tr>
<tr>
<td>last saved crash</td>
<td>saved crashes: 54</td>
</tr>
<tr>
<td>last saved hang</td>
<td>saved hangs: 0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>cycle progress</td>
<td></td>
</tr>
<tr>
<td>now processing</td>
<td>map coverage</td>
</tr>
<tr>
<td>426.0 (83.2%)</td>
<td>map density: 5.77% / 13.55%</td>
</tr>
<tr>
<td>runs timed out</td>
<td>count coverage: 3.71 bits/tuple</td>
</tr>
<tr>
<td>0 (0.00%)</td>
<td></td>
</tr>
<tr>
<td>stage progress</td>
<td>findings in depth</td>
</tr>
<tr>
<td>now trying</td>
<td>favored items: 237 (46.29%)</td>
</tr>
<tr>
<td>havoc</td>
<td>new edges on: 328 (64.06%)</td>
</tr>
<tr>
<td>stage execs</td>
<td>total crashes: 1 (1 saved)</td>
</tr>
<tr>
<td>16.2k/32.8k (49.44%)</td>
<td>total tmouts: 0 (0 saved)</td>
</tr>
<tr>
<td>total execs</td>
<td></td>
</tr>
<tr>
<td>202k</td>
<td></td>
</tr>
<tr>
<td>exec speed</td>
<td></td>
</tr>
<tr>
<td>3605/sec</td>
<td></td>
</tr>
<tr>
<td>fuzzing strategy yields</td>
<td></td>
</tr>
<tr>
<td>bit flips</td>
<td>item geometry</td>
</tr>
<tr>
<td>disabled (default, enable with -D)</td>
<td>levels: 5</td>
</tr>
<tr>
<td>byte flips</td>
<td>pending: 477</td>
</tr>
<tr>
<td>disabled (default, enable with -D)</td>
<td>pend fav: 208</td>
</tr>
<tr>
<td>arithmetics</td>
<td>own finds: 508</td>
</tr>
<tr>
<td>disabled (default, enable with -D)</td>
<td>imported: 0</td>
</tr>
<tr>
<td>known ints</td>
<td></td>
</tr>
<tr>
<td>disabled (default, enable with -D)</td>
<td>stability: 67.93%</td>
</tr>
<tr>
<td>dictionary</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>havoc/splice</td>
<td>482/191k, 9/16.7k</td>
</tr>
<tr>
<td>py/custom/rq</td>
<td>unused, unused, unused, unused, unused</td>
</tr>
<tr>
<td>trim/eff</td>
<td>1.95%/32.1k, disabled</td>
</tr>
</tbody>
</table>

[cpu000: 50%]
Mostly floating-point exceptions :(

```plaintext
id:00000    sig:08  src:000001, time:159670, op:flip1, pos:248
did:00000    sig:08  src:000001, time:159671, op:flip1, pos:248
did:00000    sig:08  src:000001, time:163277, op:flip2, pos:248
did:00000    sig:08  src:000001, time:281256, op:arith6, pos:247, val:-1
did:00000    sig:08  src:000001, time:281256, op:arith6, pos:247, val:-2
did:00000    sig:08  src:000001, time:281260, op:arith6, pos:247, val:-9
did:00000    sig:08  src:000001, time:281266, op:arith6, pos:247, val:-18
did:00000    sig:08  src:000001, time:226148, op:arith32, pos:245, val:-1
did:00000    sig:08  src:000001, time:226171, op:arith32, pos:246, val:-4
did:00000    sig:08  src:000056, time:1905518, op:flip2, pos:608
did:00001    sig:08  src:000056, time:1923383, op:arith8, pos:608, val:+5
did:00001    sig:08  src:000056, time:2016965, op:ext_A0, pos:252
did:00001    sig:08  src:000058, time:2041379, op:flip2, pos:608
did:00001    sig:08  src:000065, time:2581381, op:flip1, pos:264
did:00001    sig:08  src:000066, time:2988743, op:flip1, pos:620
did:00001    sig:08  src:000066, time:2991948, op:flip2, pos:256
did:00001    sig:08  src:000068, time:2992215, op:flip2, pos:376
did:00001    sig:08  src:000068, time:2996028, op:flip4, pos:256
did:00001    sig:08  src:000068, time:2996029, op:flip4, pos:256
did:00001    sig:08  src:000068, time:2996030, op:flip4, pos:256
did:00001    sig:08  src:000068, time:2996239, op:flip4, pos:352
did:00001    sig:08  src:000068, time:2999561, op:flip8, pos:256
did:00001    sig:08  src:000068, time:3004869, op:arith8, pos:256, val:-6
```

