Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks

Michael Rodler [1], Wenting Li [2], Ghassan O. Karame [2], Lucas Davi [1]

[1] University of Duisburg-Essen
[2] NEC Laboratories Europe

26th Network and Distributed System Security Symposium (NDSS19)

The DAO Hack

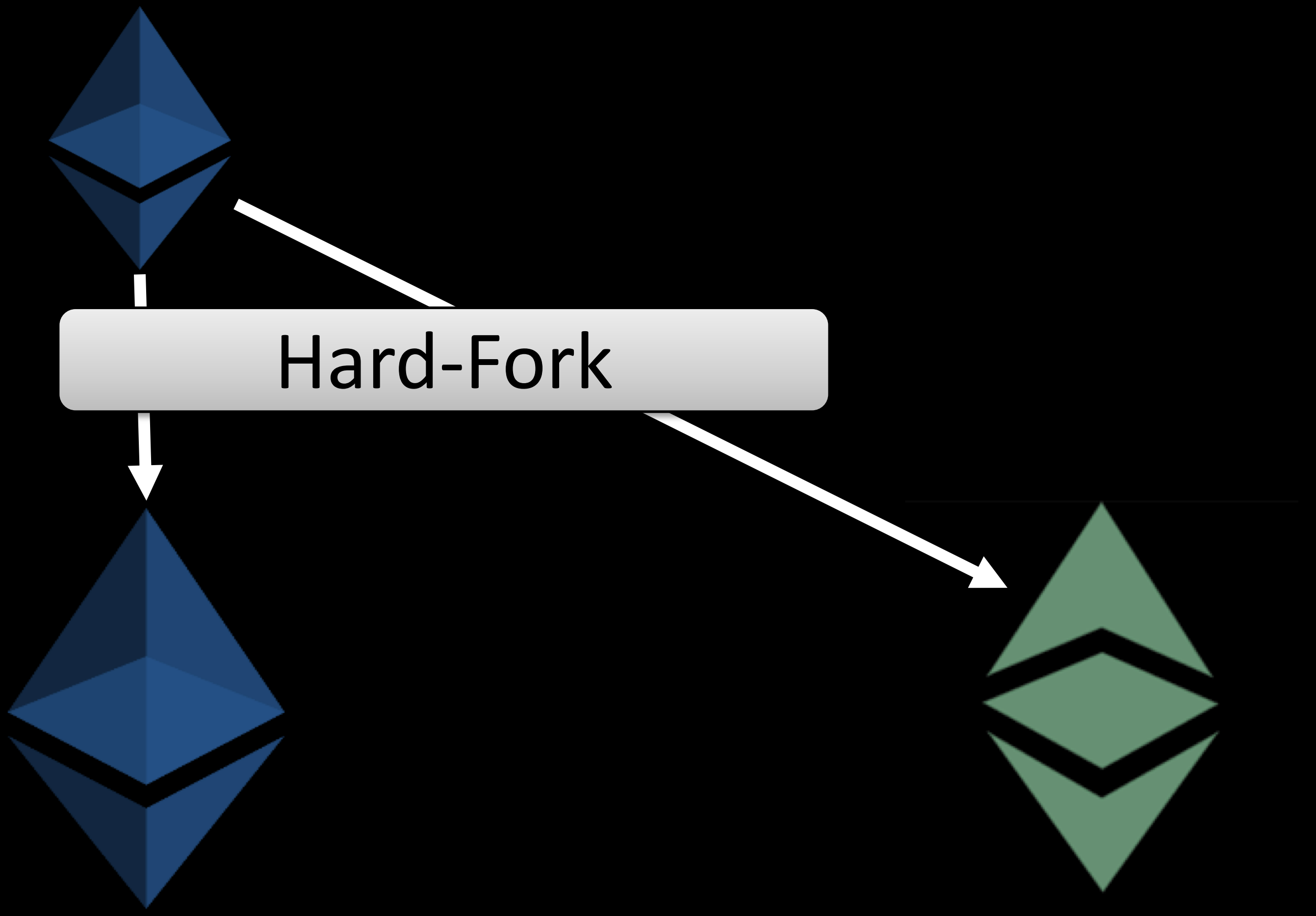
17 June 2016

3.6 Million Ether Stolen

worth $50 Million

5% of all available Ether

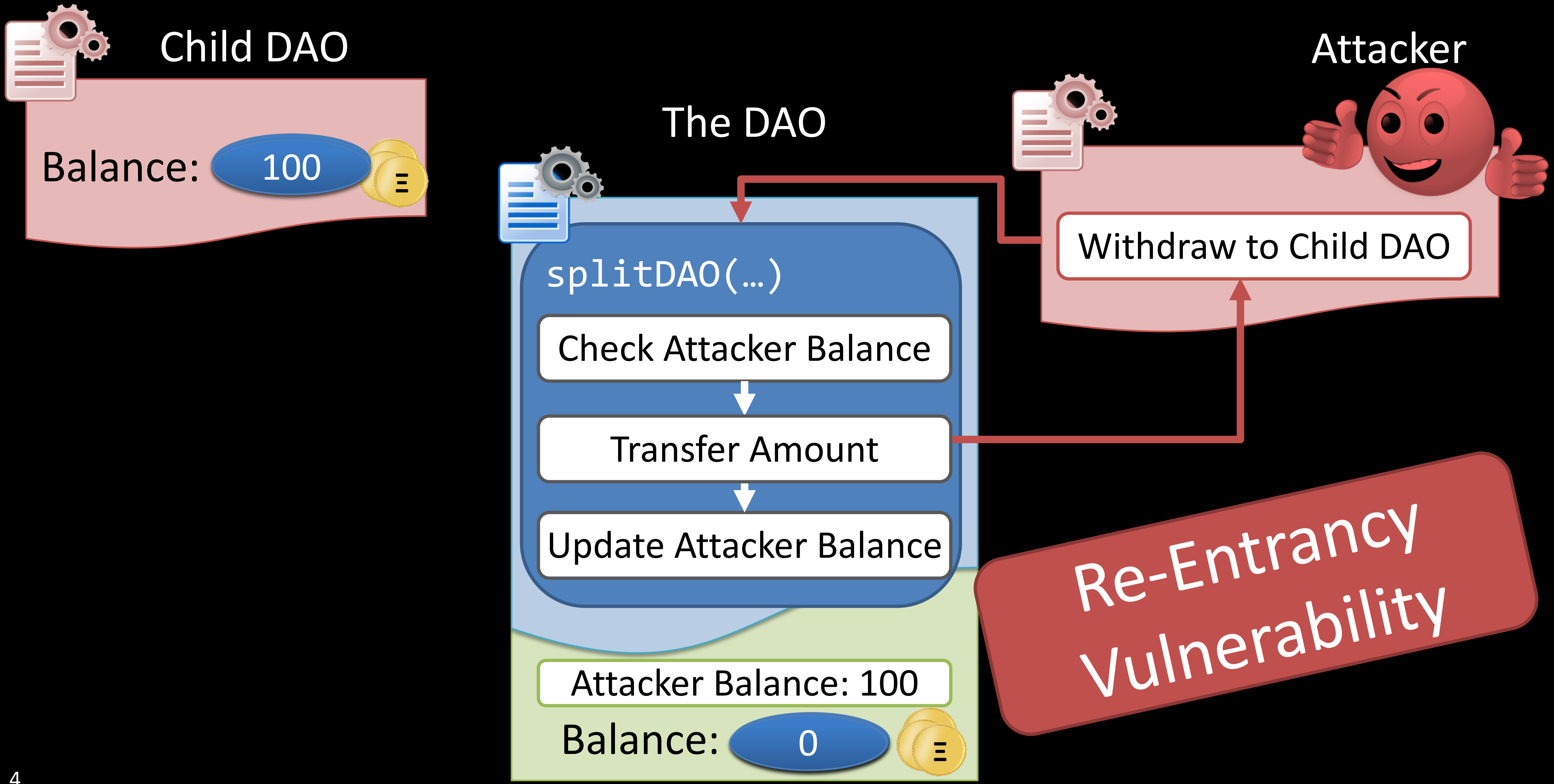
The DAO Aftermath



Ethereum ETH

Ethereum Classic ETC

The DAO Attack



Can we automatically detect re-entrancy vulnerabilities?

Prior Research on Bug Finding and Exploitation in Smart Contracts

Symbolic execution

- **Oyente** [Luu et al., CCS16]
- **TeEther** [Krupp+Rossow, USENIX SEC 18]
- **Manticore** (Trail of Bits)
- **MAIAN** [Nikolic et al., ACSAC18]
- **Mythril** (ConsenSys)
- **OSIRIS** [Torres et al., ACSAC18]

Static analysis

- **Securify** [Tsankov et al., CCS18]
- **SmartCheck** [Tikhomirov et al., CCS18]

