Smart Contract Security

Florian Tramèr
Nicolas Kokkalis

Agenda

- Smart Contract Verification
  - Why? More examples of bugs!
  - How?

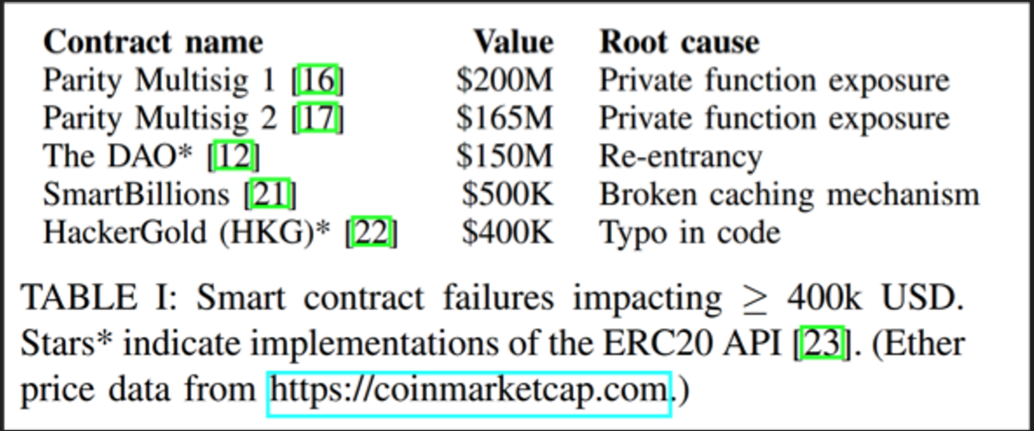
- Beyond bugs in code: networks, miners and incentives
  - Attacks by miners: reorder, delay, drop transactions
  - Front running
  - Commit-reveal & Submarine sends
  - Other mining attacks

- Randomness, Confidentiality, Authenticity, Fairness
  - Randomness and secrecy in public contracts
  - Confidential transactions & contracts
  - Data oracles
  - Fair exchange and crypto for crime

Smart Contract Verification: Why?

Pervasive bugs in an adversarial computation environment:

- All user input is arbitrary (overflows / underflows)
- Data/computation flow is non-trivial (reentrancy, deadlocks)
- Extra care about termination (gas costs)
- Tricky specifications...



Contract name	Value	Root cause
Parity Multisig 1 [16]	$200M	Private function exposure
Parity Multisig 2 [17]	$165M	Private function exposure
The DAO* [12]	$150M	Re-entrancy
SmartBillions [21]	$500K	Broken caching mechanism
HackerGold (HKG)* [22]	$400K	Typo in code

TABLE I: Smart contract failures impacting $\geq$ 400k USD. Stars* indicate implementations of the ERC20 API [23]. (Ether price data from https://coinmarketcap.com.)

Hildenbrandt et al.

Example: charity considered harmful

```
ERC20_ok {
    uint256 num_tokens;
    function totalSupply() {
            return num_tokens;
    }
}
```

```
ERC20_bad {
    function totalSupply() {
            return
this.balance;
    }
}
```

“Recently” discovered ~~bug~~/feature: send funds to contract via `selfDestruct` without triggering default function. Also:

