Security Smells in Smart Contracts

A static analysis survey on vulnerabilities in Solidity smart contracts

Mehmet Demir, Manar Alalfi, Ozgur Turetken, Alex Ferworn

Ryerson University

This subject is interesting since

+The blockchain technology is new and exciting

+Distributed applications (Dapp) concept is different. Distributed computing is known but applications to be distributed and getting executed in many places at the same time is new

+Nature of smart contract code is different

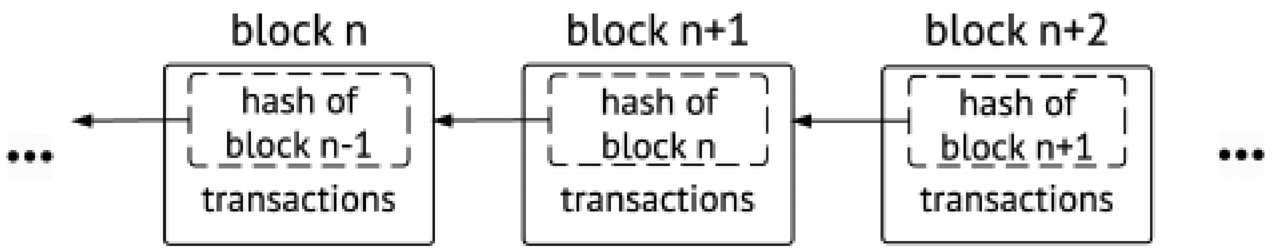
+Security issues related to smart contracts are different

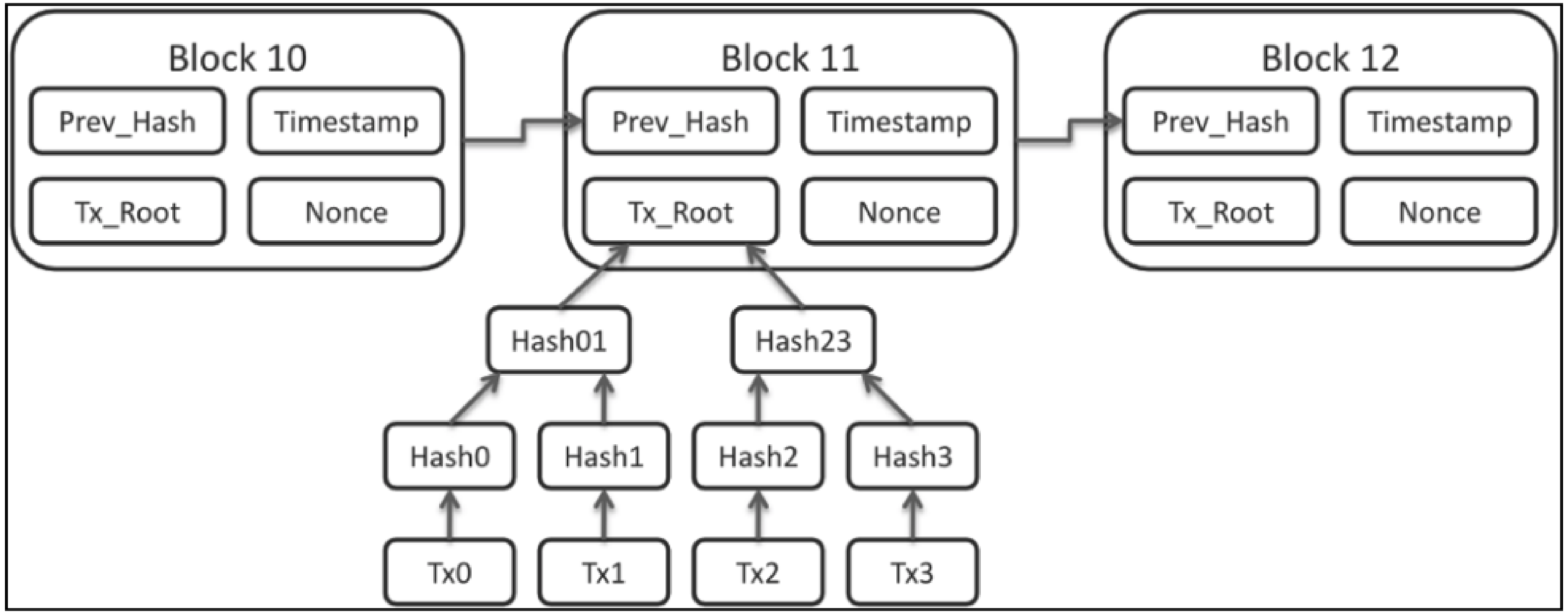
+They almost always handle money. Risk is high. Attackers have high motivation.

