Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach

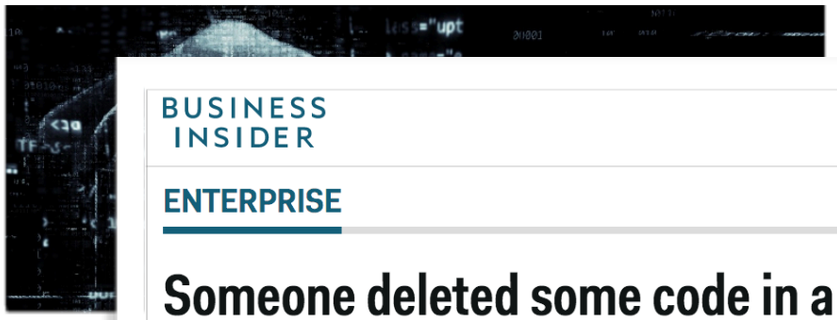
Anastasia Mavridou[1] and Aron Laszka[2]

[1] Vanderbilt University
[2] University of Houston

freeCodeCamp

HOME DEV DESIGN DATA | LEARN TO CODE FOR

A hacker stole $31M of Ether—how it happened, and what it means for Ethereum

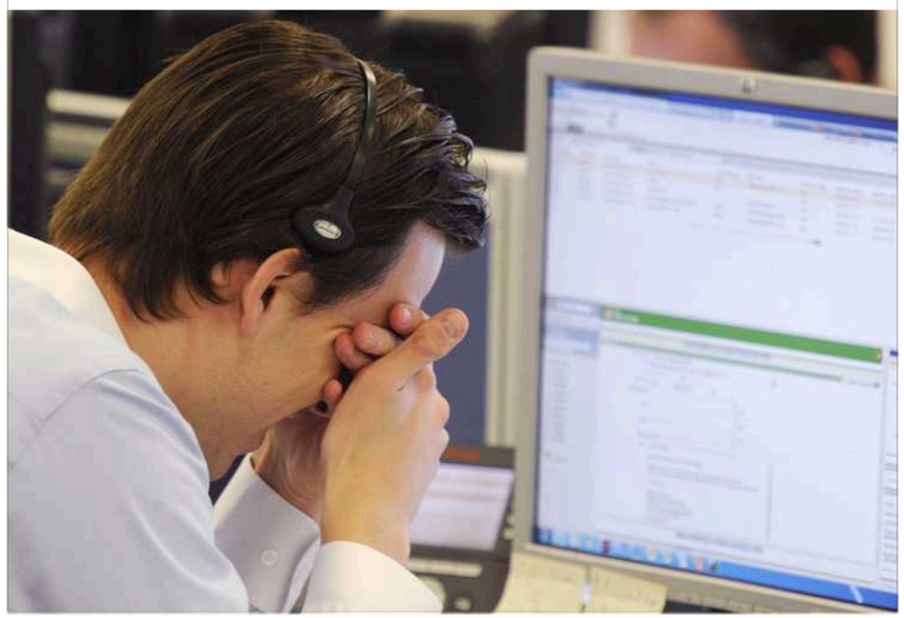


BUSINESS INSIDER

ENTERPRISE

Someone deleted some code in a popular cryptocurrency wallet — and as much as $280 million in ether is locked up

