Smart Contract Security

In academia and beyond

Phil(ip) Daian : IC3 @ Cornell

Devcon2 2016

IC3 The Initiative For CryptoCurrencies & Contracts

CORNELL UNIVERSITY FOUNDED A.D. 1865

Who is IC3?

- Research hub: Cornell University, Cornell Tech, UC Berkeley, UIUC and the Technion
- Cryptocurrency / smart contract focus

(our dashing directors)

- 12 faculty (at last count), students at all levels

… with special thanks to

- The Ethereum foundation!

- Our industry partners

Chain Digital Asset IBM

… and more to be announced soon.

Including you? Contact us through initc3.org

5 Grand Challenges

- Scaling / Performance

  Solidus, Bitcoin-NG, Miniature World, Fruitchain, Falcon, HoneyBadger
- Correctness

  FLAC, Theoretical Foundations, Hawk
- Confidentiality

  Hawk, Town Crier, Solidus
- Authenticated Data

  Town Crier, Virtual Notary, EtherScrape
- Safety / Compliance

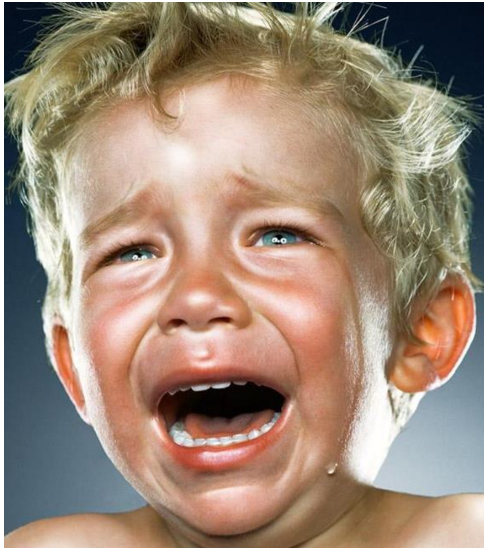
  Gyges

This talk

- High level, not comprehensive
- Overview, suggestions for practitioners

- Parallels to safety-critical software

The Problem



Security more closely tied to correctness than anywhere

Adversarial environment, public code, bad actors strongly incentivized

The Three Prongs

- **Formal Verification and Specification**
  what are we building and how can we check it?

- **Escape Hatches**
  how can we react to the unforeseen?

- **Bug Bounties**
  how can we address perverse incentives?

Formal Verification!

"The priest heard you finished the Functional Specification Document and wanted to witness the miracle."

Formal Verification! The good

- **Specification as a virtue**: know what you're building
- Specifying code helps you understand it
- Specifications of lower layers aid understanding

- English specifications are not enough; admit ambiguity
- Formal specifications can serve as fork criteria – EVM specs diverge from implementation, fork clear
- Obviously, specs help find bugs, can generate tools

Formal Verification! The work

Oyente – “Making Smart Contracts Smarter” – Luu et. Al

Builds on Ethereum Yellow Paper (=awesome!)

