What blockchain got right

No, really

Trail of Bits

Cyber security research company - High-end security research with a real-world attacker mentality to reduce risk and fortify code.

Security Research

- As a leading cybersecurity research provider to DARPA, the Army and the Navy – we create and release open source research tools

Security Engineering

- We offer custom engineering for every stage of software creation, from initial planning to enhancing the security of completed works

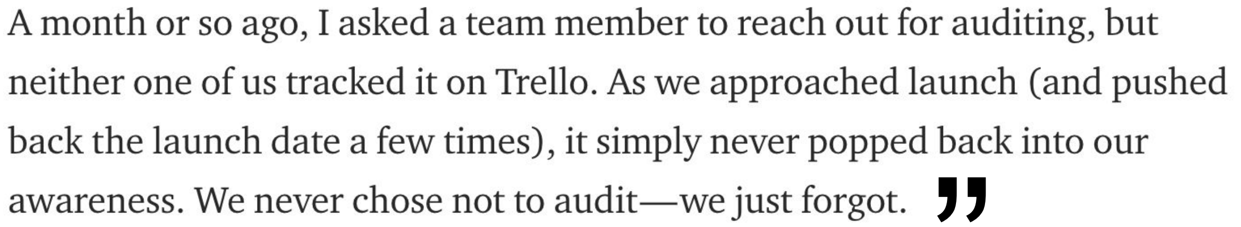
Security Assessments

- We offer security auditing for code and systems requiring extreme robustness and niche system expertise