Becky Peterson Nov. 7, 2017, 6:29 PM 145,211

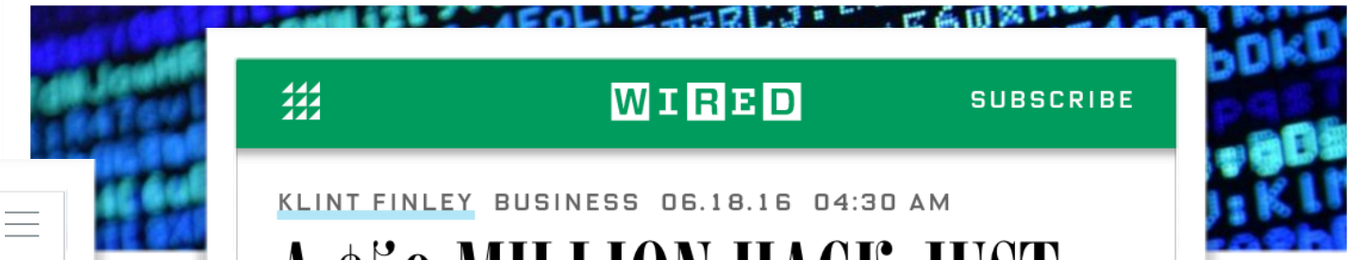


BBC Home News Sport Weather Shop More

NEWS

Technology

Hack attack drains start-up investment fund



WIRED SUBSCRIBE

KLINT FINLEY BUSINESS 06.18.16 04:30 AM

A $50 MILLION HACK JUST SHOWED THAT THE DAO WAS ALL TOO HUMAN

Smart Contract Insecurity

- Smart contracts are riddled with bugs and security vulnerabilities
- A recent automated analysis of 19,336 Ethereum contracts
  - 8,333 contracts suffer from at least one security issue

Luu, Loi, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. "Making smart contracts smarter." *ACM CCS*, 2016.

Millions of Dollars In Ethereum Are Vulnerable to Hackers Right Now

Researchers discovered 34,200 buggy smart contracts on Ethereum.

Nikolic, Ivica, Aashish KolluriChu, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. “Finding the Greedy, Prodigal, and Suicidal Contracts at Scale.”arXiv:1802.06038, 2018.

Security Vulnerabilities are a Serious Issue

- Smart contracts handle financial assets of significant value
  - Value held by Ethereum contracts is 12,205,706 ETH or $10B
- Smart contract bugs cannot be patched
  - Once a contract is deployed, its code cannot be changed
- Blockchain transactions cannot be rolled back
  - Once a malicious transaction is recorded it cannot be removed
- Well… actually…
  - It can be rolled back with a hard fork of the blockchain

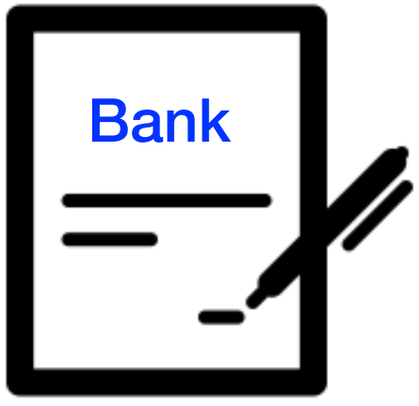
Common Vulnerabilities

- Examples of common vulnerabilities [1]
  - Reentrancy
  - Transaction-Ordering Dependency

[1] Luu, Loi, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. "Making smart contracts smarter." *ACM CCS*, 2016.

Reentrancy

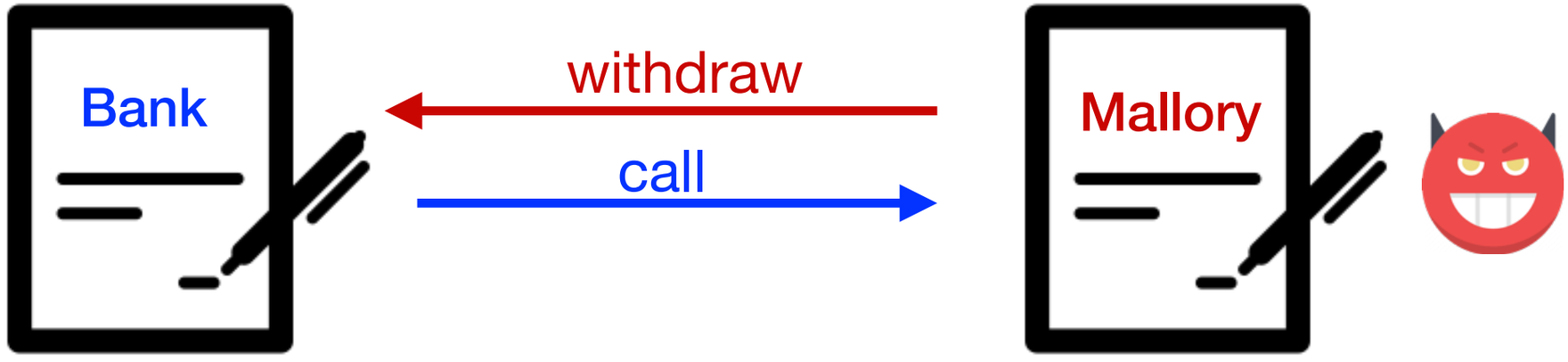
- In Ethereum, when there is a function call
  - The caller has to wait for the call to finish
  - A malicious callee might take advantage of this




```
  function withdraw(uint amount) {
    if (credit[msg.sender]>= amount) {
      msg.sender.call.value(amount)();
      credit[msg.sender]-=amount;
}}
```