Runtime Checking

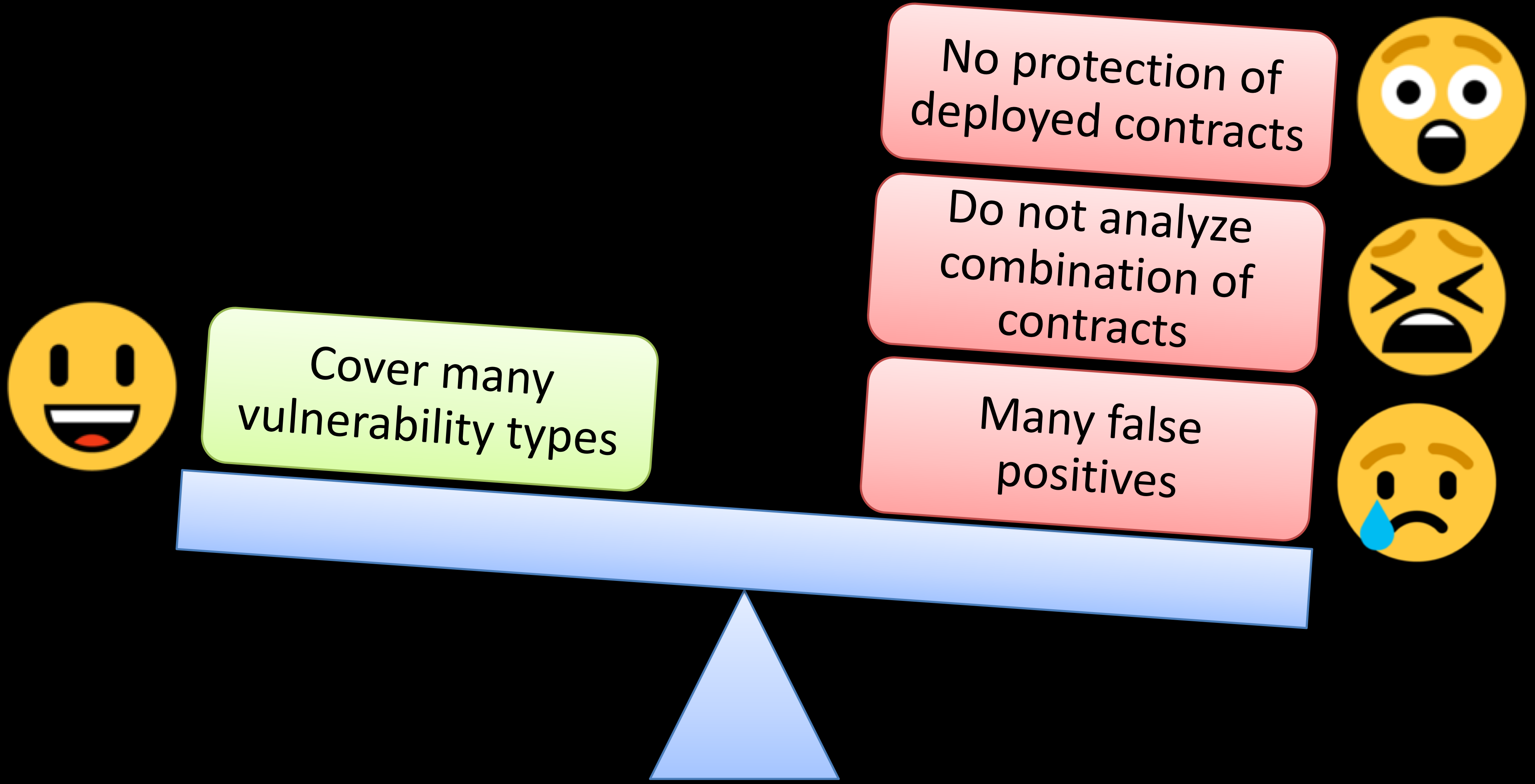
- **ECFChecker** [Grossman et al., POPL18]

Verification

- **ZEUS** [Kalra et al., NDSS18]

Detects Re-Entrancy

Current Bug Finding Tools



Our Research Questions:

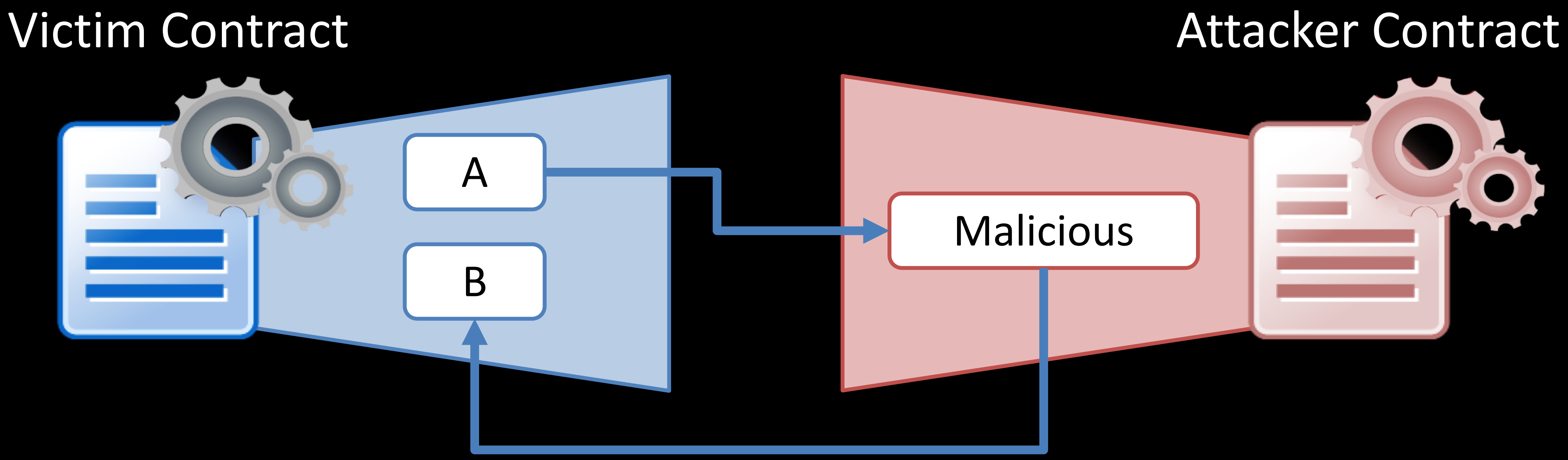
1. **Do existing tools cover all re-entrancy bugs?**
2. **Can we protect deployed contracts?**

Our Contributions

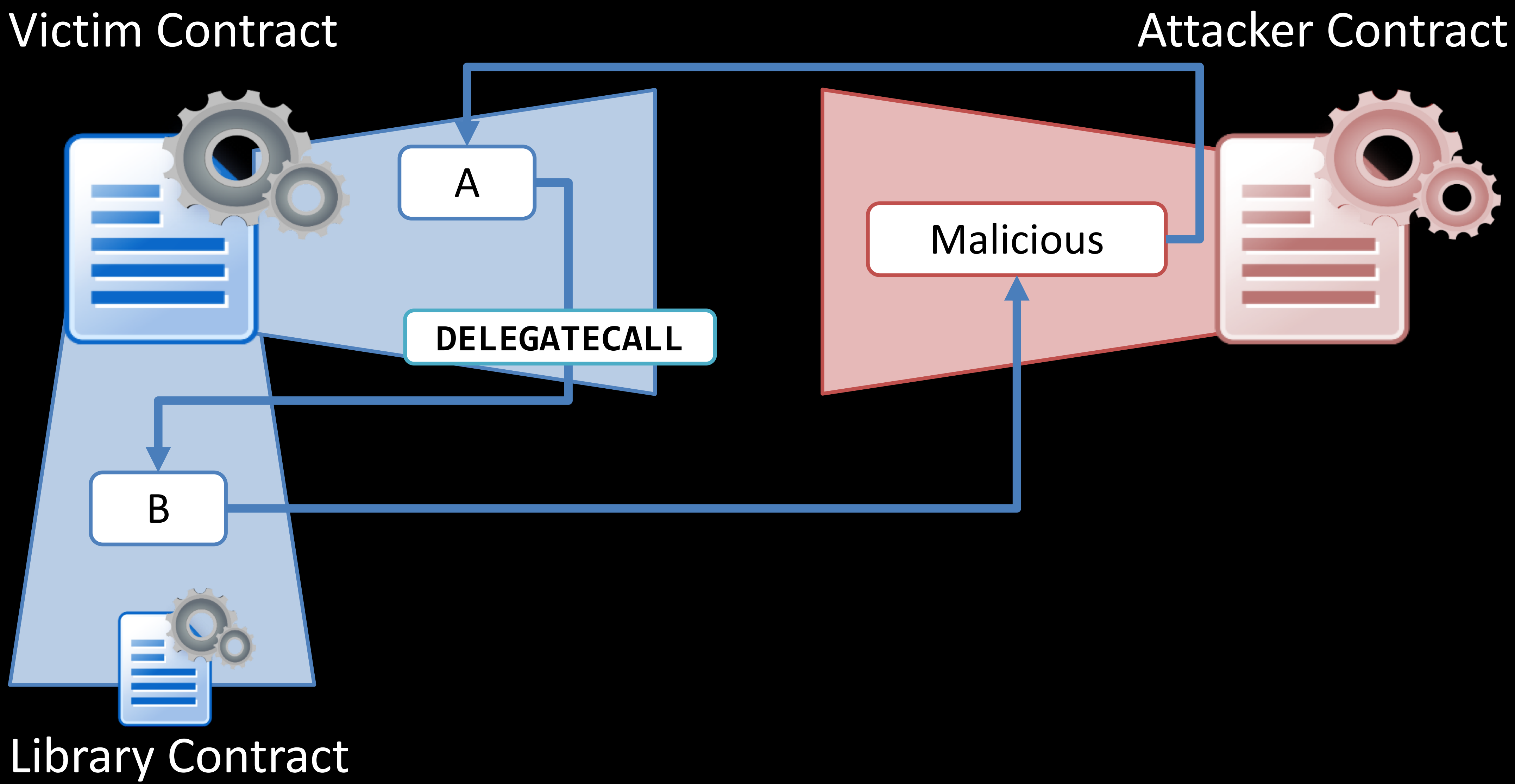
- Overlooked re-entrancy attack patterns
- Sereum – Hardened Ethereum Client
  - Taint tracking engine for EVM bytecode
  - Runtime detection of re-entrancy attacks
- Investigation of root causes for false positives