-The subject is new. There is not enough data.

-Limited analysis yet. Findings of the existing studies are limited. Many issues are related to the language and platform.

Blockchain





Ryerson University

Smart Contract

Involves parties

Computer Code

Added to blockchain

Triggered by event(s)

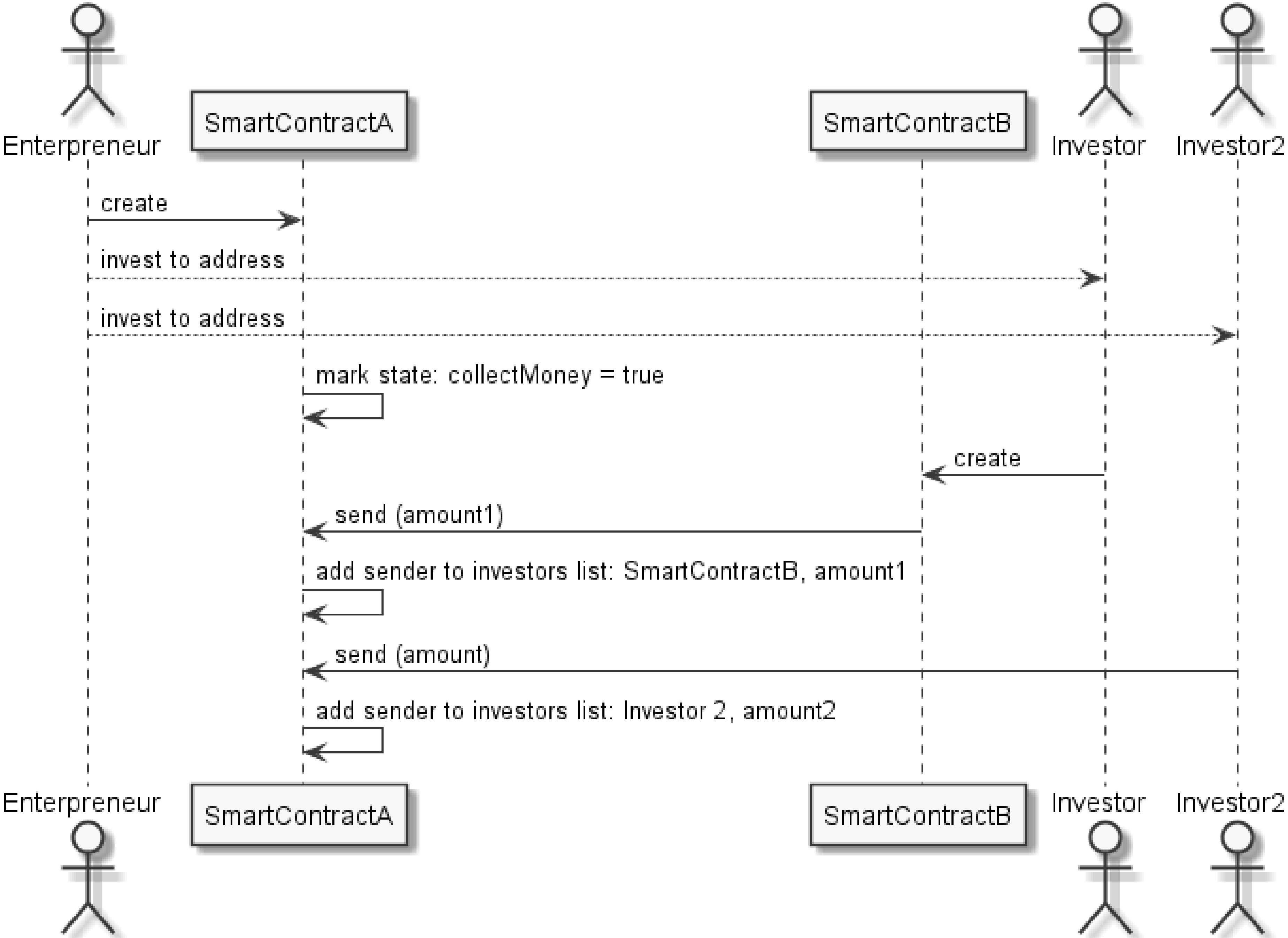
Execute transactions

```
Get ($5) from (Mehmet)
When Event (Pizza boy delivered)
    Give ($4) to PizzaX
    Give ($1) to PizzaBoy

If deliveryTime>requestTime + 30 min
    Return ($4) to Mehmet
    Give ($1) to PizzaBoy

If no delivery in 60 min
    Return ($5) to Mehmet
```