Reentrancy

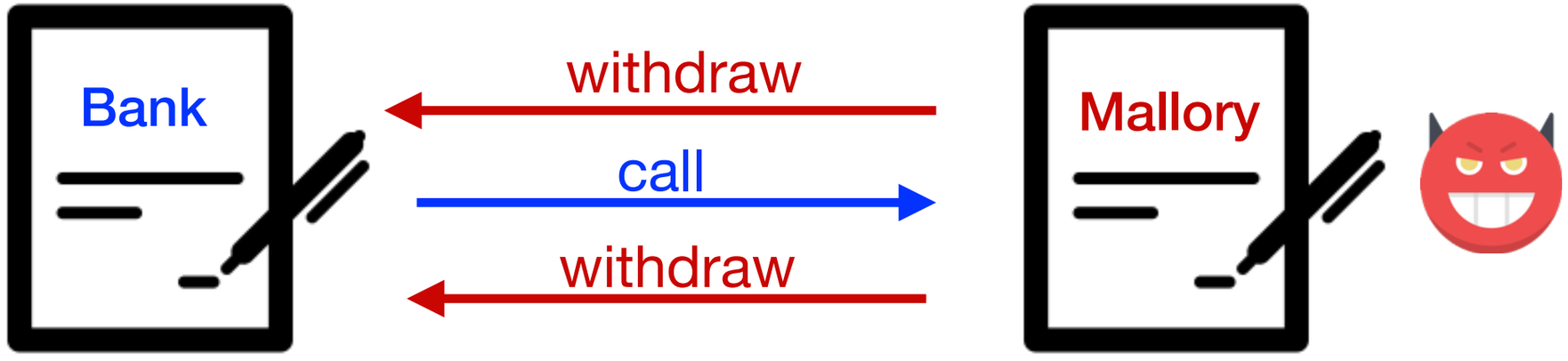
- In Ethereum, when there is a function call
  - The caller has to wait for the call to finish
  - A malicious callee might take advantage of this




```
  function withdraw(uint amount) {
    if (credit[msg.sender]>= amount) {
      msg.sender.call.value(amount)();
      credit[msg.sender]-=amount;
}}
```

Reentrancy

- In Ethereum, when there is a function call
  - The caller has to wait for the call to finish
  - A malicious callee might take advantage of this

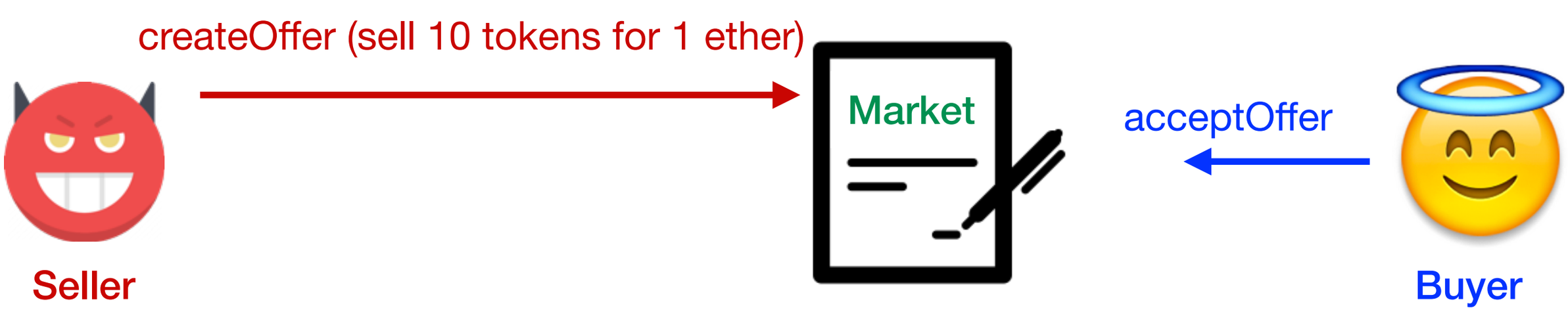



```
  function withdraw(uint amount) {
    if (credit[msg.sender]>= amount) {
      msg.sender.call.value(amount)();
      credit[msg.sender]-=amount;
}}
```

```
function() {
 bank.withdraw(bank.queryCredit(this));
}
```

