SMARTSHIELD: Automatic Smart Contract Protection Made Easy

Yuyao Zhang\textsuperscript{1}, Siqi Ma\textsuperscript{2}, Juanru Li\textsuperscript{1}, Kailai Li\textsuperscript{1}, Surya Nepal\textsuperscript{2}, Dawu Gu\textsuperscript{1}

\textsuperscript{1}Shanghai Jiao Tong University, Shanghai, China
\textsuperscript{2}Data61, CSIRO, Sydney, Australia
1. Background
2. Motivation
3. Automated Rectification with SMARTSHIELD
4. Evaluation
5. Conclusion
Blockchain

- A decentralized and distributed system.
- Secured using cryptography.
- Trust arises from the majority of peers, not an authority.

**Blockchain 1.0:**
- Cryptocurrency (*Bitcoin*)

**Blockchain 2.0:**
- Smart Contract (*Ethereum*)
Ethereum Smart Contract

- Programs that permanently exist and automatically run on the blockchain.
- Enabling the encoding of complex logic:
  - Payoff schedule
  - Investment assumptions
  - Interest policy
  - ……
Ethereum Smart Contract

- Written in high-level languages (e.g., Solidity).
- Compiled to low-level bytecode.

```
1 mapping(address => uint) public balances;
2 ...
3 function send(address receiver, uint amount) public {
4     require(amount <= balances[msg.sender]);
5     balances[msg.sender] -= amount;
6     balances[receiver] += amount;
7 }
```
Outline

1. Background

2. Motivation

3. Automated Rectification with SMARTSHIELD

4. Evaluation

5. Conclusion
Attacks on Smart Contracts

Wallet bug freezes more than $150 million worth of Ethereum

Ethereum's Parity Users Lose Millions in a Multi-Sig Hack

On July 19 the ethereum community was warned that the Parity client version 1.5 and above contained a critical vulnerability in the multi-signature wallet feature. Further, a group of multi-signature “black hat exploiters” has managed to drain 150,000 ether from multi-sig wallets and ICO projects.

SpamChain Loses $40K in Hack Due to Smart Contract Bug

25% of All Smart Contracts Contain Critical Bugs

For every problem that smart contracts solve, they seem to introduce another. In a week in which EOS has made news for all the wrong reasons over a RAM vulnerability, a code auditor has revealed the prevalence of smart contract bugs. Security firm Hasho, which has forged a new partnership with community managers Amaize, has found that one in four projects contains critical vulnerabilities.
Motivation

Key Insights

• A smart contract can never be updated after its deployment to the blockchain.
• Existing tools only locate smart contract bugs instead of helping developers fix the buggy code.
• A large portion of smart contract bugs share common code patterns, indicating that they can be fixed through a unified approach.
Insecure Code Patterns in Smart Contracts

- **Code Pattern 1: State Changes after External Calls.**
  - A state variable is updated after an external function call.
  - May result in a **re-entrancy bug**.

```
mapping (address => uint) public userBalances;
...
function withdrawBalance(uint amountToWithdraw) public {
    require(userBalances[msg.sender] >= amountToWithdraw);
    userBalances[msg.sender] -= amountToWithdraw;
    msg.sender.call.value(amountToWithdraw)();
    userBalances[msg.sender] -= amountToWithdraw;
}
```
Insecure Code Patterns in Smart Contracts

  - An arithmetic operation is executed without checking the data validity in advance.
  - May cause an arithmetic bug.

```solidity
uint public lockTime = now + 1 weeks;
address public user;
...
function increaseLockTime(uint timeToIncrease) public {
    require(msg.sender == user);
    require(lockTime + timeToIncrease >= lockTime);
    lockTime += timeToIncrease;
}
...
function withdrawFunds() public {
    require(now > lockTime);
    user.transfer(address(this).balance);
```
Insecure Code Patterns in Smart Contracts

- Code Pattern 3: Missing Checks for Failing External Calls.
  - The return value is not being checked after an external function call.
  - May cause an **unchecked return value bug**.

```solidity
bool public payedOut = false;
address public winner;
uint public bonus;
...
function sendToWinner() public {
    require(!payedOut && msg.sender == winner);
    - msg.sender.send(bonus);
+   require(msg.sender.send(bonus));
    payedOut = true;
}
```
Our Approach

- Automatically fix insecure cases with typical patterns in smart contracts before their deployments.