Segmentation faults 😞
<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time: 0 days, 0 hrs, 1 min, 12 sec</td>
<td>cycles done: 0</td>
</tr>
<tr>
<td>last new find: 0 days, 0 hrs, 0 min, 0 sec</td>
<td>corpus count: 512</td>
</tr>
<tr>
<td>last saved crash: 0 days, 0 hrs, 0 min, 4 sec</td>
<td>saved crashes: 1</td>
</tr>
<tr>
<td>last saved hang: none seen yet</td>
<td>saved hangs: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>now processing: 426.0 (83.2%)</td>
<td>map density: 5.77% / 13.55%</td>
</tr>
<tr>
<td>runs timed out: 0 (0.00%)</td>
<td>count coverage: 3.71 bits/tuple</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage Progress</th>
<th>Findings in Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>now trying: havoc</td>
<td>favored items: 237 (46.29%)</td>
</tr>
<tr>
<td>stage execs: 16.2k/32.8k (49.44%)</td>
<td>new edges on: 328 (64.06%)</td>
</tr>
<tr>
<td>total execs: 262k</td>
<td>total crashes: 1 (1 saved)</td>
</tr>
<tr>
<td>exec speed: 3605/sec</td>
<td>total ttmouts: 0 (0 saved)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuzzing Strategy Yields</th>
<th>Item Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit flips: disabled (default, enable with -D)</td>
<td>levels: 5</td>
</tr>
<tr>
<td>byte flips: disabled (default, enable with -D)</td>
<td>pending: 477</td>
</tr>
<tr>
<td>arithmetics: disabled (default, enable with -D)</td>
<td>pend fav: 208</td>
</tr>
<tr>
<td>known ints: disabled (default, enable with -D)</td>
<td>own finds: 508</td>
</tr>
<tr>
<td>dictionary: n/a</td>
<td>imported: 0</td>
</tr>
<tr>
<td>havoc/splice: 482/191k, 9/16.7k</td>
<td>stability: 67.93%</td>
</tr>
<tr>
<td>py/custom/rq: unused, unused, unused, unused</td>
<td>trim/eff: 1.95%/32.1k, disabled</td>
</tr>
<tr>
<td>process timing</td>
<td>overall results</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>run time : 0 days, 0 hrs, 1 min, 12 sec</td>
<td>cycles done : 0</td>
</tr>
<tr>
<td>last new find : 0 days, 0 hrs, 0 min, 0 sec</td>
<td>corpus count : 512</td>
</tr>
<tr>
<td>last saved crash : 0 days, 0 hrs, 0 min, 4 sec</td>
<td>saved crashes : 1</td>
</tr>
<tr>
<td>last saved hang : none seen yet</td>
<td>saved hangs : 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cycle progress</th>
<th>map coverage</th>
<th>findings in depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>now processing : 426.0 (83.2%)</td>
<td>map density : 5.77% / 13.55%</td>
<td>favored items : 237 (46.29%)</td>
</tr>
<tr>
<td>runs timed out : 0 (0.00%)</td>
<td>count coverage : 3.71 bits/tuple</td>
<td>new edges on : 328 (64.06%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>stage progress</th>
<th>total crashes : 1 (1 saved)</th>
<th>item geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>now trying : havoc</td>
<td>total timeouts : 0 (0 saved)</td>