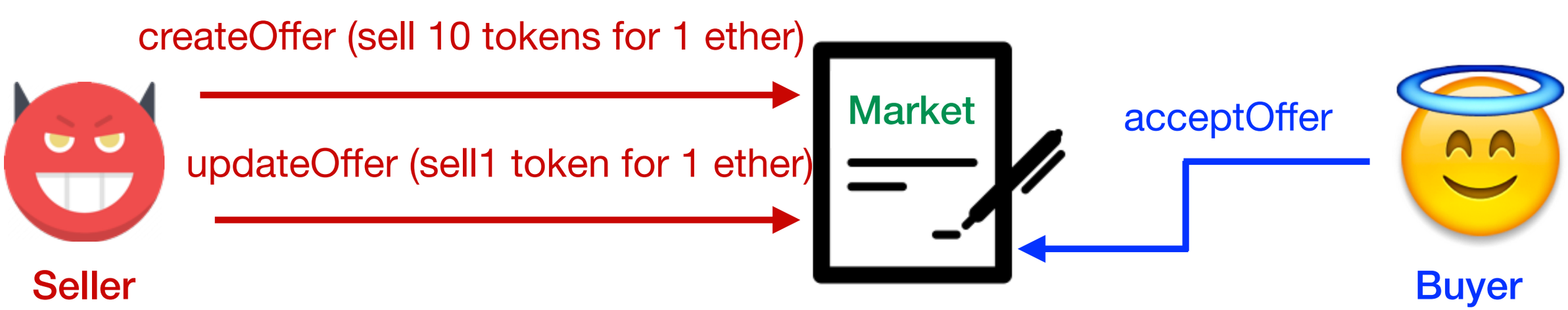
Transaction Ordering Dependency

- Also known as unpredictable state vulnerability
- The order of execution of function calls cannot be predicted
- No prior knowledge of a contract's state during call execution



Transaction Ordering Dependency

- Also known as unpredictable state vulnerability
- The order of execution of function calls cannot be predicted
- No prior knowledge of a contract's state during call execution

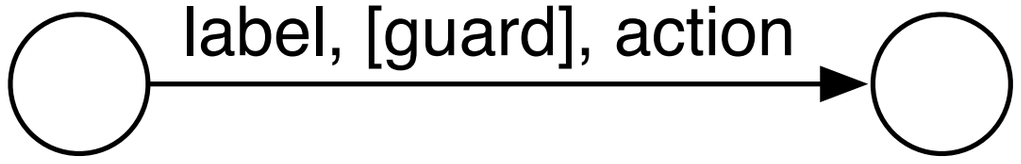


Our Motivation

- Vulnerabilities often arise due to the semantic gap
  - The assumptions developers make about execution semantics
  - The actual semantics
- Prior work:
  - Tools for identifying existing vulnerabilities
  - Tools for static analysis
  - Design patterns, e.g., Checks-Effects-Interactions
- We explore a different avenue
  - We want to help developers to create secure smart contracts
  - Correctness-by-design

