SMARTSHIELD: Automatic Smart Contract Protection Made Easy

Yuyao Zhang[1], Siqi Ma[2], Juanru Li[1], Kailai Li[1], Surya Nepal[2], Dawu Gu[1]

[1]Shanghai Jiao Tong University, Shanghai, China

[2]Data61, CSIRO, Sydney, Australia

Outline

Outline

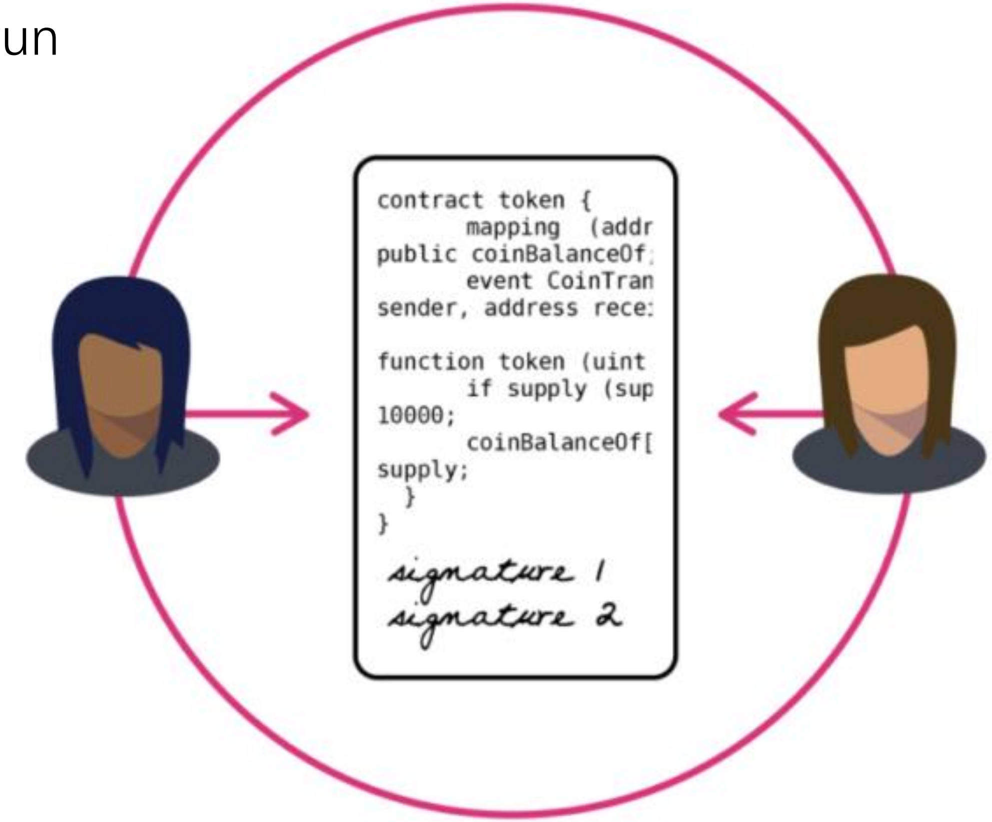
Blockchain

- 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

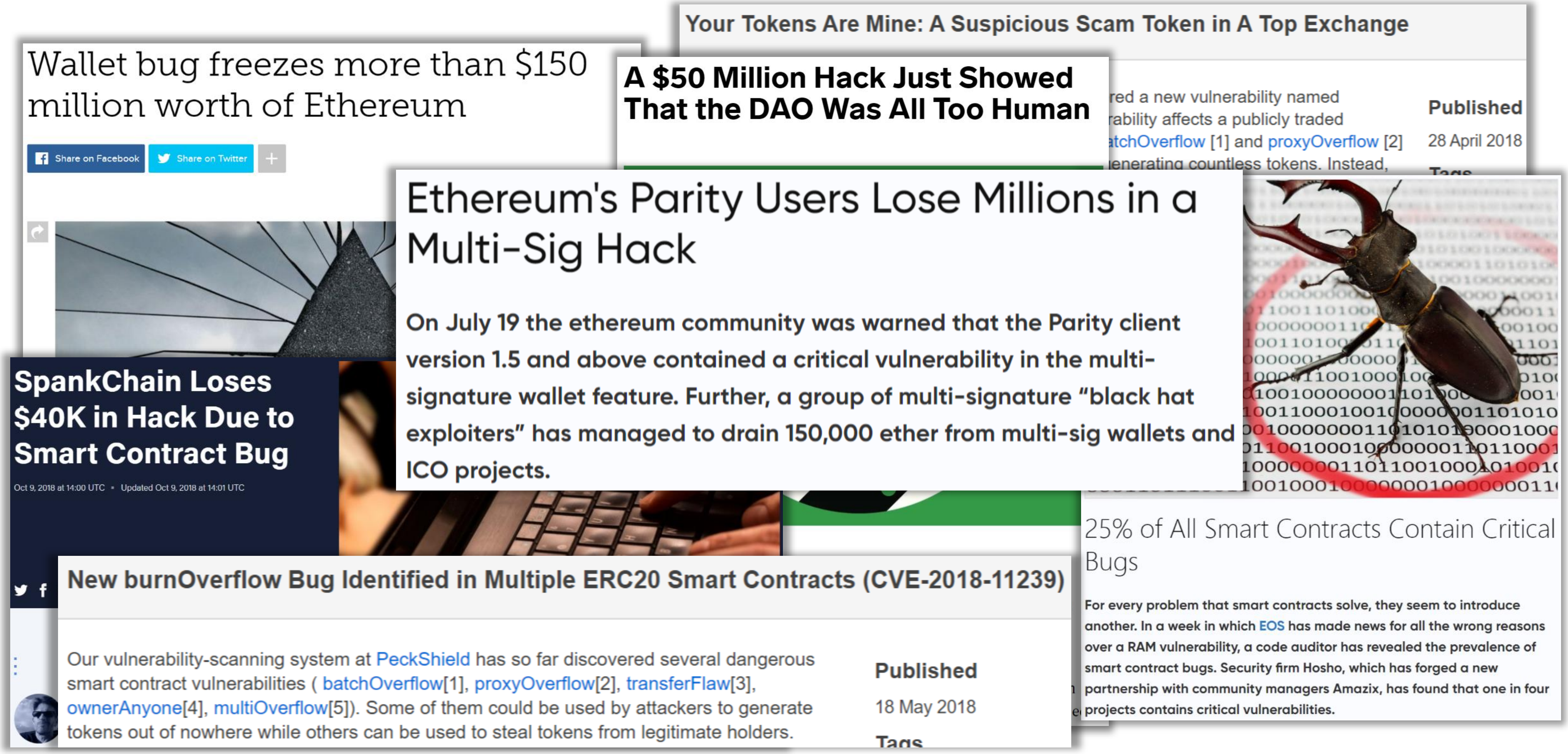
- Written in high-level languages (e.g., Solidity).
- Compiled to low-level bytecode.
- Executed on the Ethereum Virtual Machine (EVM).

```
mapping(address => uint) public balances;
...
function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
}
```

```
0000:   6001      PUSH1 0x01
0002:   60FF      PUSH1 0xFF
0004:   16        AND
0005:   6080      PUSH1 0x80
0007:   52        MSTORE
0008:   6080      PUSH1 0X80
000A:   51        MLOAD
000B:   15        ISZERO
000C:   61008A    PUSH2 0x0011
000F:   57        JUMPI
0010:   00        STOP
0011:   5B        JUMPDEST
0012:   6000      PUSH1 0x00
0014:   80        DUP1
0015:   FD        REVERT
```

Outline

Attacks on Smart Contracts



Your Tokens Are Mine: A Suspicious Scam Token in A Top Exchange

red a new vulnerability named
rability affects a publicly traded
atchOverflow [1] and proxyOverflow [2]
lenerating countless tokens. Instead,

Published
28 April 2018

Wallet bug freezes more than $150 million worth of Ethereum

Share on Facebook · Share on Twitter

A $50 Million Hack Just Showed That the DAO Was All Too Human

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.

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

Oct 9, 2018 at 14:00 UTC • Updated Oct 9, 2018 at 14:01 UTC

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 Hosho, which has forged a new partnership with community managers Amazix, has found that one in four projects contains critical vulnerabilities.