Case study: Ethereum

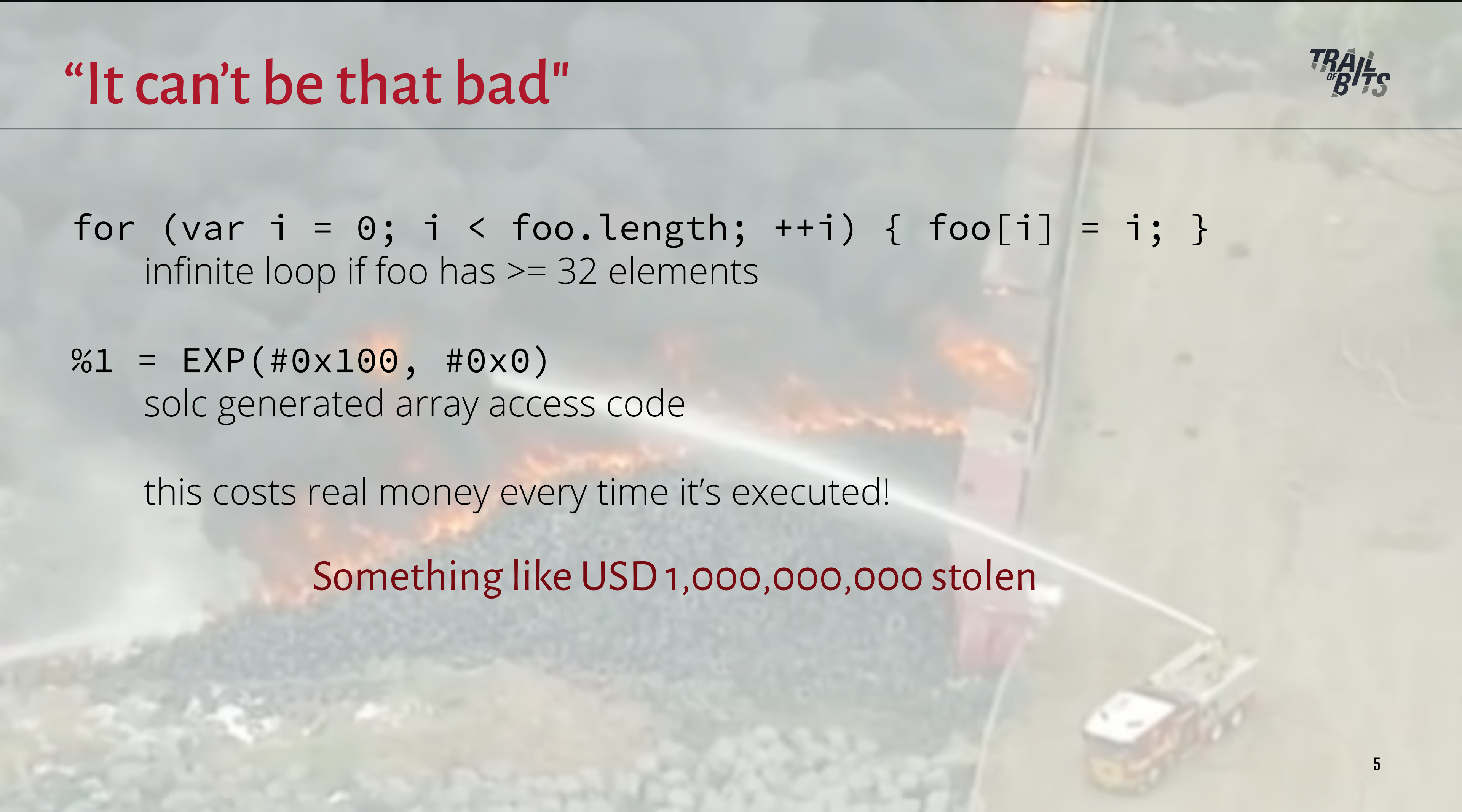
Let’s talk about blockchain

- **Ethereum smart contracts**
  - Tiny programs, run in consensus, keep getting hacked
  - Lingua franca, Solidity, is “JavaScript but worse”
    - Compiler frequently introduces serious correctness bugs
    - *Anatomy of an Unsafe Programming Language, Evan Sultanik*
  - Community:



> A month or so ago, I asked a team member to reach out for auditing, but neither one of us tracked it on Trello. As we approached launch (and pushed back the launch date a few times), it simply never popped back into our awareness. We never chose not to audit—we just forgot.

“It can’t be that bad"



```
for (var i = 0; i < foo.length; ++i) { foo[i] = i; }
```

infinite loop if foo has >= 32 elements

```
%1 = EXP(#0x100, #0x0)
```

solc generated array access code

this costs real money every time it’s executed!

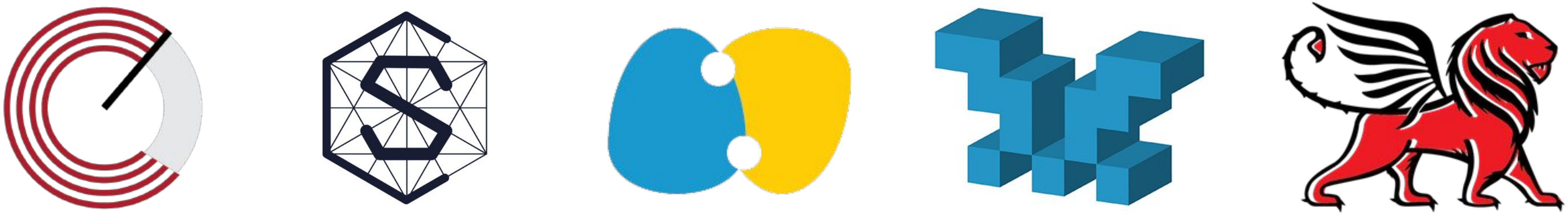
Something like USD 1,000,000,000 stolen

Solidity correctness: Expectations

- **Easy bugs**
  - OK, this is mostly true
- **Analysis is tricky**
  - How do you deal with that weird stack machine?
  - The language doesn’t make any sense
- **Confidence in these systems is near-impossible**
  - Regular software is bad enough