<td>levels : 5</td>
</tr>
<tr>
<td>stage execs : 16.2k/32.8k (49.44%)</td>
<td></td>
<td>pending : 477</td>
</tr>
<tr>
<td>total execs : 262k</td>
<td></td>
<td>pend fav : 208</td>
</tr>
<tr>
<td>exec speed : 3605/sec</td>
<td></td>
<td>own finds : 508</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fuzzing strategy yields</th>
<th>imported : 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit flips : disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>byte flips : disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>arithmetics : disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>known ints : disabled (default, enable with -D)</td>
<td></td>
</tr>
<tr>
<td>dictionary : n/a</td>
<td></td>
</tr>
<tr>
<td>havoc/splice : 482/191k, 9/16.7k</td>
<td></td>
</tr>
<tr>
<td>py/custom/rq : unused, unused, unused, unused</td>
<td></td>
</tr>
<tr>
<td>trim/eff : 1.95%/32.1k, disabled</td>
<td>stability : 67.93%</td>
</tr>
</tbody>
</table>
It's not deterministic?!
/// Ratio of native host instructions per random no-op in JIT (0 = OFF)
pub noop_instruction_rate: u32,
<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time: 0 days, 0 hrs, 0 min, 3 sec</td>
<td>Cycles done: 0</td>
</tr>
<tr>
<td>Last new find: 0 days, 0 hrs, 0 min, 0 sec</td>
<td>Corpus count: 201</td>
</tr>
<tr>
<td>Last saved crash: none seen yet</td>
<td>Saved crashes: 0</td>
</tr>
<tr>
<td>Last saved hang: none seen yet</td>
<td>Saved hangs: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now processing: 2.0 (1.0%)</td>
<td>Map Density: 5.60% / 7.98%</td>
</tr>
<tr>
<td>Runs timed out: 0 (0.00%)</td>
<td>Count Coverage: 1.67 bits/tuple</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage Progress</th>
<th>Findings in Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now trying: havoc</td>
<td>Favored items: 4 (1.99%)</td>
</tr>
<tr>
<td>Stage execs: 9180/24.6k (37.35%)</td>
<td>New edges on: 120 (59.70%)</td>
</tr>
<tr>
<td>Total execs: 12.4k</td>
<td>Total crashes: 0 (0 saved)</td>
</tr>
<tr>
<td>Exec speed: 3422/sec</td>
<td>Total tmouts: 0 (0 saved)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuzzing Strategy Yields</th>
<th>Item Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit flips: disabled (default, enable with -D)</td>
<td>Levels: 2</td>
</tr>
<tr>
<td>Byte flips: disabled (default, enable with -D)</td>
<td>Pending: 201</td>
</tr>
<tr>
<td>Arithmetics: disabled (default, enable with -D)</td>
<td>Pend fav: 4</td>
</tr>
<tr>
<td>Known ints: disabled (default, enable with -D)</td>
<td>Own finds: 197</td>
</tr>
<tr>
<td>Dictionary: n/a</td>
<td>Imported: 0</td>
</tr>
<tr>
<td>Havoc/Splice: 0/0, 0/0</td>
<td>Stability: 98.09%</td>
</tr>
<tr>
<td>Py/Custom/Rq: unused, unused, unused, unused</td>
<td></td>
</tr>
<tr>
<td>Trim/Eff: 0.00%/908, disabled</td>
<td></td>
</tr>
</tbody>
</table>