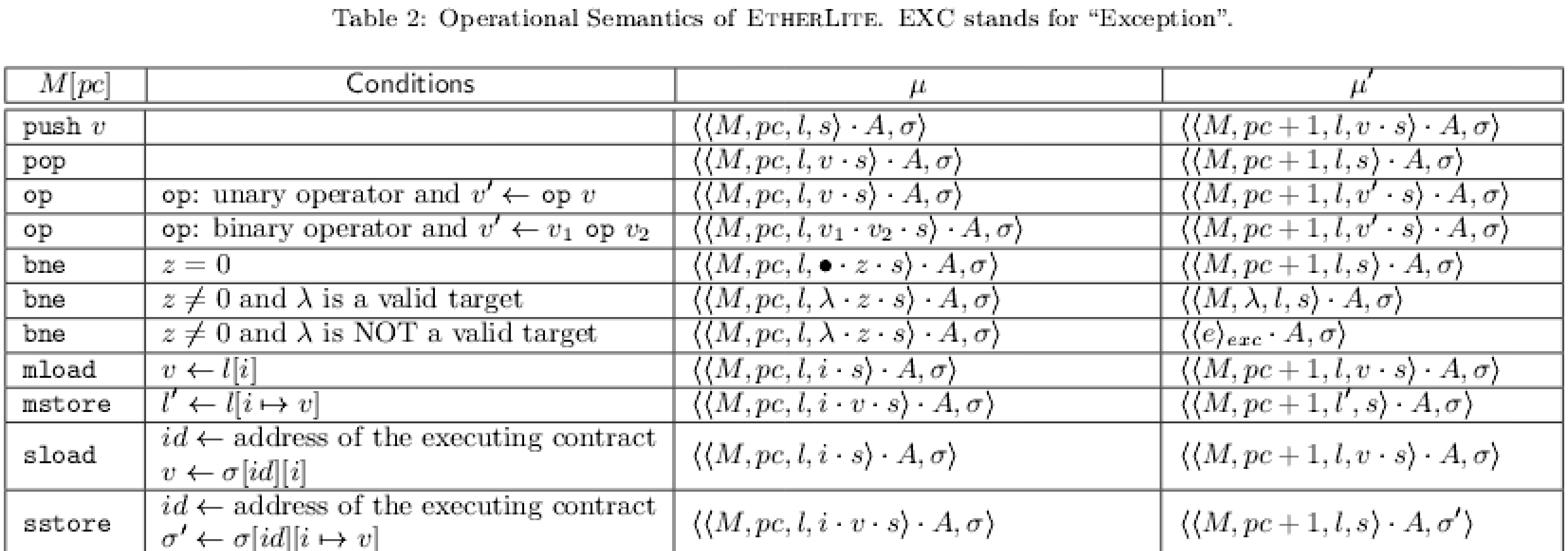
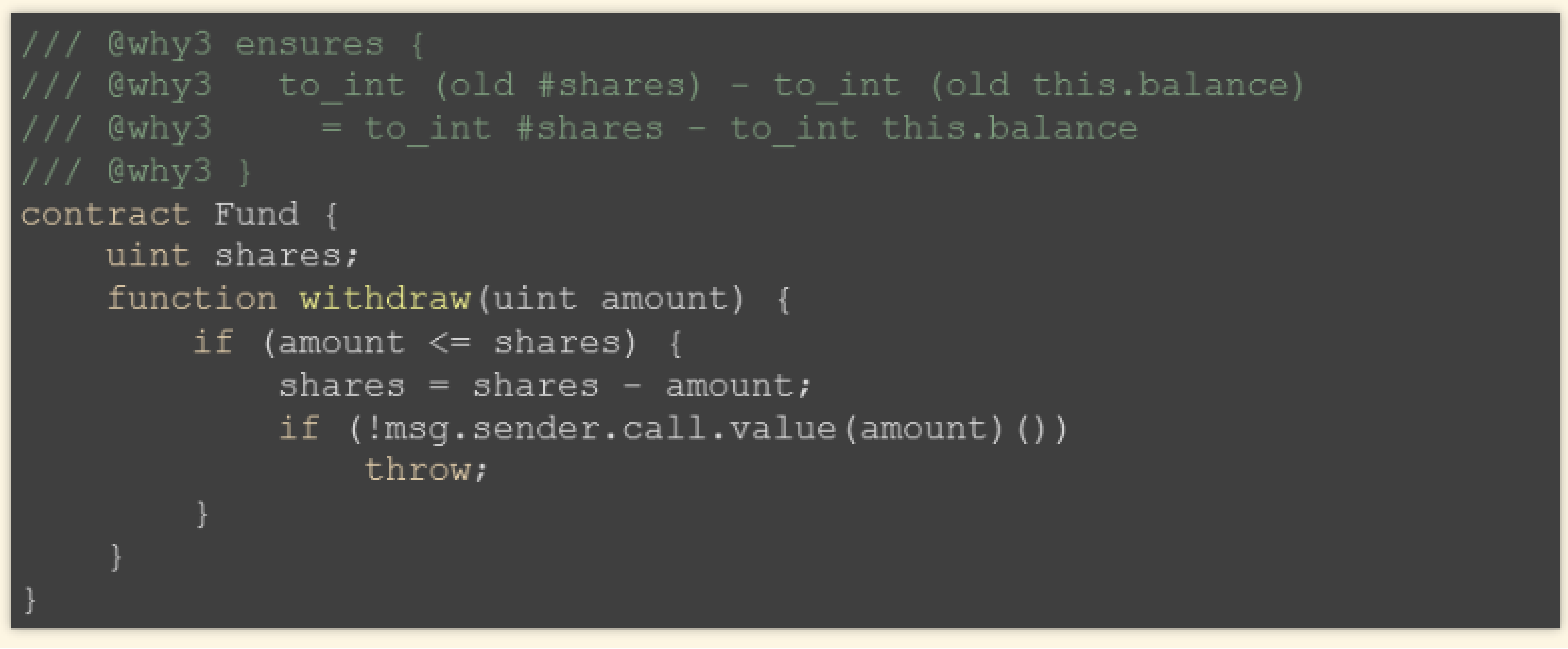


