Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks

Michael Rodler¹, Wenting Li², Ghassan O. Karame², Lucas Davi¹

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

Hard-Fork

Ethereum ETH

Ethereum Classic ETC
The DAO Attack

The DAO

splitDAO(...)

Check Attacker Balance

Transfer Amount

Update Attacker Balance

Attacker Balance: 100

Balance: 0

Withdraw to Child DAO

Re-Entrancy Vulnerability

Child DAO

Balance: 100

Attacker

Transfer Amount

Withdraw to Child DAO

Attacker Balance: 100

Balance: 0
Can we automatically detect re-entrancy vulnerabilities?
Prior Research on Bug Finding and Exploitation in Smart Contracts

**Symbolic execution**
- **Oyente** [Luu et al., CCS16]
- **Manticore** (Trail of Bits)
- **Mythril** (ConsenSys)

**Runtime Checking**
- **ECFChecker** [Grossman et al., POPL18]
- **MAIAN** [Nikolic et al., ACSAC18]
- **OSIRIS** [Torres et al., ACSAC18]

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

**Verification**
- **ZEUS** [Kalra et al., NDSS18]
Current Bug Finding Tools

- Cover many vulnerability types
- No protection of deployed contracts
- Do not analyze combination of contracts
- Many false positives
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

Victim Contract

Attacker Contract

A

B

Malicious
Attack 2: Delegated Re-Entrancy

Victim Contract

Library Contract

Attacker Contract

DELEGATECALL

A

B

Malicious
Attack 3: Create-Based Re-Entrancy

Victim Contract

Attacker Contract

Newly Created Contract

Malicious
# Overview on Re-Entrancy Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Same-Function</th>
<th>Cross-Function</th>
<th>Delegated</th>
<th>Create-based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oyente</strong></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Luu et al., CCS16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Securify</strong></td>
<td>✓</td>
<td>✓ *</td>
<td>✓ *</td>
<td></td>
</tr>
<tr>
<td>[Tsankov et al., CCS18]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ECFChecker</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[Grossman et al., POPL18]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Manticore</strong></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>(Trail of Bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mythril</strong></td>
<td>✓</td>
<td>✓ *</td>
<td></td>
<td>✓ *</td>
</tr>
<tr>
<td>(ConsenSys)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sereum</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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

Mark variables that influence branching decisions as critical

Prevent further updates with write-locks
Sereum Architecture

Ethereum Virtual Machine (EVM)

*go-ethereum*

---

**Sereum**

- **Transaction Manager**
- **Bytecode Interpreter**
- **Attack Detector**
- **Taint Engine**

**Enforcement:**
Transaction roll-back on detected attack
Alert: Write to locked variable

Update Balance

SSTORE(0x12345..., ...)

Write-lock “attacker balance” at 0x12345...

Mark “attacker balance” at 0x12345... as critical variable.

Update Balance

SSTORE(0x12345..., ...)

Alert: Write to locked variable

Sereum Write Locks

0x12345... Locked
Evaluation Results

Evaluation on first 4.5 Million Ethereum blocks

- Successful detection of The DAO incident
- Manual reverse-engineering and analysis of flagged transactions
- ~50k flagged transactions
- ~2k true attack transactions
- 14 distinct contracts result in false positive
- FP rate: 0.06%

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
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

- J. Krupp and C. Rossow, “TeEther: Gnawing at Ethereum to Automatically Exploit Smart Contracts,” USENIX Security 2018