[cpu000: 75%]
51 segmentation faults!
Triage
Triage
So many indirections...

eBPF
Bytecode
Triage
So many indirections...

- eBPF Bytecode
- rBPF JIT
Triage
So many indirections...

- eBPF Bytecode
- rBPF JIT
- Native x64 instructions
Triage
So many indirections...

- eBPF Bytecode
- rBPF JIT
- Native x64 instructions
- Actual process memory

kraken
There is a running process, kill it and restart?: [Y/n] y
Process 124037 exited with status = 9 (0x00000009)
Process 172729 launched: '/root/solana_triage/target/debug/solana_run_ebpf' (x86_64)
Process 172729 stopped
→ thread #1, name = 'solana_run_ebpf', stop reason = signal SIGSEGV: address access protected (fault address: 0x7f4c0320ff5)
   frame #0: 0x00007f43c0320f8b
   -> 0x7f43c0320f8b: movq $0x1, (%r10)
   0x7f43c0320f8b: addq $0x1c, %r11
   0x7f43c0320f8c: movq %r11, %x010(%r10)
   0x7f43c0320f90: jmp 0x7f43c032bb1b
There is a running process, kill it and restart?: [Y/n] y
Process 172729 exited with status = 9 (0x00000009)
Process 179500 launched: '/root/solana_triage/target/debug/solana_run_ebpf' (x86_64)
Process 179500 stopped
→ thread #1, name = 'solana_run_ebpf', stop reason = signal SIGSEGV: address access protected (fault address: 0x7f4a6678b8ff5)
   frame #0: 0x00007f4a6678b89b
   -> 0x7f4a6678b89b: movq $0x1, (%r10)
   0x7f4a6678b89b: addq $0x1c, %r11
   0x7f4a6678b89c: movq %r11, %x010(%r10)
   0x7f4a6678b8a0: jmp 0x7f4a6678bb1b
There is a running process, kill it and restart?: [Y/n] y
Process 179500 exited with status = 9 (0x00000009)
Process 187880 launched: '/root/solana_triage/target/debug/solana_run_ebpf' (x86_64)
Process 187880 stopped
→ thread #1, name = 'solana_run_ebpf', stop reason = signal SIGSEGV: address access protected (fault address: 0x7ff5de1dd8ff5)
   frame #0: 0x00007fff5de1dd9b
   -> 0x7fff5de1dd9b: movq $0x1, (%r10)
   0x7fff5de1dd9b: addq $0x1c, %r11
   0x7fff5de1d9c: movq %r11, %x010(%r10)
   0x7fff5de1dd0: jmp 0x7fff5de1ddbb1b
There is a running process, kill it and restart?: [Y/n] y
Process 126037 exited with status = 9 (0x00000009)
Process 172729 launched: '/root/solana_triage/target/debug/solana_run_epbf' (x86_64)
Process 172729 stopped
* thread #1, name = 'solana_run_epbf', stop reason = signal SIGSEGV: address access protected (fault address: 0x7f43c0328f5)
  frame #0: 0x00007f43c03209bd
  -> 0x7f43c03289bd: movq $0x1, (%r10)
             0x7f43c03209c5: addq $0x1c, %r11
             0x7f43c03209cc: movq %r11, 0x10(%r10)
             0x7f43c0320980: jmp 0x7f43c032bb1b

There is a running process, kill it and restart?: [Y/n] y
Process 172729 exited with status = 9 (0x00000009)
Process 179500 launched: '/root/solana_triage/target/debug/solana_run_epbf' (x86_64)
Process 179500 stopped
* thread #1, name = 'solana_run_epbf', stop reason = signal SIGSEGV: address access protected (fault address: 0x7fa6678b88f5)
  frame #0: 0x00007fa6678b89bd
  -> 0x7fa6678b89bd: movq $0x1, (%r10)
             0x7fa6678b89c5: addq $0x1c, %r11
             0x7fa6678b89cc: movq %r11, 0x10(%r10)
             0x7fa6678b890: jmp 0x7fa6678bb1b

There is a running process, kill it and restart?: [Y/n] y
Process 179500 exited with status = 9 (0x00000009)
Process 187809 launched: '/root/solana_triage/target/debug/solana_run_epbf' (x86_64)
Process 187809 stopped
* thread #1, name = 'solana_run_epbf', stop reason = signal SIGSEGV: address access protected (fault address: 0x7ff5de1dd8f5)
  frame #0: 0x00007ff5de1dd9bd
  -> 0x7ff5de1dd9bd: movq $0x1, (%r10)
             0x7ff5de1dd9c5: addq $0x1c, %r11
             0x7ff5de1dd9cc: movq %r11, 0x10(%r10)
             0x7ff5de1dd90: jmp 0x7ff5de1ddbb1b
Triage

- Customized JIT to emit raw binary + map of eBPF instructions
- Spent hours analyzing

Findings:
- Multiple out-of-bound reads
- Multiple out-of-bound writes
Affected in the wild?
The program verifier...