**Challenges & Solutions:**

- Compatibility → Bytecode-Level Program Analysis.
- Economy → Gas Optimization.
Outline

1. Background
2. Motivation
3. Automated Rectification with SMARTSHIELD
4. Evaluation
5. Conclusion
Automated Rectification with SMARTSHIELD
High-Level Workflow of SMARTSHIELD

- Take a **smart contract** as input.
- Output a **secure EVM bytecode** without any of the three insecure code patterns:
  - *State changes after external calls.*
  - *Missing checks for out-of-bound arithmetic operations.*
  - *Missing checks for failing external calls.*
- Generate a **rectification report** to the developer.
Semantic Extraction

- **Bytecode-Level Semantic Information:**
  - Control and data dependencies among instructions in EVM bytecode.
  - Necessary for further code transformation and secure bytecode generation.

- Extract bytecode-level semantic information from:
  - **Abstract Syntax Tree (AST):** Control- and data-flow analysis.
  - **Unrectified EVM Bytecode:** Abstractly emulate the execution of the contract bytecode.
Contract Rectification

- **Strategy 1: Control Flow Transformation.**
  - Revise *state changes after external calls*.

- Adjust the original control flow by moving state change operations to the front of external calls.

- Preserve the original dependencies among instructions in EVM bytecode.
Contract Rectification

- **Strategy 2: DataGuard Insertion.**
  - Fix *missing checks for out-of-bound arithmetic operations*, and *missing checks for failing external calls*.

- **Dataguard:**
  - Sequences of instructions that perform certain data validity checks.

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Operation</th>
<th>DataGuard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic ops</td>
<td>ADD</td>
<td>$a + b$</td>
<td>$a + b \geq a$</td>
</tr>
<tr>
<td></td>
<td>SUB</td>
<td>$a - b$</td>
<td>$a \geq b$</td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td>$a \times b$</td>
<td>$a \times b \div a = b$</td>
</tr>
<tr>
<td>External calls</td>
<td>CALL</td>
<td>$ret = a.call()$</td>
<td>$ret \neq 0$</td>
</tr>
</tbody>
</table>

```
0000: 6004  PUSH1 0x04
0002: 35    CALLDATALOAD
0003: 6193A8 PUSH2 0x93A8
0006: + 01 ADD
0007: + 61000E PUSH2 0x000E
000A: + 61008A PUSH2 0x008A
000D: + 56 JUMP
000E: + 5B JUMPDEST
008A: + 5B JUMPDEST
...  ...
009A: + 56 JUMP
```

*Safe Function for Addition*
Rectified Contract Generation

- **Bytecode Relocation:**
  - Update all unaligned target addresses of jump instructions.

- **Bytecode Validation:**
  - Validate whether the other irrelevant functionalities are affected.

- **Rectification Report:**
  - Record the concrete modifications for further manual verification or adjustments.
Outline

1. Background
2. Motivation
3. Automated Rectification with SMARTSHIELD
4. Evaluation
5. Conclusion
Research Questions

▪ RQ1: Scalability.
  ▪ How scalable is SMARTSHIELD in rectifying real-world smart contracts?

▪ RQ2: Correctness.
  ▪ How effective and accurate is SMARTSHIELD in fixing insecure cases with typical patterns and assuring the functionality consistency between the rectified and the original contracts?

▪ RQ3: Cost.
  ▪ What is the additional cost of the rectified contract?
Dataset

- A snapshot of the first 7,000,000 blocks in the *Ethereum Mainnet* (ETH).
- 2,214,409 real-world smart contracts.
- Label insecure cases with the help of state-of-the-art smart contract analysis tools.
- 95,502 insecure cases in 28,621 contracts.

<table>
<thead>
<tr>
<th>Category</th>
<th># of insecure cases</th>
<th># of insecure contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP.1</td>
<td>4,521</td>
<td>726</td>
</tr>
<tr>
<td>CP.2</td>
<td>80,825</td>
<td>25,470</td>
</tr>
<tr>
<td>CP.3</td>
<td>10,156</td>
<td>4,811</td>
</tr>
<tr>
<td>Total</td>
<td>95,502</td>
<td>28,621*</td>
</tr>
</tbody>
</table>

* Some contracts contain multiple insecure patterns.

CP.1: State Changes after External Calls
CP.2: Missing Checks for Out-of-Bound Arithmetic Ops
CP.3: Missing Checks for Failing External Calls
RQ1: Scalability

- **87,346 (91.5%)** insecure cases were fixed.
- **25,060 (87.6%)** insecure contracts were fully rectified.