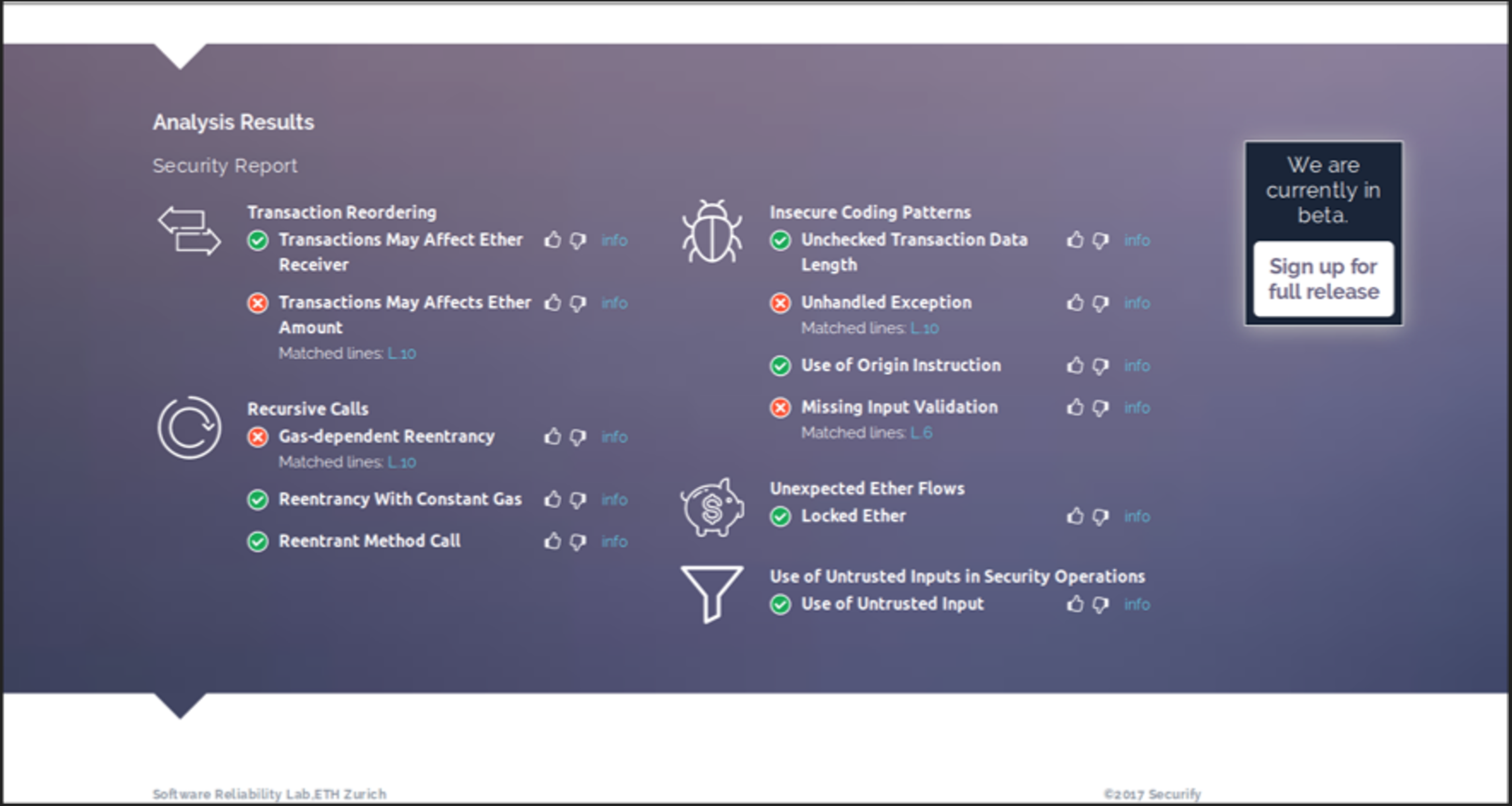
- mine to a contract
- send money to a not yet existent contract! (more on this later)

Smart Contract Verification: How?

1. Static or symbolic analysis of sourcecode / bytecode to find common errors/anti-patterns
    - Quite a few false-positives (not too bad for small contracts)
    - Only finds things it knows to look for
    - Won’t detect (most) logic errors

Examples:

- Oyente (https://github.com/ethereum/oyente)
- Securify (https://securify.ch/)
- Mythril (https://github.com/ConsenSys/mythril)
- Remix (http://remix.ethereum.org/)



Analysis Results

Security Report

Transaction Reordering

- ✓ Transactions May Affect Ether Receiver — info
- ✗ Transactions May Affects Ether Amount — info
  Matched lines: L.10

Recursive Calls

- ✗ Gas-dependent Reentrancy — info
  Matched lines: L.10
- ✓ Reentrancy With Constant Gas — info
- ✓ Reentrant Method Call — info

Insecure Coding Patterns

- ✓ Unchecked Transaction Data Length — info
- ✗ Unhandled Exception — info
  Matched lines: L.10
- ✓ Use of Origin Instruction — info
- ✗ Missing Input Validation — info
  Matched lines: L.6

Unexpected Ether Flows

- ✓ Locked Ether — info

Use of Untrusted Inputs in Security Operations

- ✓ Use of Untrusted Input — info

We are currently in beta.