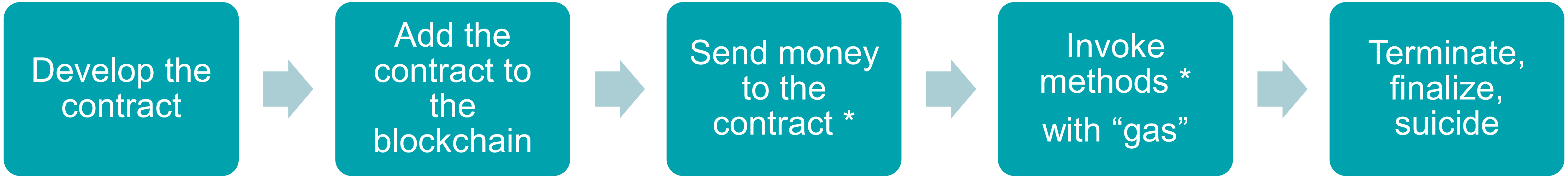
What can be done with smart contracts

- Digital asset transfer on the blockchain
- Coin creation
- Distributed autonomous organizations
- Collective investment and seed funding
- Wallet representation
- Ponzi schemes
- Bidding

Sample (


Enterpreneur
SmartContractA
SmartContractB
Investor
Investor2
create
invest to address
invest to address
mark state: collectMoney = true
create
send (amount1)
add sender to investors list: SmartContractB, amount1
send (amount)
add sender to investors list: Investor 2, amount2
Enterpreneur
SmartContractA
SmartContractB
Investor
Investor2

Environment Ethereum- Solidity



Develop the contract

Add the contract to the blockchain

Send money to the contract *

Invoke methods *
with “gas”

Terminate, finalize, suicide

Smells ..

This study

defines the smells and explains them

classifies the smells

lists types and impact of security issues

explains why it is important to have code analysis of smart contracts

emphasize the difference between classical programming and blockchain contracts.

Why is it important to identify vulnerabilities in smart contracts

- Conflict of interest between who writes and who execute the applications
- Once they are deployed, there is no way for them to be modified.
- No way to fix bugs
- Risks money, mostly about money, resulting in loss of money
- An innocent issue as mistyped variable, can become a vulnerability to trap money in production.
- Hackers can read your code, understand the logic. Find vulnerabilities and exploit them.

If you have an error, they can do what it takes to take advantage of it.

Smart contract related incidents are 22% of all blockchain incidents.

Static analysis flags 45% of the existing contracts as vulnerable.