New burnOverflow Bug Identified in Multiple ERC20 Smart Contracts (CVE-2018-11239)

Our vulnerability-scanning system at PeckShield has so far discovered several dangerous smart contract vulnerabilities (batchOverflow[1], proxyOverflow[2], transferFlaw[3], ownerAnyone[4], multiOverflow[5]). Some of them could be used by attackers to generate tokens out of nowhere while others can be used to steal tokens from legitimate holders.

Published
18 May 2018

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

- **Code Pattern 2: Missing Checks for Out-of-Bound Arithmetic Operations.**
  - An arithmetic operation is executed without checking the data validity in advance.
  - May cause an **arithmetic bug**.

```
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);
}
```

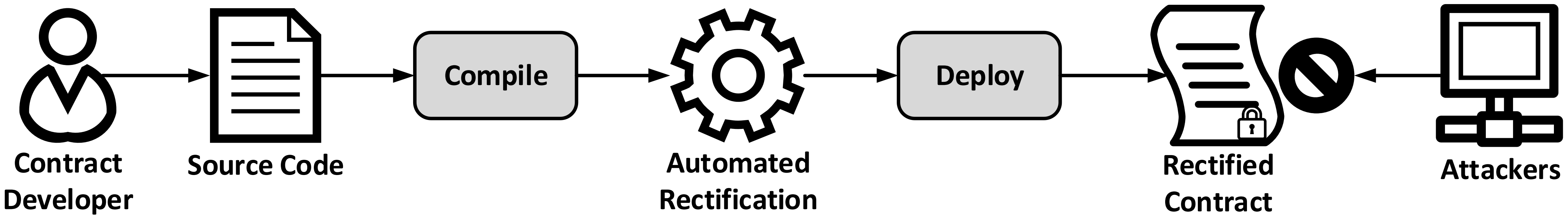
Insecure Code Patterns in Smart Contracts