Overlooked re-entrancy problems

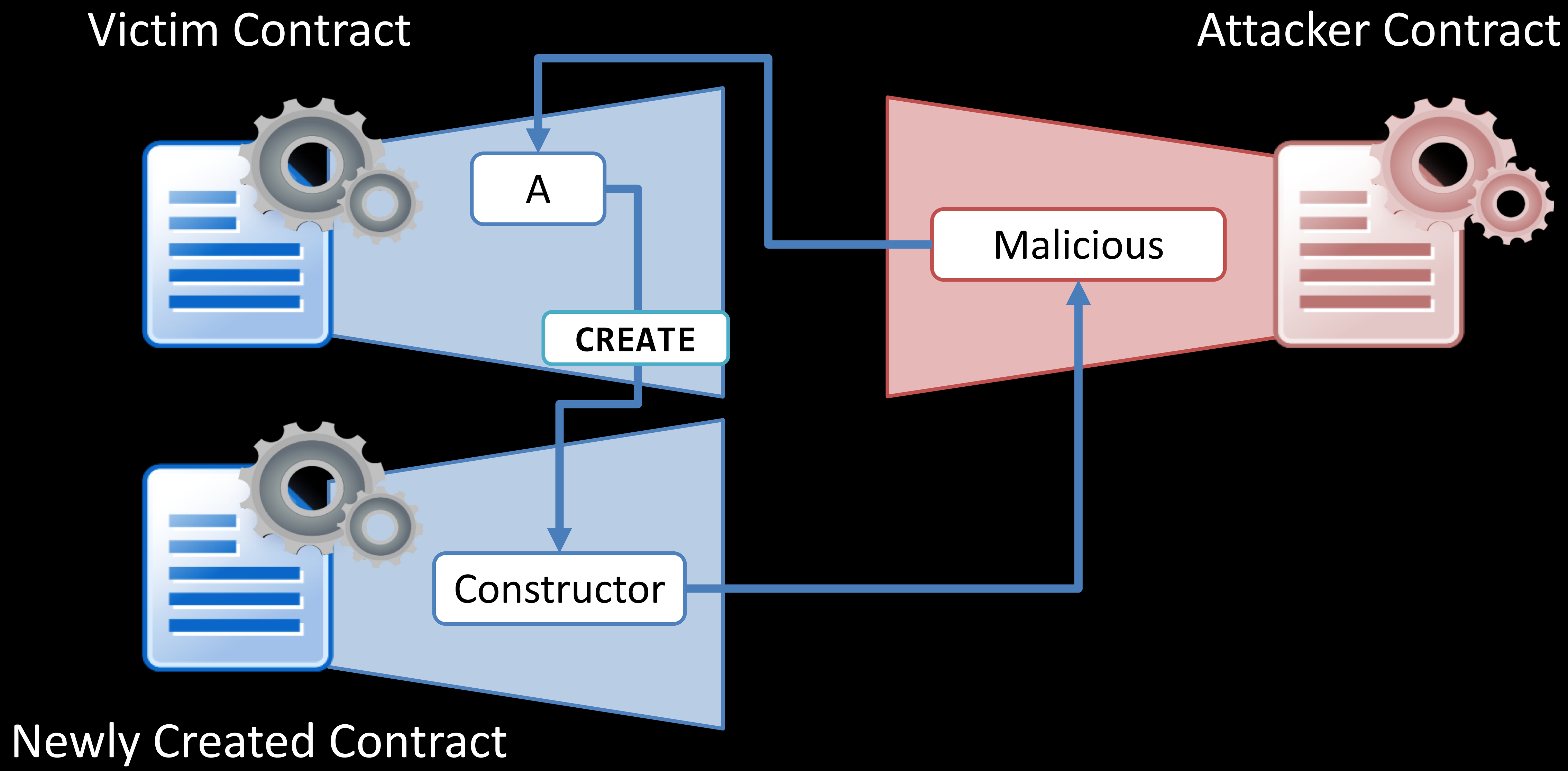
Attack 1: Cross-Function Re-Entrancy



Attack 2: Delegated Re-Entrancy



Attack 3: Create-Based Re-Entrancy



Overview on Re-Entrancy Detection

Tool	Same-Function	Cross-Function	Delegated	Create-based
Oyente [Luu et al., CCS16]	✓			
Securify [Tsankov et al., CCS18]	✓ *	✓ *		
ECFChecker [Grossman et al., POPL18]	✓	✓	✓	
Manticore (Trail of Bits)	✓	✓		
Mythril (ConsenSys)	✓ *	✓ *		
Sereum	✓	✓	✓	✓

* Conservative policy with high number of false positives

Main Observation

Typically re-entrancy attacks exploit **inconsistent state**

at the time the vulnerable contract **decides whether to take a branch**

Sereum Approach

```
function withdraw(uint amount)
  if (balance[msg.sender] >= amount)
    msg.sender.call.value(amount)("");
    balance[msg.sender] -= amount;
  return;
```