Solidity correctness: Reality

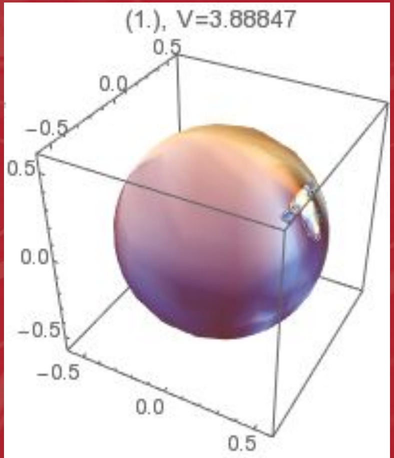
- **In many ways, leading the industry**
- **Clients *come in the door* with:**
  - Property-based tests
  - Symbolic execution results
- **SEaaS (Symbolic Execution aaS) is a competitive space**
  - Trail of Bits has one, crytic.io, and many others exist:



Why? Incentives?

- "code is law"
    - You need to be right the first time
    - No recourse if you're hacked*
- **Regular software needs to be correct too!**
    - Certainly self-driving cars aren't this correct
    - Nor are iPhones

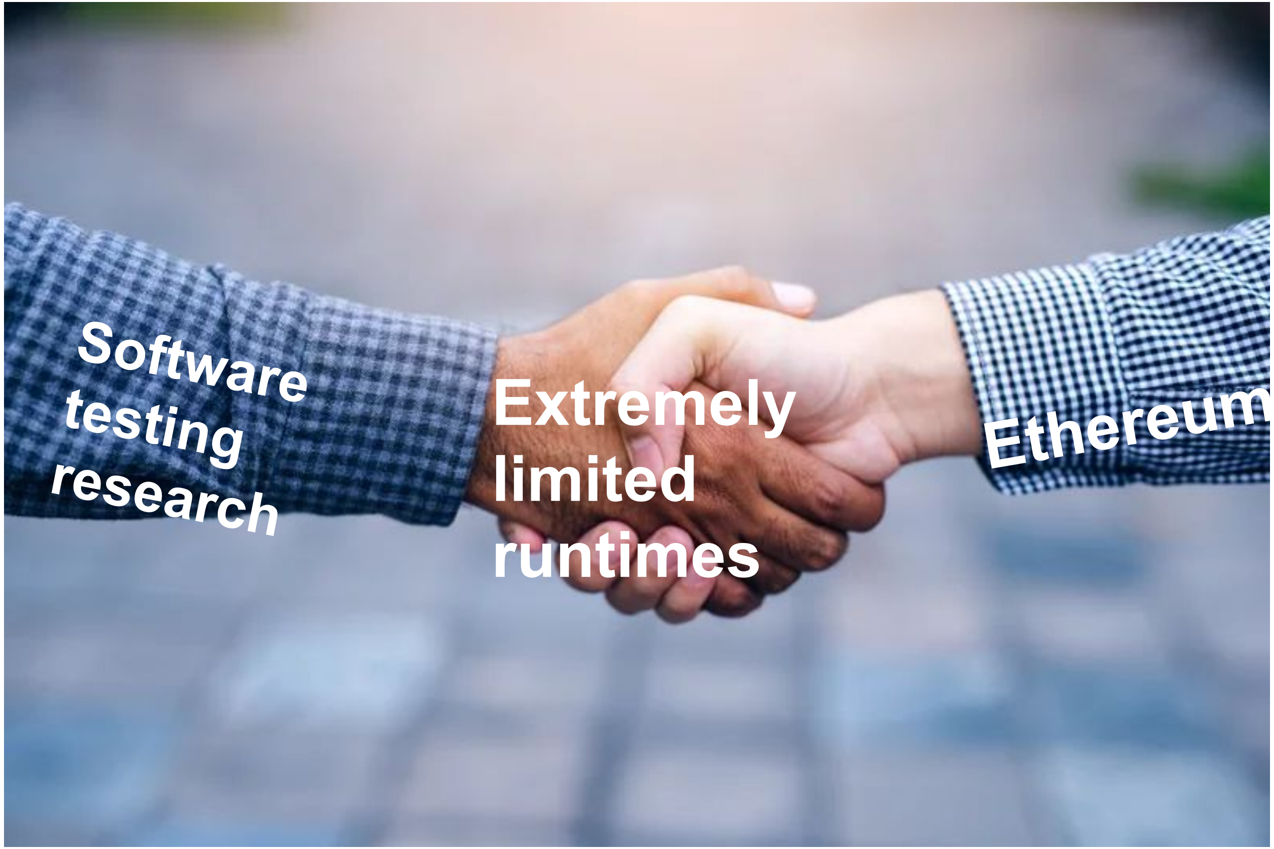
***the DAO is an exception**



"I have the solution, but it works only in the case of spherical cows in a vacuum" — anon. physicist