Our Approach - Model Based Design

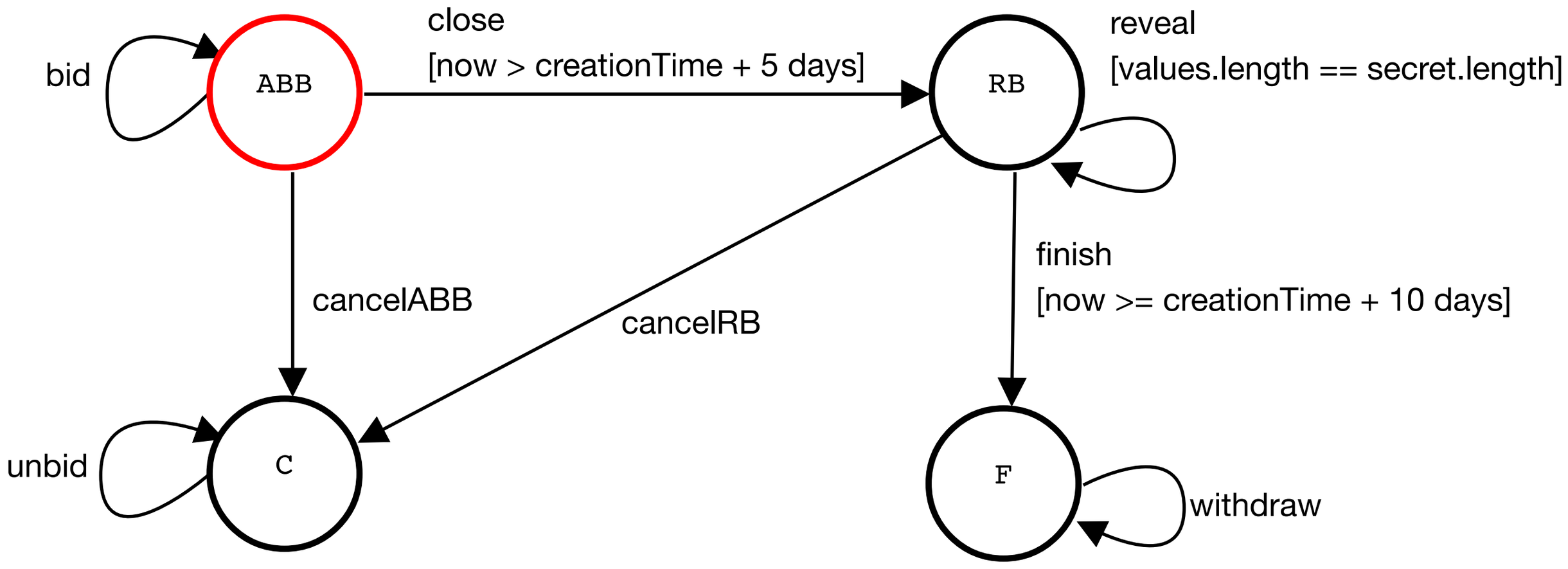
- We introduce a formal, transition-based language for smart contracts



  - A contract can be naturally represented by a transition system

- A Smart Contract is a tuple $(S, s_0, C, I, O, \rightarrow)$
  - S is a finite set of states
  - $s_0 \in S$ is the initial state
  - C, I, and O are finite sets of contract, input, and output variables
  - $\rightarrow \subseteq S \times \mathcal{G} \times \mathcal{F} \times S$ Is a transition relation
    - $\mathcal{G}$ Is a set of guards and $\mathcal{F}$ is a set of action sets

Example:
Blind Auction Contract as a Transition System



Our Approach - Model Based Design

- Advantages
  - High-level model → adequate level of abstraction
  - Rigorous semantics → amenable to formal verification
  - Code generation from transition systems to Solidity code
  - Plugins that implement security features and design patterns

Common Vulnerabilities and Design Patterns

- Examples of common vulnerabilities [1]
  - Reentrancy
  - Transaction-Ordering Dependency
- Most common design patterns [2]
  - Authorization
  - Time constraints

[1] Luu, Loi, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. "Making smart contracts smarter." *ACM CCS*, 2016.

[2] Bartoletti, Massimo, and Livio Pompianu. "An empirical analysis of smart contracts: platforms, applications, and design patterns." TSC in *FC*, 2017.