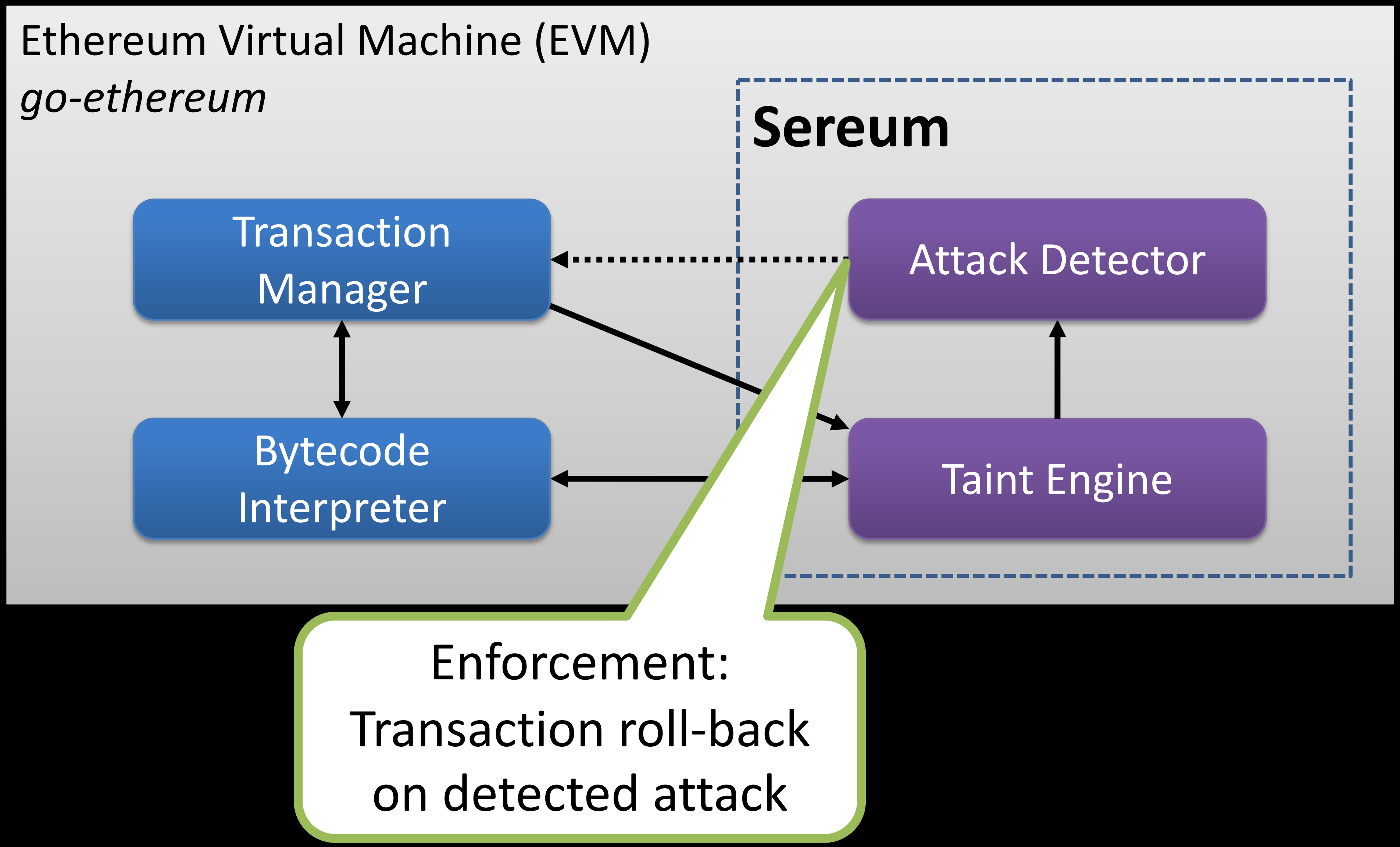
F

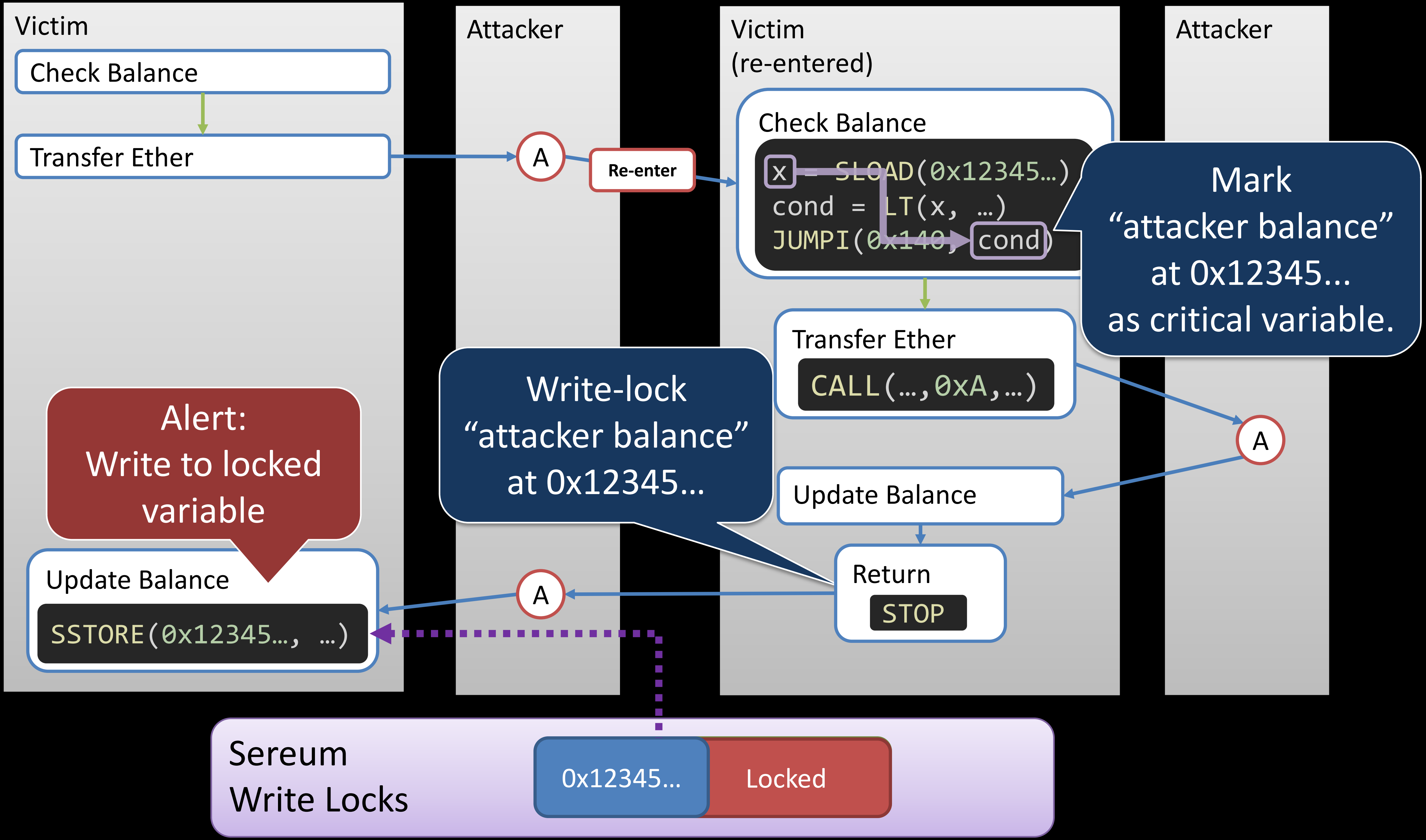
T

Mark variables that influence **branching decisions** as critical

Prevent further updates with write-locks

Sereum Architecture





Victim
Check Balance
Transfer Ether
Attacker
A
Re-enter
Victim
(re-entered)
Check Balance
x = SLOAD(0x12345…)
cond = LT(x, …)
JUMPI(0x140, cond)
Mark
“attacker balance”
at 0x12345...
as critical variable.
Attacker
Transfer Ether
CALL(…,0xA,…)
A
Write-lock
“attacker balance”
at 0x12345...
Alert:
Write to locked
variable
Update Balance
Return
STOP
Update Balance
SSTORE(0x12345…, …)
A
Sereum
Write Locks
0x12345...
Locked

Evaluation Results

Evaluation on first
4.5 Million Ethereum blocks

Successful detection of The DAO incident

~50k flagged transactions

Manual reverse-engineering and analysis of flagged transactions

~2k true attack transactions

FP rate: 0.06%

14 distinct contracts result in false positive

New Finding:
The curios case of DSEthToken

Developers hacked their own contract

7 days before The DAO incident

Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks

Michael Rodler [1], Wenting Li [2], Ghassan O. Karame [2], Lucas Davi [1]

[1] University Duisburg-Essen
[2] NEC Laboratories Germany

Questions?

github.com/uni-due-syssec/eth-reentrancy-attack-patterns

@f0rki

Backup Slides