Blockchain only supports spherical cows!

The EVM as a testing research environment



In this presentation

What lets us test smart contracts so well?

How do does this work in practice for smart contract devs?

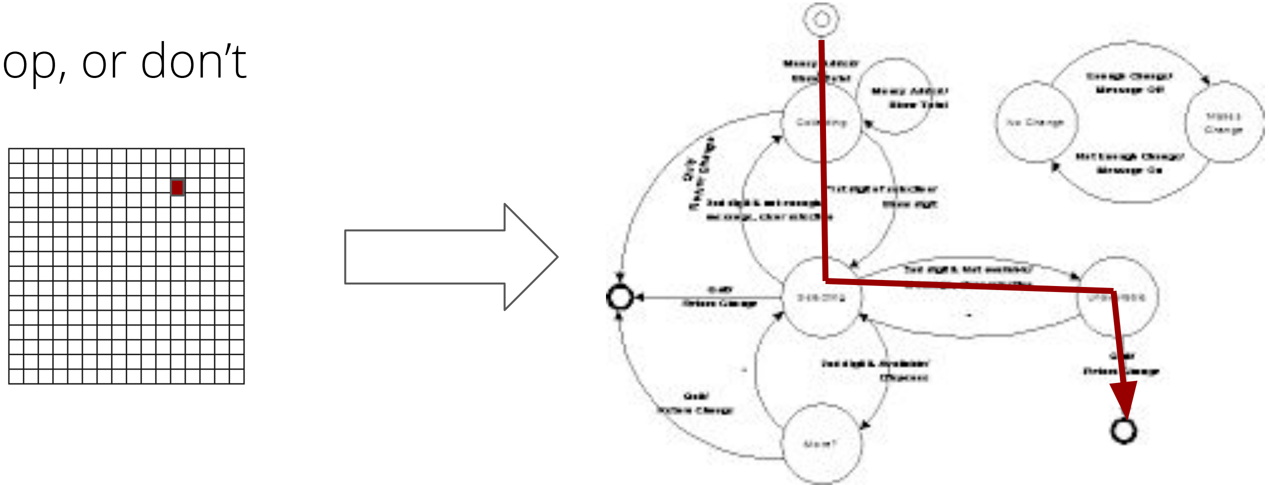
Can we replicate these results elsewhere?

The evolution of testing

TRAIL OF BITS

What is a “program”

- **Highly nuanced, but this is a keynote**
- **Let’s pretend a program is like this:**
  - Take points in some “input” space (stdin, network state, whatever)
  - Move around some state machine
  - Maybe stop, or don’t



What do we research?

Two important questions:

1. When is a state “bad”?
   - Assert, ASAN, etc.
2. What inputs cause “bad” behavior?
   - Fuzzers, symbolic executors, etc.