<table>
<thead>
<tr>
<th>Category</th>
<th># of eliminated cases</th>
<th># of uneliminable cases</th>
<th># of rectified contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP.1</td>
<td>3,567</td>
<td>954</td>
<td>573</td>
</tr>
<tr>
<td>CP.2</td>
<td>74,642</td>
<td>6,183</td>
<td>21,815</td>
</tr>
<tr>
<td>CP.3</td>
<td>9,137</td>
<td>1,019</td>
<td>4,362</td>
</tr>
<tr>
<td>Total</td>
<td>87,346</td>
<td>8,156</td>
<td>25,060*</td>
</tr>
</tbody>
</table>

* Some contracts contain multiple insecure patterns.

- The remaining insecure cases were marked as “unrectifiable” due to a conservative policy.
RQ2: Correctness

- Part 1: Evaluate whether SMARTSHIELD actually fixed the insecure code in contracts.
  - Leverage prevalent analysis techniques to examine each rectified contract.
  - Replay exploits of existing high-profile attacks against rectified contracts.

<table>
<thead>
<tr>
<th>Insecure contract</th>
<th>Category</th>
<th>Date of attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAO* [35], [36]</td>
<td>CP.1</td>
<td>Jun. 17\textsuperscript{th}, 2016 [25]</td>
</tr>
<tr>
<td>LedgerChannel [37]</td>
<td>CP.1</td>
<td>Oct. 7\textsuperscript{th}, 2018 [38]</td>
</tr>
<tr>
<td>BeautyChain [39]</td>
<td>CP.2</td>
<td>Apr. 22\textsuperscript{nd}, 2018 [26]</td>
</tr>
<tr>
<td>SmartMesh [40]</td>
<td>CP.2</td>
<td>Apr. 24\textsuperscript{th}, 2018 [41]</td>
</tr>
<tr>
<td>UselessEthereumToken [42]</td>
<td>CP.2</td>
<td>Apr. 27\textsuperscript{th}, 2018 [43]</td>
</tr>
<tr>
<td>Social Chain [44]</td>
<td>CP.2</td>
<td>May 3\textsuperscript{rd}, 2018 [45]</td>
</tr>
<tr>
<td>Hexagon [46]</td>
<td>CP.2</td>
<td>May. 18\textsuperscript{th}, 2018 [47]</td>
</tr>
<tr>
<td>KotET [48]</td>
<td>CP.3</td>
<td>Feb. 6\textsuperscript{th}, 2016 [49]</td>
</tr>
</tbody>
</table>

* The DAO and the DarkDAO contract are considered to be identical.

CP.1: State Changes after External Calls
CP.2: Missing Checks for Out-of-Bound Arithmetic Ops
CP.3: Missing Checks for Failing External Calls
Part 2: Validate whether the functionalities of each rectified contract are still executed consistently.

- Use historical transaction data to re-execute each rectified contract.
- Check whether the implemented functionalities are executed still as the same.
- 268,939 historical transactions were replayed.
- Only 13 contracts showed inconsistency due to incompatible issues.
RQ3: Cost

- The average size increment for each contract is around **1.0% (49.3 bytes)**.
- The *gas* consumption for each rectified contract increases by **0.2%** on average, that is, **0.0001 USD**.
1. Background
2. Motivation
3. Automated Rectification with SMARTSHIELD
4. Evaluation
5. Conclusion
Conclusion

▪ A first step towards a **general-purpose smart contract protection** against attacks exploiting insecure contracts.

▪ An **automated smart contract rectification system**, SMARTSHIELD, to generate secure EVM bytecode without typical insecure patterns for deployment.

▪ An evaluation with 28,621 real-world buggy contracts—**87,346 (91.5%) of insecure cases** were automatically fixed.

▪ Effective and economical contract protection:
  ▪ The rectified contracts are **secure against common attacks**.
  ▪ The rectification only introduces a **0.2% average gas increment** for each contract.
In memory of medical staff who bravely fight COVID

During the new coronavirus infection in 2020:

- Li Wenliang and 8 other doctors died of illness
- More than 3,000 health workers infected

Pay the highest respect to all the medical staff!
SMARTSHIELD: Automatic Smart Contract Protection Made Easy

Yuyao Zhang\textsuperscript{1}, Siqi Ma\textsuperscript{2}, Juanru Li\textsuperscript{1}, Kailai Li\textsuperscript{1}, Surya Nepal\textsuperscript{2}, Dawu Gu\textsuperscript{1}

\textsuperscript{1}Shanghai Jiao Tong University, Shanghai, China
\textsuperscript{2}Data61, CSIRO, Sydney, Australia

Questions?