Impact of vulnerabilities

- Disabled contract
  - A contract that can not function due a bug in code or a resulting contract state
- Locked money
- Reputation

Category 1) Dependence on environment

- Transaction ordering dependence
  - The order of two transactions or interaction to one contract is a factor in outcome

- Timestamp dependence
  - The timestamp is used as an event triggering an outcome

- Using block-hash as random number
  - There is a need for a random number

Category 2) Design and deployment issues

- Gas Limit and Loops
  - Indeterministic loops
- Malicious libraries
  - Calling a library that may have vulnerabilities
- Using inline assembly (next slide)
  - Can not recognize and know what is accomplished in the native code
- Compiler version not fixed
  - Old vs new version of compilers act differently

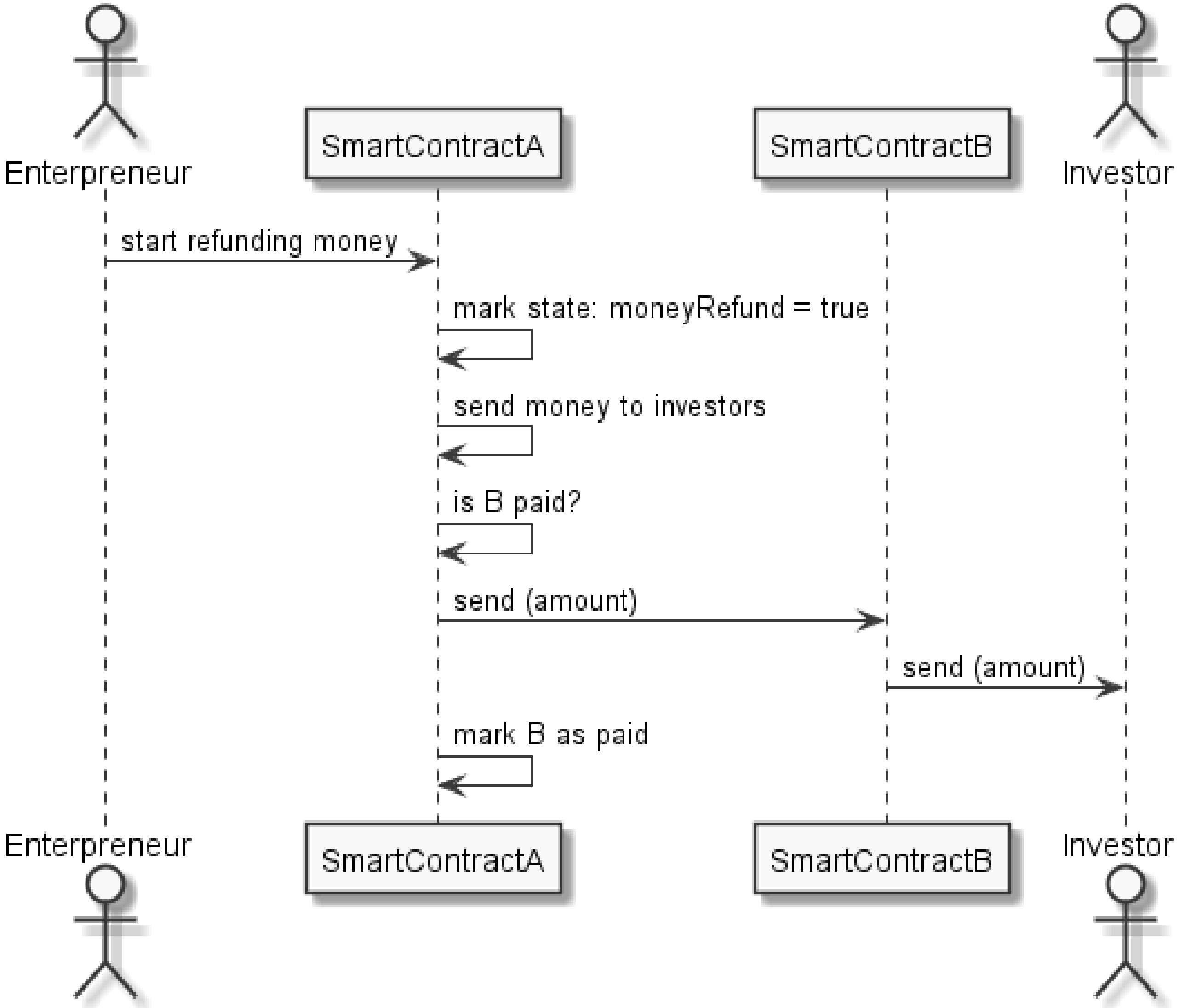
```
pragma solidity ^0.4.19; // bad: 0.4.19 and above
pragma solidity 0.4.19; // good: 0.4.19 only
```