Sign up for full release

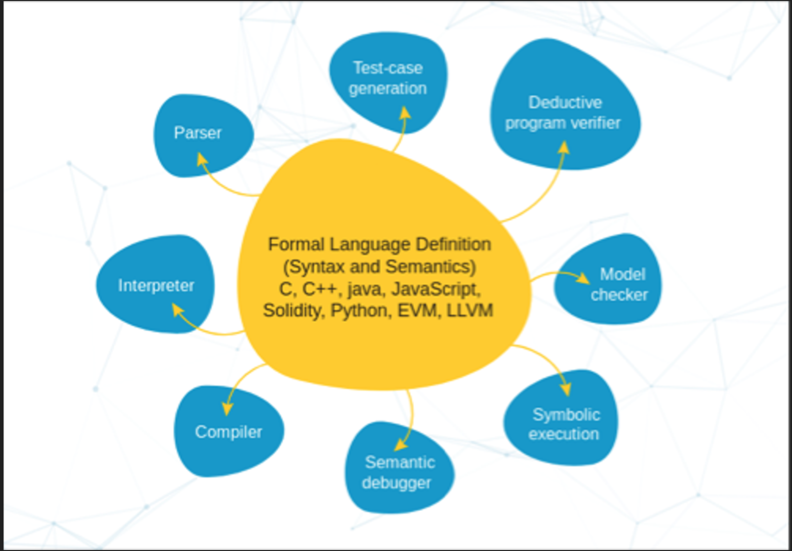
Software Reliability Lab,ETH Zurich

©2017 Securify

Smart Contract Verification: How?

2. Verify code against a formal specification
    - Lots of work (but probably worth it?)
    - Requires a full formal semantics for the EVM
    - Won’t catch specification-level bugs

Examples: KEVM (https://github.com/kframework/evm-semantics)


Formal Language Definition (Syntax and Semantics) C, C++, java, JavaScript, Solidity, Python, EVM, LLVM
Parser
Test-case generation
Deductive program verifier
Interpreter
Model checker
Compiler
Semantic debugger
Symbolic execution

Dynamic Verification

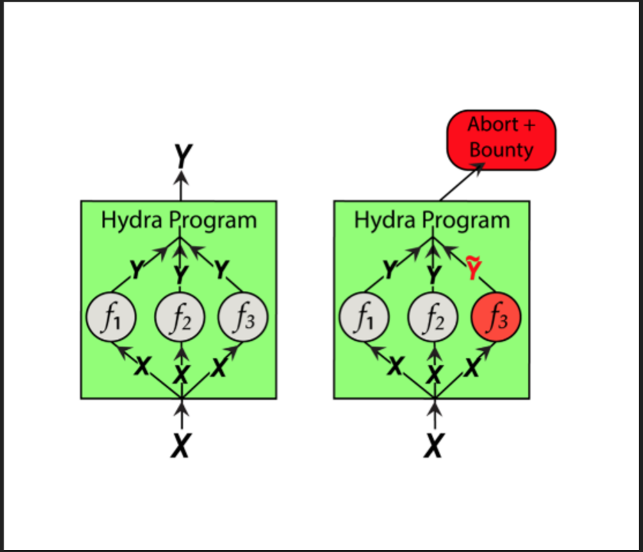
- Runtime checks: e.g., underflow/overflow checks, see Vyper
- Hydra: multiversion programming for smart contracts (Breidenbach et al.)

The Hydra Project [alpha]

Hydra is a cutting-edge Ethereum contract development framework for:

decentralized security and bug bounties
rigorous cryptoeconomic security guarantees
mitigating programmer and compiler error

thehydra.io



Beyond Bugs in Code

Congrats, your contract is now bug free! (it isn't)

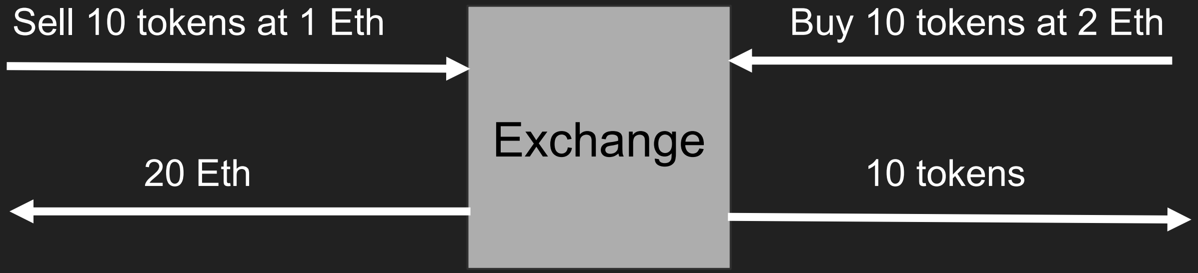
What can miners do to undermine your contract?