// called with EbpFVmFixedMbuf:: prefix
pub fn new!(prog: &'a [u8],
    data_offset: usize,
    data_end_offset: usize) -> Result<EbpFVmFixedMbuf<&'a>, Error>

// called with EbpFVmRaw:: prefix
pub fn new!(prog: &'a [u8]) -> Result<EbpFVmRaw<&'a>, Error>

// called with EbpFVmNoData:: prefix
pub fn new!(prog: &'a [u8]) -> Result<EbpFVmNoData<&'a>, Error>

This is used to create a new instance of a VM. The return type is dependent of the struct from which the function is called. For instance, \texttt{EbpFVmRaw::new(key_program)} would return an instance of `struct rbpfi::EbpFVmRaw` (wrapped in a `Result`). When a program is loaded, it is checked with a very simple verifier (nothing close to the one for Linux kernel). Users are also able to replace it with a custom verifier.

For \texttt{EbpFVmFixedMbuf}, two additional arguments must be passed to the constructor: \texttt{data_offset} and \texttt{data_end_offset}. They are the offset (byte number) at which the pointers to the beginning and to the end, respectively, of the memory area of packet data are to be stored in the internal metadata buffer each time the program is executed. Other structs do not use this mechanism and do not need those offsets.

// for struct EbpFVmMbuf, struct EbpFVmRaw and struct EbpFVmRawData
pub fn set_program(&mut self, prog: &'a [u8]) -> Result<(), Error>

// for struct EbpFVmFixedMbuf
pub fn set_program(&mut self, prog: &'a [u8],
    data_offset: usize,
    data_end_offset: usize) -> Result<(), Error>
The program verifier...

This is used to create a new instance of a VM. The return type is dependent of the struct from which the function is called. For instance, `ebpf::EbpVnFixed::new(key_program)` would return an instance of `struct ebpf::EbpVnFixed` (wrapped in a `Result`). When a program is loaded, it is checked with a very simple verifier (nothing close to the one for Linux kernel). Users are also able to replace it with a custom verifier.

For `struct EbpfVnFixed`, two additional arguments must be passed to the constructor: `data_offset` and `data_end_offset`. They are the offset (byte number) at which the pointers to the beginning and to the end, respectively, of the memory area of packet data are to be stored in the internal metadata buffer each time the program is executed. Other structs do not use this mechanism and do not need those offsets.

// for struct EbpfVnFixed, struct EbpfVnRaw and struct EbpfVnData
pub fn set_program(&mut self, prog: &a [u8]) -> Result{}, Error

// for struct EbpfVnFixed
pub fn set_program(&mut self, prog: &a [u8],
data_offset: usize,
data_end_offset: usize) -> Result{}, Error
Disclosure...
Timeline

- 04. Of April: Submitted this talk to $conference
- 01. Of May: Submitted this talk to DEF CON
- 11. Of May: Secret.Club's Addison publishes a blogpost...
Timeline

- 04. Of April: Submitted this talk to $conference
- 01. Of May: Submitted this talk to DEF CON
- 11. Of May: Secret Club's Addison publishes a blogpost...
Bug collision

- Full bug collision with secret.club
- Patches completely patch any segmentation-fault we had
- Fuzzer only finds floating-point exceptions now
Takeaways

• JITs are fun and often pretty easy to fuzz
• Triaging JITs however is pretty difficult
• Performance sometimes impacts security
• Verify the full chain before getting to excited about a vulnerability
• Bug collisions happen
Thank you!