Sereum Performance

- Benchmark: Execute 50 Blocks in Batch (10 000 repetitions)
  - Sereum – mean 2494.5 ms (σ = 174.8 ms)
  - Geth – mean 2277.0 ms (σ = 146.7 ms)
  - Mean overhead: 9.6 %
  - Average memory consumption: geth 9252MB, Sereum 9767MB

- Timings on newer blocks (around block ~6 700 000)
  - Average 5 sec to process block with Sereum (about 150 TX)
  - New block every ~15 sec
  - Sereum can keep up with network!

Evaluation of Sereum

1. We verified that Sereum successfully detects the new attack patterns
2. Evaluation on the Ethereum blockchain
   - We **re-executed all blocks** up until block number **4 500 000** **(77 987 922 transactions)**
   - We detected attacks related to “the DAO”
   - Sereum flagged **49 080 transactions** as **re-entrancy attacks**
3. We manually reverse-engineered and analyzed detected contracts/attacks
   - We identified 2 337 true attack transactions
   - Sereum has an overall **false positive rate as low as 0.06%**
   - We identified 5 major classes of root-causes of false positives (see details in the paper)

False Positive Causes

I. Lack of field-sensitivity on the EVM level
   - Small types packed densely into one storage address

II. Storage Deallocation
   - Deallocation: overwrite with zero

III. Constructor Callbacks
   - Instead of passing data as argument, retrieved