- Delay, drop or reorder transactions
- No guarantee of "meaningful" total ordering of transactions

Only miners? No!

- Attacker can observe (maybe even tamper with) the network layer
- Attacker can bribe miners (or just pay higher transaction fees than you)

Reordering transactions: on-chain exchange



Arbitrage / Frontrunning

Miner sees the following transactions to be mined:

- Sell 10 tokens for 1 Eth
- Buy 10 tokens for 2 Eth

A miner can insert it's own transactions:

- Sell 10 tokens for 1 Eth
- Buy 10 tokens for 1 Eth
- Sell 10 tokens for 2 Eth
- Buy 10 tokens for 2 Eth

1. Other users could achieve the same by e.g., setting transaction fees appropriately (if miners are honest)
2. Not typically illegal in finance but in cryptocurrencies, miners are in an (unfairly?) advantageous position

Profit!!!

Preventing Frontrunning with Commit-Reveal

- Cryptographic commitment:
  - `Commit`(msg, r) -> c => hides the message
  - `Open`(c, msg, r) -> {0, 1} => only evaluates to 1 if c == Commit(msg, r)
  - Example: H(msg || r)

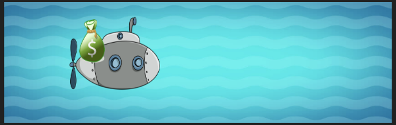
- New exchange contract:
  - Accepts *commitments* of buy/sell orders. Stores commitment and block number
  - Accepts opening of order iff corresponding commitment is older than k blocks (e.g, 1h)

    => too late to frontrun

- Miner/user could “optimistically” frontrun:
  - Commit to many different orders
  - Only open the ones that are useful

    => need to make commits “expensive”: e.g., place funds in escrow and reimburse if opened

Hidden Commitments: Submarine sends



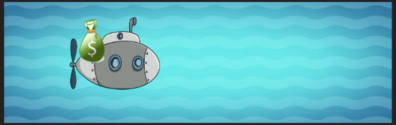
What if you don’t want others to know that you placed an order before its confirmed (or too late to do anything about it)? E.g., useful in auctions

Solution: put your commitment “somewhere” on chain and later point back to it
=> how do we ensure commits are expensive?

Solution’: combine with “proof of burn”: e.g., send money to address `0x0`
=> wasteful

Solution’’: send funds to “random” address so that they can later be re-claimed
=> Submarine sends

Hidden Commitments: Submarine sends



Submarine sends (post-Metropolis version):

1. Send your commit and funds to address *Addr* = `H(0x123,data,Forwarder)`
2. Later, send the opening, and *Addr* to the auction contract (at address `0x123`)
3. The auction contract can spawn a Forwarder at address *Addr* and call it to recover the funds

```
Forwarder {
  address addrContract = 0x123;

  function () {
   if (msg.sender == addrContract)
     addrContract.send(this.balance);
  }
}
```

Other mining attacks & incentive issues

Selfish mining (https://arxiv.org/abs/1311.0243)

- When miner finds a block, “sit on it” instead of directly announcing it
- If miner has a secret chain of ≥2 blocks, he can use it to invalidate other miners’ work
- In some settings, a miner with <½ hash power can control the whole network

Miner’s dilemma (https://arxiv.org/abs/1411.7099)

- Miners should verify transactions. But, no real incentive to do so
- Miners that don’t verify transactions can spend more time mining!

Cryptoeconomics (https://projectchicago.io/)

- How to properly price commodities used in blockchains (network, computation, etc.)?
- Gastoken: exploiting gas refunds for price arbitrage: (http://www.gastoken.io)
- Incentives at the P2P layer (https://ethresear.ch/t/incentivizing-a-robust-p2p-network-relay-layer/1438)

Randomness, Confidentiality, Authenticity, Fairness

- How can contracts make randomized, fair, unbiased decisions?
- How can contracts deal with confidential data?
- How do we get trusted data into contracts?
- How do we fairly exchange digital goods?