Assembly

```
function rot13Encrypt (string text) public {
    uint256 length = bytes(text).length;
    for (var i = 0; i < length; i++) {
      byte char = bytes(text)[i];
      //inline assembly to modify the string
      assembly {
          char := byte(0,char) // get the first byte
          if and(gt(char,0x6D), lt(char,0x7B)) // if the character is in [n,z], i.e. wrapping.
          { char:= sub(0x60, sub(0x7A,char)) } // subtract from the ascii number a by the difference char is from z.
          if iszero(eq(char, 0x20)) // ignore spaces
          {mstore8(add(add(text,0x20), mul(i,1)), add(char,13))} // add 13 to char.
      }
    }
    emit Result(text);
  }
```

https://github.com/sigp/solidity-security-blog/blob/master/README.md

Category 3) Control of execution and reentrancy



Enterpreneur

SmartContractA

SmartContractB

Investor

start refunding money

mark state: moneyRefund = true

send money to investors

is B paid?

send (amount)

send (amount)

mark B as paid

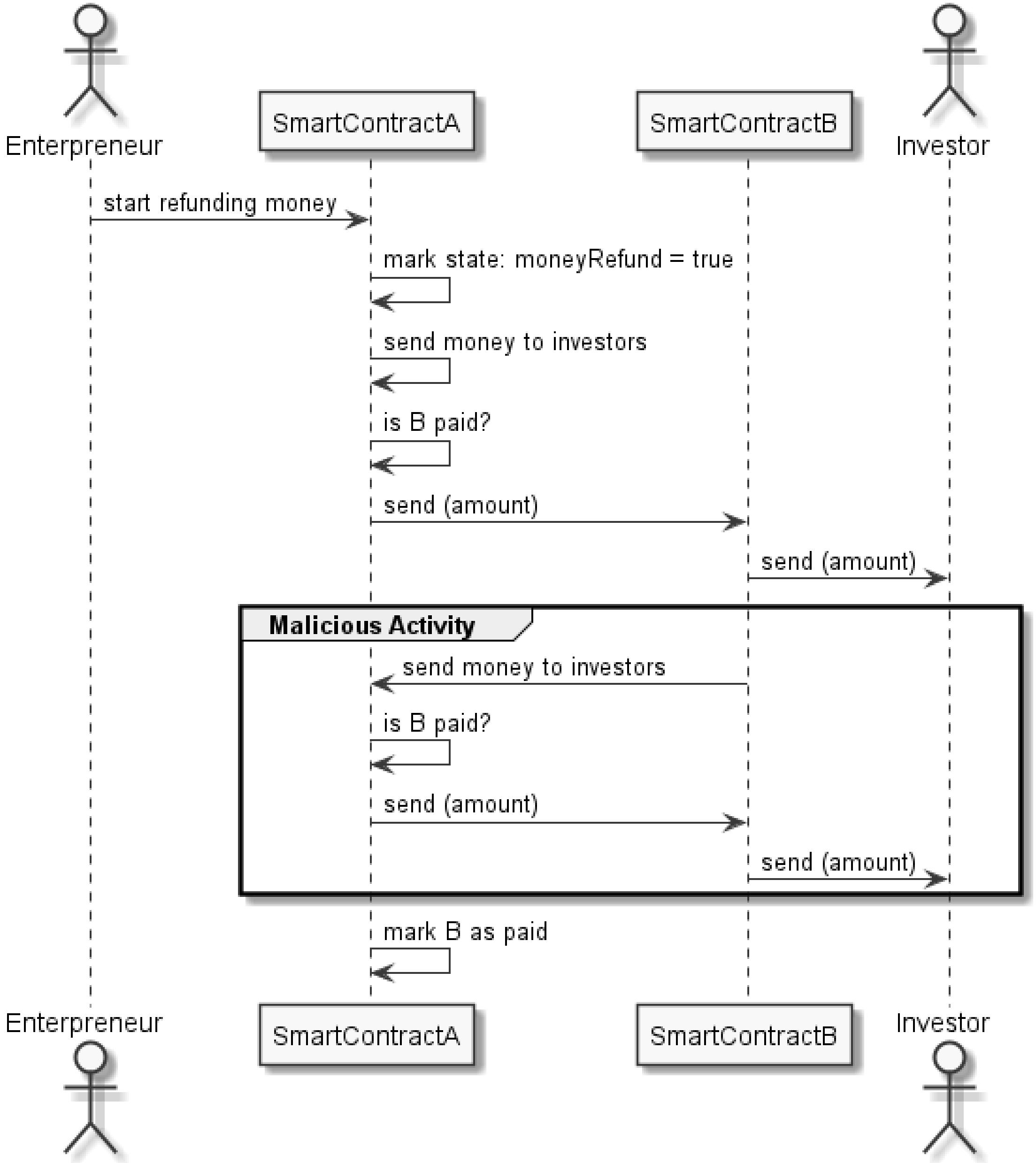
Enterpreneur

SmartContractA

SmartContractB

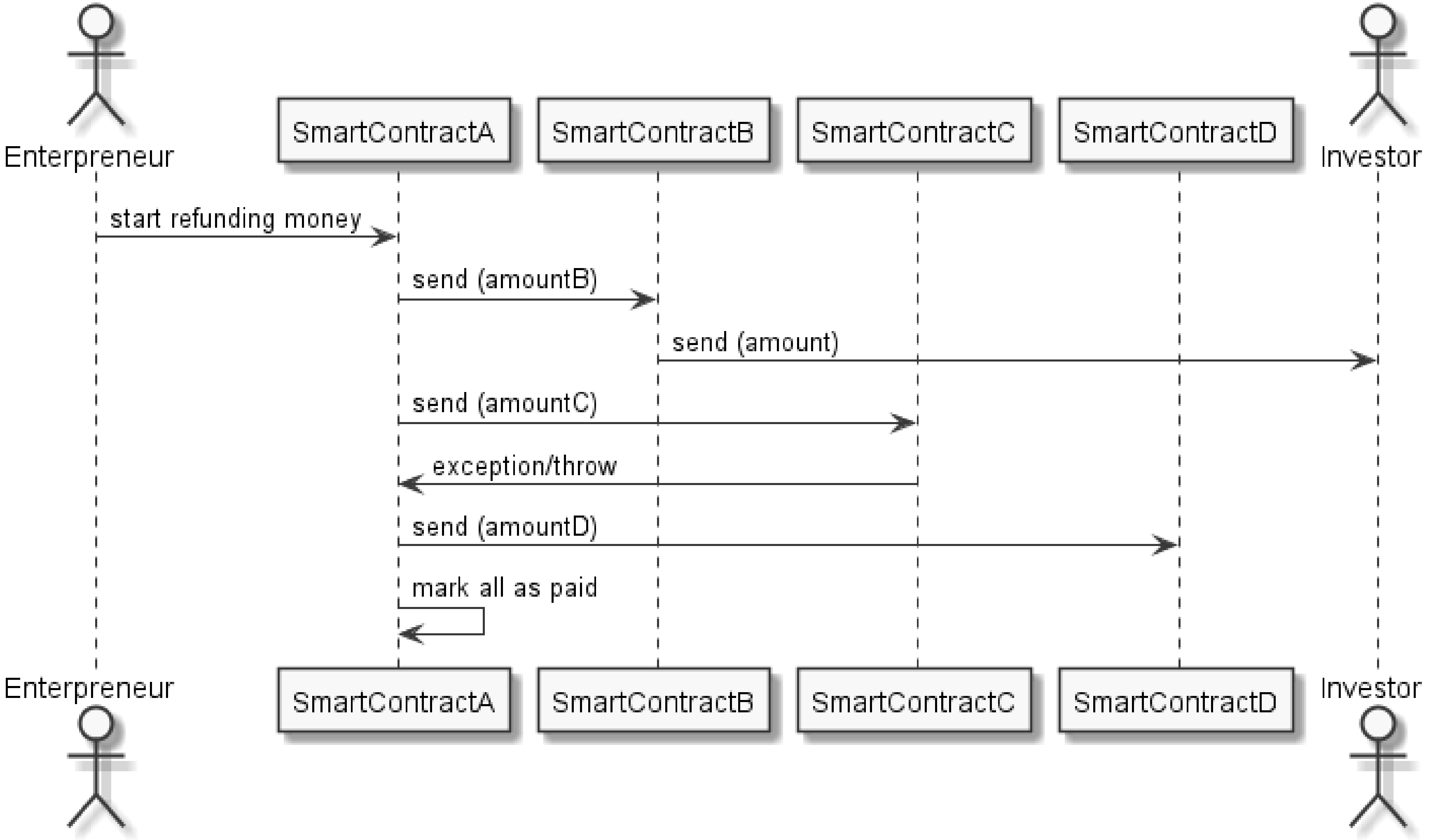
Investor

Reentrancy cont..



Category 4) DoS by misuse of trust

- Exception disorder
- Unexpected throw - DoS
- Unexpected revert - DoS
- DoS with Block Gas Limit
- Unchecked external call