Examples of FSolidM Plugins

- Locking

```
bool private locked = false;
modifier locking {
  require(!locked);
  locked = true;
  _;
  locked = false;
}
```

Reentrancy

- Transition counter

```
uint private transitionCounter = 0;
modifier transitionCounting(uint nextTransitionNumber) {
  require(nextTransitionNumber == transitionCounter);
  transitionCounter += 1;
  _;
}
```


Transaction-Ordering Dependency

Examples of FSolidM Plugins

- Locking

```
bool private locked = false;
modifier locking {
  require(!locked);
  locked = true;
  _;
  locked = false;
}
```


Reentrancy

- Transition counter

```
uint private transitionCounter = 0;
modifier transitionCounting(uint nextTransitionNumber) {
  require(nextTransitionNumber == transitionCounter);
  transitionCounter += 1;
  _;
}
```

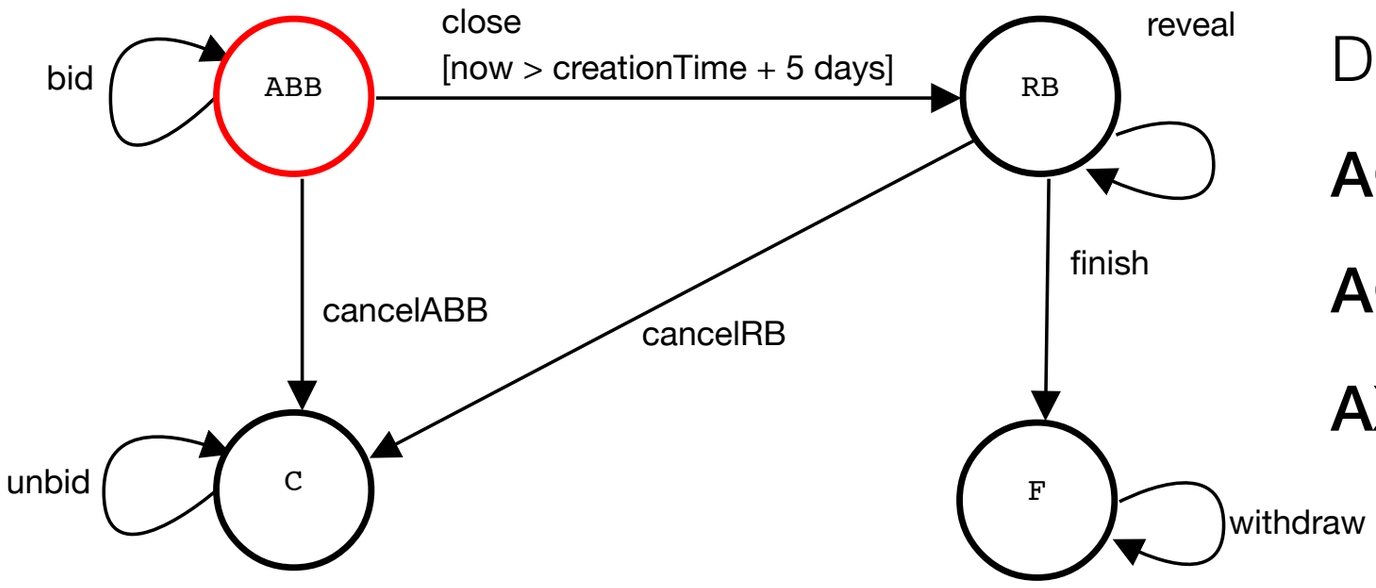
Transaction-Ordering Dependency

Ongoing Work on Verification

- NuSMV model checker to verify
  - Safety properties
    - e.g., a faulty state should not be reached
  - Deadlock freedom
  - Liveness properties
    - e.g., a state of the system will be eventually reached

Ongoing Work on Verification

- NuSMV model checker to verify
  - Safety properties
    - e.g., a faulty state should not be reached
  - Deadlock freedom
  - Liveness properties
    - e.g., a state of the system will be eventually reached



Deadlock freedom ✓

AG (close → **AG** !bid) ✓

AG (withdraw →

AX A [!withdraw **W** subtract]) ✓

Discussion

- Formal model, clear semantics, easy-to-use graphical editor
  - Decreasing the semantic gap
- Rigorous semantics
  - Amenable to analysis and verification
- Code generation + functionality and security plugins
  - Minimal amount of error-prone manual coding
- FSolidM source code: http://github.com/anmavrid/smart-contracts
- FSolidM also available at: http://cps-vo.org/group/SmartContracts

Thank you!