Table 2: Operational Semantics of EtherLite. EXC stands for “Exception”.

$M[pc]$	Conditions	μ	μ'
push v		$\langle\langle M, pc, l, s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, v \cdot s\rangle \cdot A, \sigma\rangle$
pop		$\langle\langle M, pc, l, v \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, s\rangle \cdot A, \sigma\rangle$
op	op: unary operator and $v' \leftarrow$ op v	$\langle\langle M, pc, l, v \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, v' \cdot s\rangle \cdot A, \sigma\rangle$
op	op: binary operator and $v' \leftarrow v_1$ op v_2	$\langle\langle M, pc, l, v_1 \cdot v_2 \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, v' \cdot s\rangle \cdot A, \sigma\rangle$
bne	$z = 0$	$\langle\langle M, pc, l, \bullet \cdot z \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, s\rangle \cdot A, \sigma\rangle$
bne	$z \neq 0$ and λ is a valid target	$\langle\langle M, pc, l, \lambda \cdot z \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, \lambda, l, s\rangle \cdot A, \sigma\rangle$
bne	$z \neq 0$ and λ is NOT a valid target	$\langle\langle M, pc, l, \lambda \cdot z \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle e\rangle_{exc} \cdot A, \sigma\rangle$
mload	$v \leftarrow l[i]$	$\langle\langle M, pc, l, i \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, v \cdot s\rangle \cdot A, \sigma\rangle$
mstore	$l' \leftarrow l[i \mapsto v]$	$\langle\langle M, pc, l, i \cdot v \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l', s\rangle \cdot A, \sigma\rangle$
sload	$id \leftarrow$ address of the executing contract $v \leftarrow \sigma[id][i]$	$\langle\langle M, pc, l, i \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, v \cdot s\rangle \cdot A, \sigma\rangle$
sstore	$id \leftarrow$ address of the executing contract $\sigma' \leftarrow \sigma[id][i \mapsto v]$	$\langle\langle M, pc, l, i \cdot v \cdot s\rangle \cdot A, \sigma\rangle$	$\langle\langle M, pc+1, l, s\rangle \cdot A, \sigma'\rangle$

Formal Verification! The work

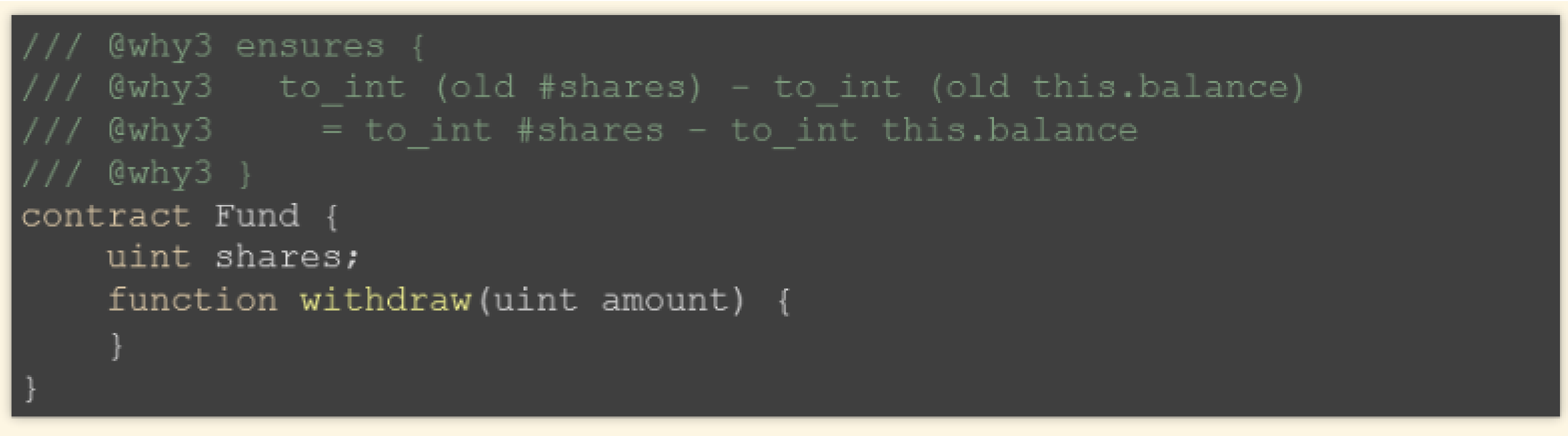
“Formal Verification for Solidity” - Dr. C. Reitwiessner



```
/// @why3 ensures {
/// @why3   to_int (old #shares) - to_int (old this.balance)
/// @why3     = to_int #shares - to_int this.balance
/// @why3 }
contract Fund {
    uint shares;
    function withdraw(uint amount) {
        if (amount <= shares) {
            shares = shares - amount;
            if (!msg.sender.call.value(amount)())
                throw;
        }
    }
}
```