To illustrate, let’s dive into the industry’s solutions to (2)

Phase 0: Try really hard

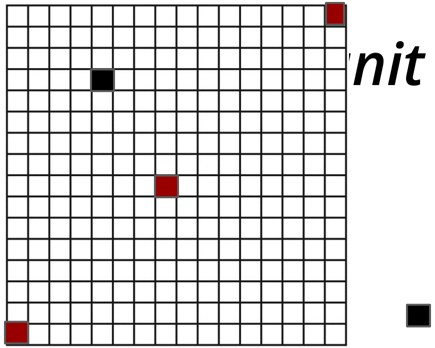
“I would simply think really hard and not introduce memory corruption bugs into my C codebase”

- This absolutely does not work
- How are you managing your team
- Honestly wtf

Phase 1: Try a few inputs

“I would simply list all the things I forgot, then make unit tests”

“I would simply list all the things I forgot, ***nit tests”***

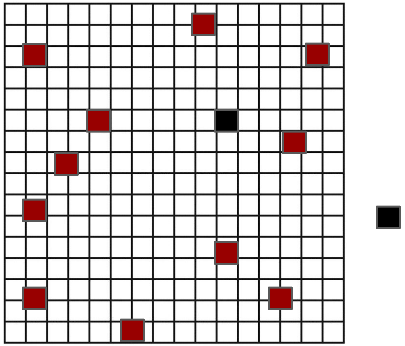


- ***Considerably*** **better than phase 0**
- **Still doesn’t really work**
- **Most things aren’t unit tested**
- **Programmers won’t know all their unknowns**

Currently, approximately industry state of the art

Phase 2: Try lots of random inputs

- **Fuzzers, property-based testing**
- **Hot new research area!**
  - Tons of fuzzer papers
  - Tons of property-based testing talks/libraries



- **Unreasonably effective**
  - afl is the world's #1 bugfinding tool
  - How do you know if someone uses property tests? They'll tell you
- **Fuzzing is starting to gain industry acceptance**

Phase 3: Test all the inputs

- This is the endgame
  - not a cure-all though (quis certificat...)
- Verification, symbolic execution
- Mostly rejected as impractical
- When it works, *incredible*

Where we are today

Industry today

- **Almost everyone is phase 0**
- **Phase 1 is worth bragging about**
- **Phase 2 is next-level, cutting edge**
    - Proof: Fuzzing like it’s 1989, Artem Dinaburg
- **Phase 3 is “wildly impractical”**

Why are we stuck?

Our developer tools don’t create testable programs, yet our testing tools require them

- Great research gets stuck in the academy
- Tools work on small code/data, nothing in prod is small

How do we move past this?

(1) Problem: Bad abstraction boundaries

- **Breaking programs into smaller programs gets us**
  - integration tests → unit tests
  - fuzzing → property-based testing
  - impossible verification problems → slightly less impossible
- **Programmers don't want to write code like that**
  - Unix philosophy lost, Linux philosophy won
  - Functional lost, OO won
- **Global mutable state is the root of all evil**

(2) Problem: Reproducibility? What's that

- **We assume wherever code runs works like our laptops**
- **Programmers only sometimes even use docker**
- **Builds are a mess**
- **We outsource logic to massively unpredictable environments**
    - Deploy to different OS's
    - No dependency versioning
    - Network calls to different places

(3) Problem: Input space is huge

- **Programmers don’t know what affects their logic**
- **People expect everything to be referentially transparent**
    - Nothing is. See, *How to write a rootkit without really trying aka krf*
- **Programmers don’t restrict their inputs well (types)**
    - “stringly-typed” code gets whitespace bugs everywhere
    - null
    - “who needs fancy types, we have lists and tuples”

Trail of Bits tries to use research

- **We read papers, review carefully, check out code**
  - They aren't designed for real software
  - Reproduction is near-impossible
  - At best, code is optimized for coreutils
- **Fuzzers are most realistic, but have awful methodology**
  - *How to spot good fuzzing research, Trent Brunson*
  - Systematic review of 32 fuzzing papers by Andrew Ruef @ UMD

The crux: we don’t know when code is safe

- **How can you convince someone code is good?**
  - Unit tests are just more code
  - Security reviews mean many different things, have bad signal/noise
  - Bounty size? Bug tracker? Community sentiment?
  - Machine learning?
- **We’ve accepted that correctness is for academics**

“Unhackable” is a punchline if it ever comes up
Default assumption: code probably has bad bugs, we don’t know where

Blockchain as… the future?