IV. Tight Contract Coupling
   - Contract execution passes between two or more contracts

V. Manual Re-Entrancy Locking
   - Manual locking is identical to malicious re-entrancy pattern

Sereum Usage

- Detection mode
  - Developer continuously runs Sereum
  - Re-play all public Ethereum transactions, looking for attacks
  - Developer reacts to attacks
- Enforcement mode
  - Integrate Sereum into all Ethereum clients
  - For example: private blockchain based on Ethereum

References

- L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making Smart Contracts Smarter”, ACM CSS 2016
- P. Tsankov, A. Dan, D. D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, “Securify: Practical Security Analysis of Smart Contracts”, ACM CCS 2018
- S. Kalra, S. Goel, M. Dhawan, and S. Sharma, “ZEUS: Analyzing Safety of Smart Contracts”, NDSS 2018
- J. Krupp and C. Rossow, “TeEther: Gnawing at Ethereum to Automatically Exploit Smart Contracts,” USENIX Security 2018
- I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, “Finding The Greedy, Prodigal, and Suicidal Contracts at Scale”, ACSAC 2018
- S. Grossman et al., “Online Detection of Effectively Callback Free Objects with Applications to Smart Contracts”, POPL 2018.
- S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, “SmartCheck: Static Analysis of Ethereum Smart Contracts,” 2018.