Formal Verification! The work

“Formal Verification for Solidity” - Dr. C. Reitwiessner



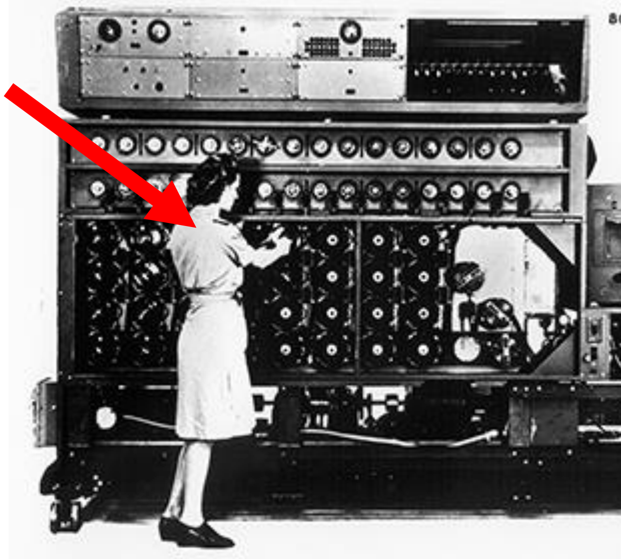
```
/// @why3 ensures {
/// @why3   to_int (old #shares) - to_int (old this.balance)
/// @why3     = to_int #shares - to_int this.balance
/// @why3 }
contract Fund {
    uint shares;
    function withdraw(uint amount) {
    }
}
```

Formal Verification! The gaps

- **Specification is hard!** Some properties? Impossible
- When you output a proof, you’re trusting tools
- **Semantics!** Can be unclear or ambiguous
- **Any good tool must define semantics**
- How to audit tools? Test of time?

- Right now: experts required, multiple PhDs to do right
- Incompleteness and undecidability result

Escape hatches! The good



- So, we can’t always verify. We need
- Humans in the loop; tried and tested
- Covers if verification, bounties fail
- In theory, reduces need for **forks**

- Parallels to contract law
- safety-critical systems –

  would you build a nuclear plant with no killswitch?

Escape hatches! The work

“Setting Standards for Altering and Undoing Smart Contracts” - Bill Marino, Ari Juels

- Parallels to “legacy” contract law
- **Termination** by right
- **Rescission** by right, court
- **Modification** by right, agreement
- **Reformation**
- *(and some code mirroring these)*

Escape hatches! The gaps

- How to verify escape hatch code?
- Where to put escape hatches?
  EVM layer (high assurance, less general)?
  Compiler (moderate assurance, some generality)?
  Contract libraries (flexible assurance, full generality)?

