Fuzzing Simplicity: A Story

Andrew Poelstra

Blockstream Engineering Call
June 25, 2021

Preamble

Elements (from Bitcoin) has fuzzing infrastructure.

- Run without corpus in 2020 or so. Not since(?)
- No pre-built fuzzer input corpus. Didn't know how to build it. (doc/fuzzer.md)
- Elements needs old boost, etc., which means nix.
- But lcov has weird expectations about its environment.

Preamble

Byron threw together a fuzz target (#1332 on ElementsProject/elements)

- `PrecomputedTransactionData` requires calling an `Init` method or it doesn't work.
- And actually it still won't work; needs a complete list of input data or it will silently not initialize the data.
- Also we have a `tapEnv` object that needs a valid CMR (can't be obtained by the fuzzer).

A First Start

Ok, so fix these things:

- Decode some input scriptpubkeys etc from the fuzzer.
- For the input under test, decode a Simplicity program, tweak a key, make a scriptpubkey, etc.
- Just disable the CMR check.

After six days, achieve 36% coverage of the `simplicity/` directory.

Idea: Generate, Don't Decode

Coverage shows that we are only hitting a couple combinators and struggling to get nontrivial things that typecheck.

- It's hard for the fuzzer to generate valid diverse transactions from bytes.
- And even harder to generate Simplicity which is bit-aligned, and length prefixed with bit-aligned prefix-free encoding.
- Also we don't need to do taptweaks so we should not do taptweaks.

Idea: Generate, Don't Decode

So instead take each byte from the fuzzer and switch on it, generating all valid possibilities. Easy to do in Rust.

- Do this for every Elements transaction data structure including rangeproofs and surjectionproofs (should upstream this..).
- Learn weird cool stuff about Elements consensus logic (did you know that coinbase inputs cannot carry asset issuances?)
- File a bug against rust-elements about parsing pegin witnesses.

Idea: Generate, Don't Decode

Then just link the Rust code to the C++ fuzzing harness. Easy.

- Both Rust and C++ can handle byteslices. C cannot. You have to go through C.
- Both Rust and C++ have RAII. C does not. You have to manually get your destructors right.

Idea: Generate, Don't Decode

Then just link the Rust code to the C++ fuzzing harness. Easy.

- rust-simplicity and Elements both have copies of libsimplicity. They are not the same. And even if they were, the linker will still barf on them because C has no namespaces and C has poisoned everything.

- So use a shared library for the Rust code. Good luck getting the `LD_LIBRARY_PATH` var in the fuzzer Python script correct inside a Nix environment.

How to Generate Simplicity Programs

Fuzzer-guided recursive type generation is tricky.

- You need to cap your sizes somehow. If it's possible to make arbitrarily-large things the fuzzer will figure it out.
- Simplicity has exponentially-sized types and exponentially-sized values.
- No problem, we have static analysis. Except rust-simplicity actually evaluates entire types before the static bounds are applied (rust-simplicity #221 and #222)

How to Generate Simplicity Programs

Simplicity types are weird.

- During type inference, types are not fully specified and do not have exact sizes.
- They can also be infinitely sized, and checking for this is expensive so we defer it.
- But finalization can't happen until your whole program is built.

How to Generate Simplicity Programs

Simplicity types are weird.

- Programs must be 1-¿1 (take no input, take no output).
- "Inputs" are witnesses, "outputs" are aborts.
- To glue two programs together, easiest is to pair them... but this will blow up your type sizes if you are not careful.

Misc

- Anyway I got up to 91.5% once but then stalled out, still not hitting all jets, and with some seed inputs *very* slow.
- At one point honggfuzz started crashing because of something to do with memcmp and I had to update it.
- Also my keyboard broke.

Future Work

- More covenant Script fragments (scheduled payouts, dividends, bonds)
- Reissuance covenants
- Improving Script for efficiency/expressivity
- Porting this all to Simplicity, zero-knowledge, crossing chains, $\cdots$