🤢

What does winning look like?

When we apply research, what happens?

- How do security workflows look?
- What’s different for devs?

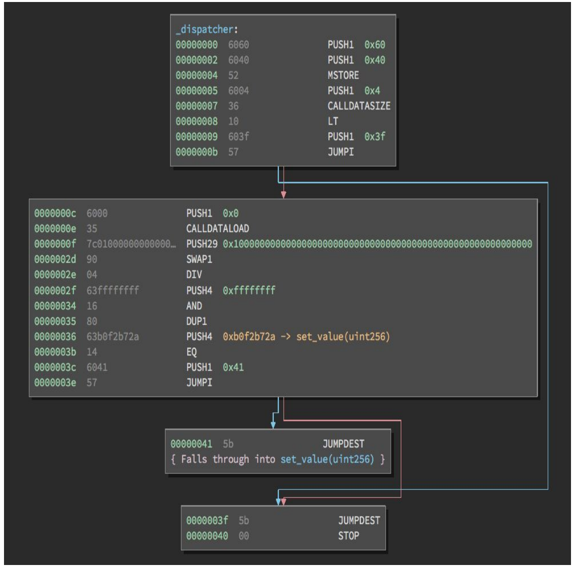
Blockchain as a test case for good testing technology

- If brown cows make chocolate milk, do spherical cows make dippin’ dots?

How do smart contracts work?

- **Everything that changes state is a transaction**
  - Human to contract and contract to contract calls are the same
  - Transactions are atomic
- **Code is tiny, has almost no control flow**
  - Code size is expensive
  - Input size is really expensive
  - Pay per instruction executed
  - Termination is thus necessary

…maybe incentives do matter!



```
_dispatcher:
00000000  6060                PUSH1  0x60
00000002  6040                PUSH1  0x40
00000004  52                  MSTORE
00000005  6004                PUSH1  0x4
00000007  36                  CALLDATASIZE
00000008  10                  LT
00000009  603f                PUSH1  0x3f
0000000b  57                  JUMPI
```

```
0000000c  6000                PUSH1  0x0
0000000e  35                  CALLDATALOAD
0000000f  7c01000000000000…   PUSH29 0x100000000000000000000000000000000000000000000000000000000
0000002d  90                  SWAP1
0000002e  04                  DIV
0000002f  63ffffffff          PUSH4  0xffffffff
00000034  16                  AND
00000035  80                  DUP1
00000036  63b0f2b72a          PUSH4  0xb0f2b72a -> set_value(uint256)
0000003b  14                  EQ
0000003c  6041                PUSH1  0x41
0000003e  57                  JUMPI
```

```
00000041  5b                  JUMPDEST
{ Falls through into set_value(uint256) }
```

```
0000003f  5b                  JUMPDEST
00000040  00                  STOP
```

https://github.com/crytic/ethersplay

“What can happen with this contract?”

- **Any time state updates, totally determined by:**
  - Transaction data (small)
  - Existing state variables (also small)
- **Approximate numbers to demonstrate scale:**
  - Input size typically order of 100s of bytes per tx
  - 100s of bytes of state variables
  - Instruction traces order of 1000s of instructions at most
  - State space is manageable!