- Potential for abuse – exploits, bad incentives, etc.
- Can you think of a badly made escape hatch? (Hint: 666)

Bug bounties! The good

- Incentive structure is totally broken without bounties
- Attackers: incentivized to attack
- Defenders: limited to no financial incentives

Bug bounties! The good

The poor man's formal verification

-or-

"decentralized censorship-resistant anti-fragile incentive-compatible crowdsourced verification"

Bug bounties! The work

augur GNOSIS

- “Assert Guards: Towards Automated Code Bounties & Safe Smart Contract Coding on Ethereum” Simon de la Rouviere
- Ethereum.org -> best practices for smart contracts
- medium.com -> DAO challenge!
- And more

Bug bounties! The gaps

- With prediction markets: how to avoid bad incentives?
- How to create trustless bounties? Trustless payout? *Without* leaking exploit to testnet, trusting authors?
- Impact of competition?

- How do we define conditions for bug bounties?
- SGPs, SGX, zk-SNARKs?
- Bug bounties for subtle issues – aka incentive flaws?

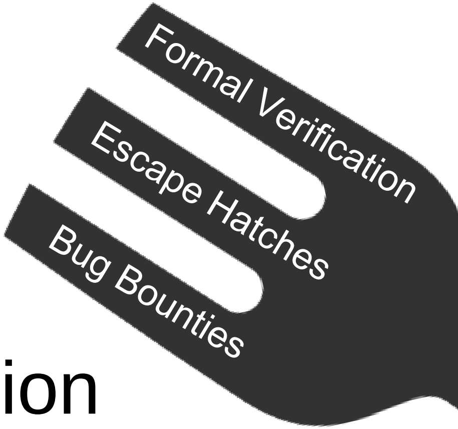
Don’t forget traditional SE!

Tests, fuzzing, static and dynamic analysis, phased deployment/upgrades, etc.

Takeaways : Secure Contracts?
Lots of work still to be done…

- This ecosystem must/will develop **good** formal tools
- Be skeptical! Formal tools are not a silver bullet

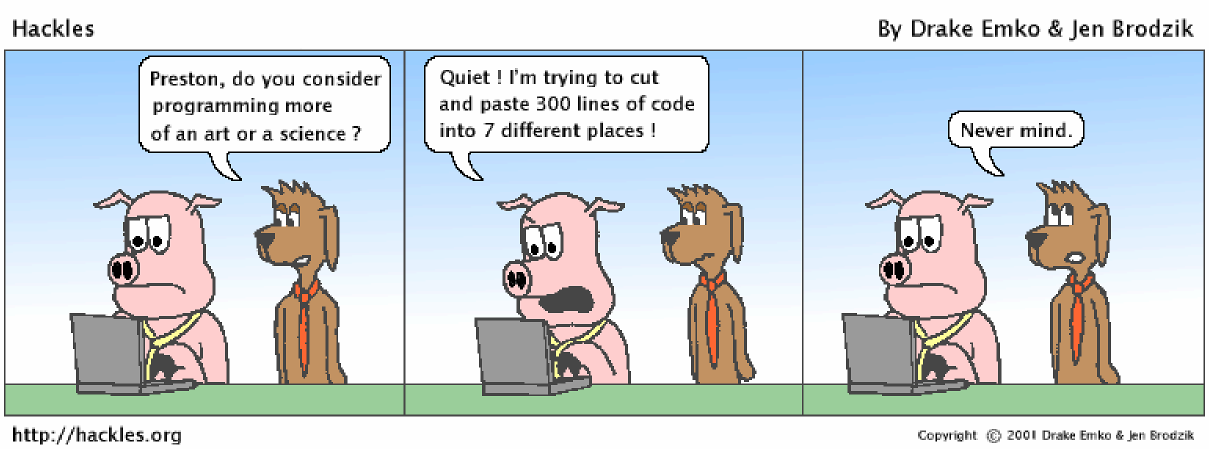
- All contracts: think humans in the loop
- *Consider* parallels to "legacy" contracts



- Bug bounties can be stop-gap for verification
- Without bug bounties, attacker incentives are perverse

Thanks!

- Learn more @ **initc3.org**
- Read our papers @ **initc3.org/publications**
- We're always open to industry collaborations!


Hackles
By Drake Emko & Jen Brodzik
Preston, do you consider programming more of an art or a science ?
Quiet ! I'm trying to cut and paste 300 lines of code into 7 different places !
Never mind.
http://hackles.org
Copyright © 2001 Drake Emko & Jen Brodzik