Ryerson University

Category 5) Unsafe external interaction

- Use of tx.origin

  vs msg.sender

- Send instead of transfer

  addr.send(42 ether); // bad

  if (!addr.send(42 ether)) revert; // better

  addr.transfer(42 ether); // good

- Gasless send

  Not enough gas left to send money

- Using Self destruct

  Termination, cleanup, other contract’s dependency

- Using throw, revert, assert, require (next slide)

http://orbilu.uni.lu/bitstream/10993/35862/1/smartcheck-paper.pdf

Return, throw, revert, assert, require

```
contract HasAnOwner {
address owner;

function useSuperPowers(){
if (msg.sender != owner) { throw; }
// do something only the owner should be allowed to do
}
}
```

```
if(msg.sender != owner) { revert(); } // new, like throw, forced rollback
assert(msg.sender == owner); // forced rollback, assertive message
require(msg.sender == owner); // forced rollback, highly readable, kind message
```

https://medium.com/blockchannel/the-use-of-revert-assert-and-require-in-solidity-and-the-new-revert-opcode-in-the-evm-1a3a7990e06e

Category 6) Vulnerable coding practices

- Balance inequality

  if (this.balance == 42 ether) // bad

  if (this.balance >= 42 ether) // good
- Redundant fallback function
- Typographical error

  += vs =+
- Unchecked math

  Overflow, Underflow, Integer division