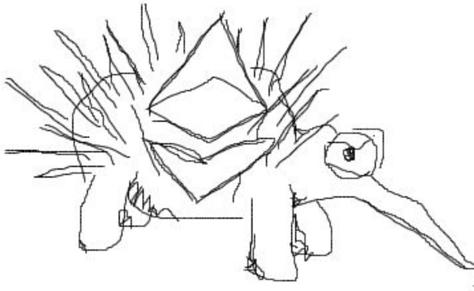
Symbolic Execution

- **The most common technique in the space (?!)**
  - About a half dozen symbolic executors exist
- **Writing a mostly sound symbolic EVM isn't that hard**
  - Writing symbolic brainfuck: 1/10 hard
  - Writing symbolic EVM: 3/10 hard
  - Writing symbolic x86: 30/10 hard
- **Even amateur developers can use it effectively**
  - The hardest part: what constitutes a bug?
  - Of course, experts can do more

https://github.com/trailofbits/manticore

Fuzzing

- **Remarkably, less common than symbex**
  - In the right environment, the correct option can be easier!
  - One public one exists, a few other private ones may
- **Hard part: how do you execute the code**
  - Working solutions just have a VM
  - Also, detecting when things go wrong is still hard
- **It may actually be easier to fuzz x86**
  - Execution is easy
  - Just look for segfaults



https://github.com/crytic/echidna

Static Analysis

- **Functionally, about the same as ever**
- **More popular than fuzzing, less than symbolic execution**
  - Maybe a dozen linters, fewer static analyzers
  - How you count depends where the line is drawn
- **Solidity is an undergrad class project language**
  - Start with an undergrad class project analyzer and find some real bugs
  - Look at writes to state variables
  - Do some dataflow
- **Hard part: write a bunch of heuristics**

SLITHER

https://github.com/crytic/slither

Big picture

- **Code correctness tools work most places**
- **Developer experience is frequently** `./find_bugs`
- **Once people try these methods, they love them!**
  - Devs from traditional backgrounds are blown away
  - We give demos and see adoption during the talk
- **Solidity is no help, but code is getting better!**
  - This ends up not mattering because testability wins

Analyses that work on real code are huge wins for everyone

Why isn’t real life like this?

1. Code is huge
2. Abstraction boundaries are broken
3. Referential transparency is rare
4. Nothing is reproducible

Where do we go from here?

Bright spot 1: programmer goals are changing

- **Old mentality: programmers ship features**
  - Win by shipping more stuff
  - “10x engineers” produce 10x more code
  - Lines of code == productivity
- **New mentality: programmers make software that works**
  - Sysadmins → DevOps
  - Security works with developers, doesn’t just block their code
  - Test engineering, CI work, observability are growing fields
  - VCs have noticed this works really well

Bright spot 2: languages are getting better

- **Compilers are where the biggest security wins happen**
- **We’re finally souring on dynamic types!**
  - Python, Ruby, PHP, are starting to get types
  - Typescript, Hack, Crystal, are starting to get popular
- **“Everything is an object” → “everything is a function”**
  - People care about pure functions
  - Moving from a for loop to a map kills a state variable
- **Package managers are getting reproducible**

Bright spot 3: reproducible environments

- **Docker ensures dev and prod use the same environment**
  - Being able to deploy is nice
- **Build systems are sandboxing now**
  - stack, virtualenv, cargo
- **NixOS is on the horizon**

Standardizing where our programs run is a force multiplier for everything: dev, prod, and security

Now is the time

- **Academic work that works IRL has been near-impossible**
  - Analyzing 90s enterprise code kills your spirit
- **Successful so far: simple analysis, tons of heuristics**
  - IDA beats Binary Ninja at so many small things
  - Most of the CGC’s output went nowhere
- **Now, we’re at an inflection point**
  - More potential than ever for applicable research
  - Not enough people talking across the gap that remains

What can we learn from blockchain: redux

If blockchain’s testability can happen accidentally, then we can do more on purpose

- We need more devs using great tools
- There are more domains where this can happen
- Blockchain proves that good tools *win*
- Imagine the EVM, but deliberate

How can we help?

- **Trail of Bits doesn’t want to just consume research**
  - We work with grad students
  - We guest-lecture
  - We pay for research
  - Can we do more? Let us know: dan@trailofbits.com
- **Review our references about blockchain security**
  - https://blog.trailofbits.com
  - https://github.com/crytic
  - https://github.com/trailofbits/publications