Randomness: Implementing a lottery

Use a “randomness beacon” to randomize the contract’s decision:

1. Use blockhash or block number
   => Miner can decide not to release a block to bias the randomness

2. Let users commit to randomness r1, …, rn. Then, open all the commitments and use r = r1 + … + rn as a random seed
   => users can bias the results by not opening (needs penalties)

3. Promising approaches: proofs of delay, verifiable secret sharing
   http://www.jbonneau.com/doc/BGB17-IEEESB-proof_of_delay_ethereum.pdf
   https://eprint.iacr.org/2016/1067.pdf

Confidentiality: A rock-paper-scissors game

Fun (and educational) read: https://eprint.iacr.org/2015/460.pdf

- If players send their action "in the clear"
  => wait for other player to go first
  => similar to frontrunning, so…

- Commit-reveal: both players commit to their action first. Then reveal.
  => losing player can abort

- Add penalties on abort, or a deadline after which we pay the honest party

Confidentiality contd.

Commit-reveal works if data should be secret for a *finite time period*

- Actions in games, auction bids, market offers, etc.

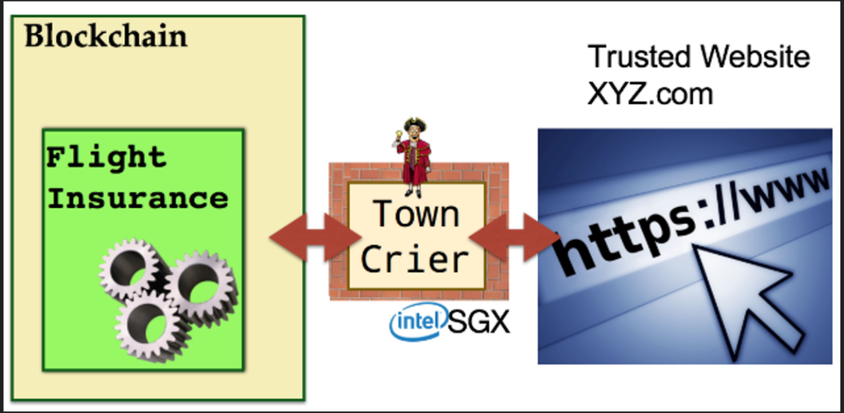
More generally: confidential transactions

- Hide combination of source, destination, amount for transactions
- E.g., ZCash, Monero, Mixer networks, Bulletproofs, etc.

Data authenticity

- Today's Dapps: Tokens & games
  - Why? No need for external data…

- How to connect smart contracts with the "real world"?
  => Data oracles



Examples:

- Oraclize (http://www.oraclize.it/)
- Smartcontract (https://www.smartcontract.com/)
- TownCrier (http://www.town-crier.org/)

Fairness

Fair exchange:

- Party A has input x, party B has input y
- Either A get y and B gets x or **neither**

Fair payment for digital goods: exchange cryptocurrency for data

Bitcoin: hash-locked transactions

- A sends `Enc(k; data)` and `Proof(data = good AND H(k) = h)` to B
- B creates a transaction that sends $$ to the party that provides a preimage of h

Issues:

- A can decide to abort, or B can decide that `Proof` is enough
- Requires off-chain interaction

Fair-Exchange with smart contracts

- B creates a contract with $$ asking for data
- A sends `Enc(k; data)` and `Proof(data = good AND H(k) = h)` to contract
- contract checks `Proof` and sends $$ to A

Example applications:

- Sealed Glass Proofs (https://eprint.iacr.org/2016/635)
  Use smart contracts (and trusted hardware) for a fair bug-bounty scheme

- Criminal Smart Contracts (https://eprint.iacr.org/2016/358)
  Fun read about (hypothetical) bad things you could trade with smart contracts

Crypto for crime

Cryptocurrencies are (pseudo)anonymous, decentralized, under regulated, ...
Good for crime!

- Ransomware
- Mining malware
- Wallet theft
- Illegal exchanges (e.g., Silk road)
- Hypothetical (or not) Criminal Smart Contracts:
  - Selling secrets (e.g., Darkleaks in Bitcoin)
  - E.g., compromised passwords, accounts, crypto keys
  - Assassination, terrorism, etc.

Additional References

Best practices:

- http://solidity.readthedocs.io/en/v0.4.23/security-considerations.html
- https://consensys.github.io/smart-contract-best-practices
- https://paritytech.io/new-smart-contract-development-processes/

Miner attacks:

- http://hackingdistributed.com/2017/08/28/submarine-sends/
- http://hackingdistributed.com/2017/06/19/bancor-is-flawed/