- Unsafe type inference

  for (**var** i = 0; i < array.length; i++) // uint8 max=256

  for (**uint256** i = 0; i < array.length; i++)
- Implicit visibility level

  Nothing is private
- Address hardcoding and sending
- Array length manipulation

  anArray.length--; // .. Underflow risk
- A setter method that transfer power to the caller

  .setOwner(address)

http://orbilu.uni.lu/bitstream/10993/35862/1/smartcheck-paper.pdf

Questions?

Contact: Mehmet.Demir@Ryerson.ca

-SmartCheck : Static Analysis of Ethereum Smart Contracts http://orbilu.uni.lu/bitstream/10993/35862/1/smartcheck-paper.pdf

-Securify: Practical Security Analysis of Smart Contracts https://arxiv.org/pdf/1806.01143.pdf

-Rethinking Blockchain Security: Position Paper https://arxiv.org/abs/1806.04358

-ContractFuzzer:FuzzingSmartContracts forVulnerabilityDetection https://arxiv.org/ftp/arxiv/papers/1807/1807.03932.pdf

-Making Smart Contracts Smarter – Oyente platform https://eprint.iacr.org/2016/633.pdf

-Smart Contracts Vulnerabilities:A Call for Blockchain Software Engineering https://ieeexplore.ieee.org/document/8327567

-Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains https://arxiv.org/abs/1809.02702

-Mythril Platform whitepaper https://mythril.ai/files/whitepaper.pdf

A total of 30 resources..