- **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**.

```
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

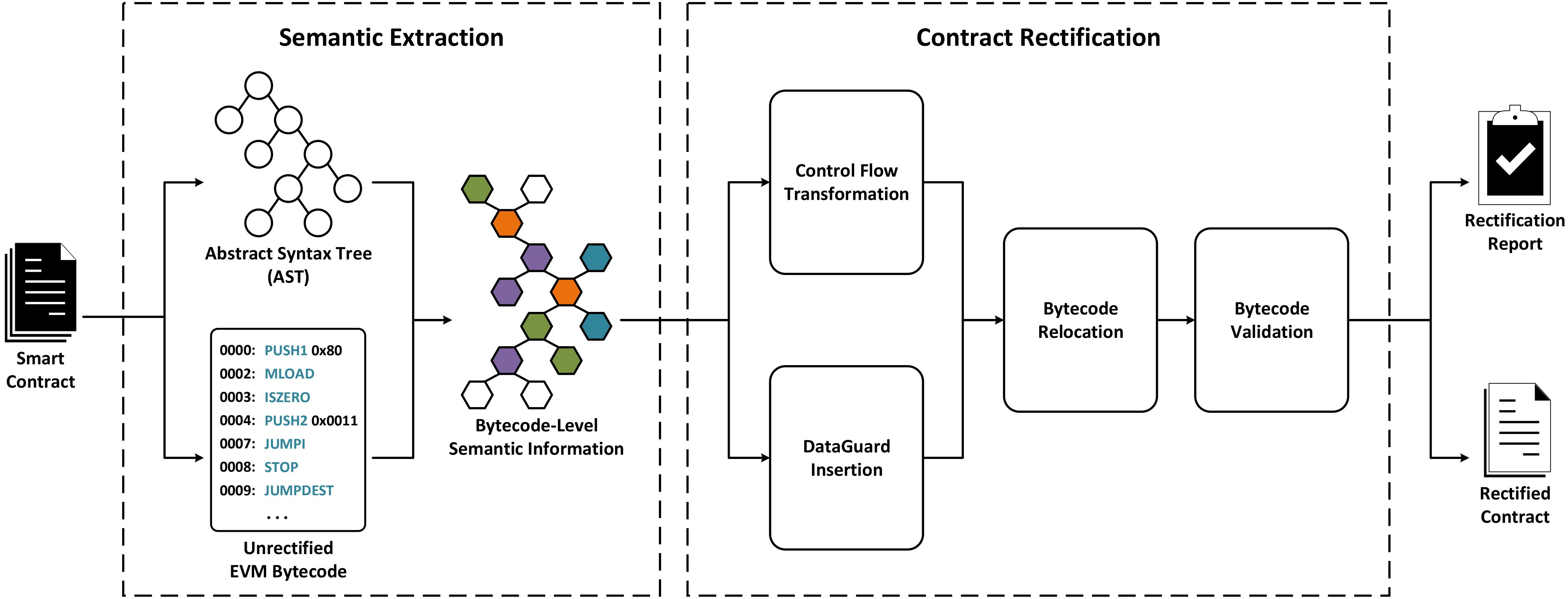
- Automatically fix insecure cases with typical patterns in smart contracts **before** their deployments.



- **Challenges & Solutions:**
  - Compatibility → Bytecode-Level Program Analysis.
  - Reliability → Semantic-Preserving Code Transformation.
  - Economy → Gas Optimization.

Outline

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

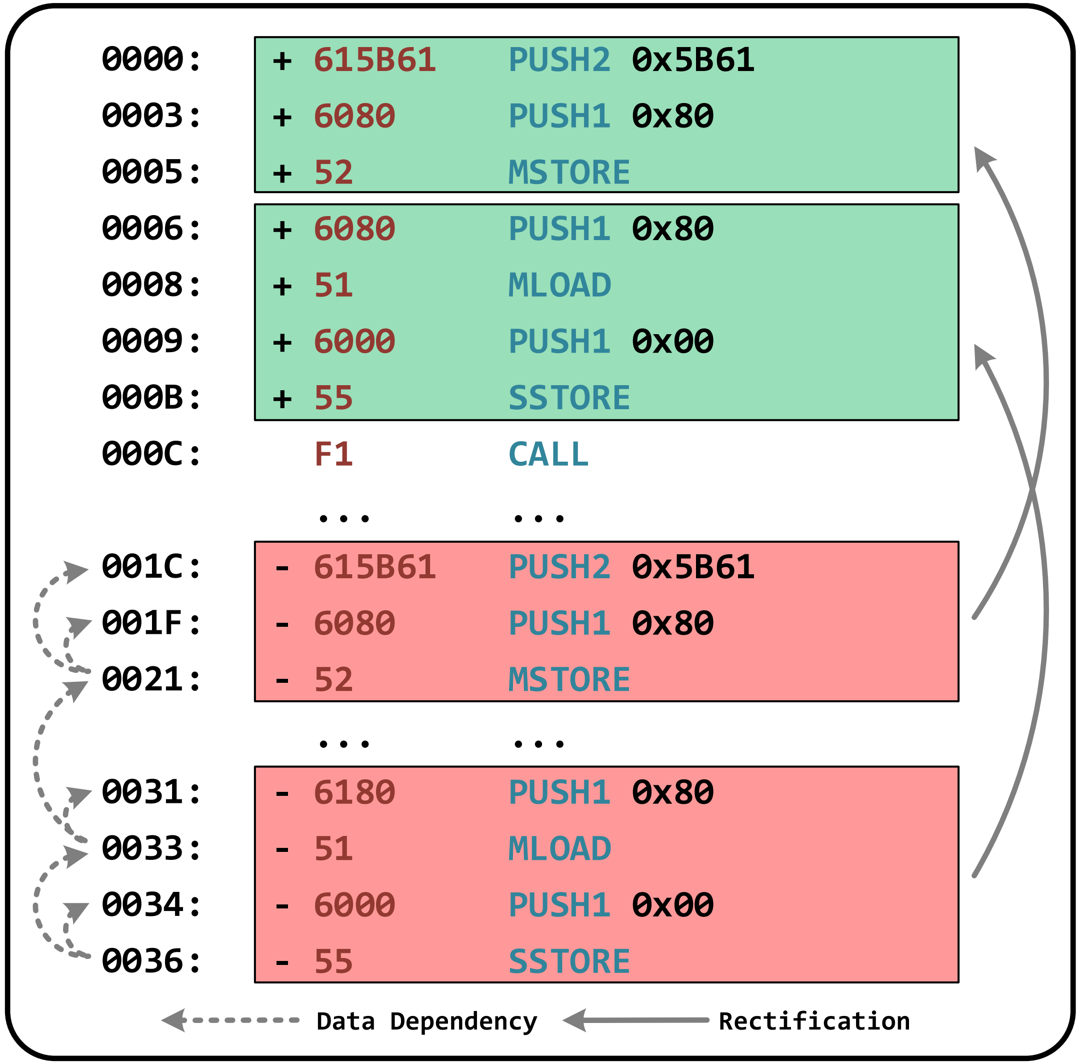
- **Bytecode-Level Semantic Information:**
  - Control and data dependencies among instructions in EVM bytecode.
  - Necessary for further code transformation and secure bytecode generation.

- 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*.

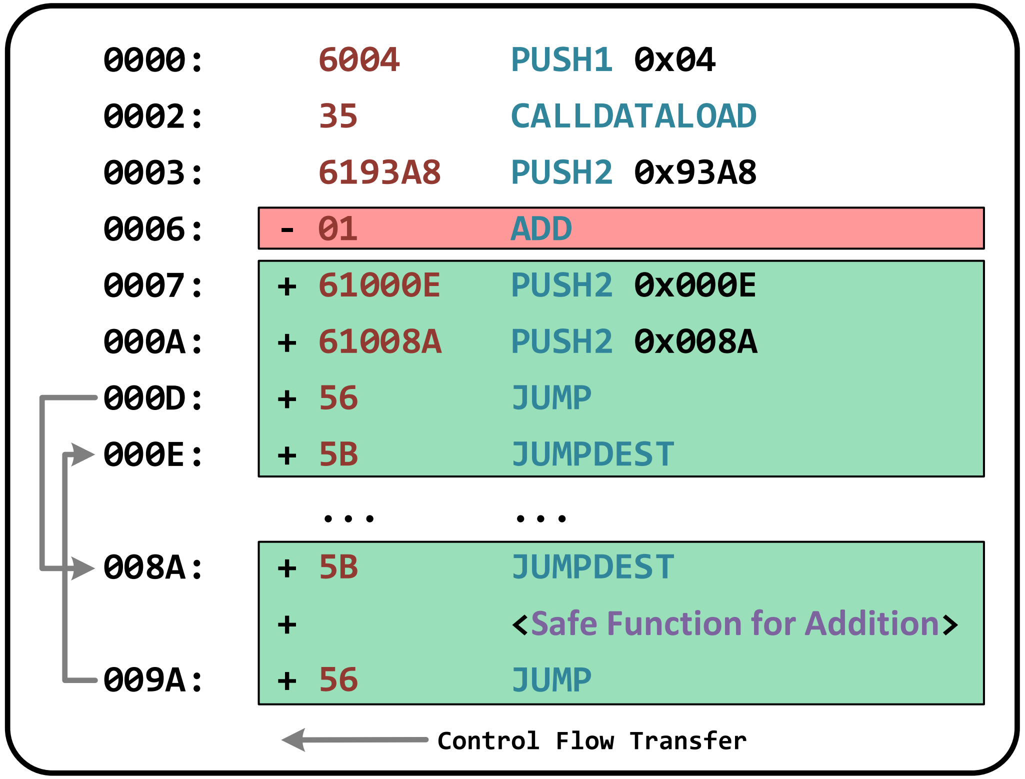
- Adjust the original control flow by moving state change operations to the front of external calls.
- Preserve the original dependencies among instructions in EVM bytecode.



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.

Category	Instruction	Operation	DataGuard
Arithmetic ops	ADD SUB MUL	$a+b$ $a-b$ $a \times b$	$a+b \geq a$ $a \geq b$ $a \times b \div a = b$
External calls	CALL	$ret = a.call()$	$ret \neq 0$



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

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.

Category	**# of insecure cases**	**# of insecure contracts**
CP.1	4,521	726
CP.2	80,825	25,470
CP.3	10,156	4,811
Total	95,502	28,621*

∗ 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.

Category	**# of eliminated cases**	**# of uneliminable cases**	**# of rectified contracts** Fully	Partially
CP.1	3,567	954	573	153
CP.2	74,642	6,183	21,815	3,655
CP.3	9,137	1,019	4,362	449
Total	87,346	8,156	25,060*	3,561*

∗ 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

- 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.

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

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

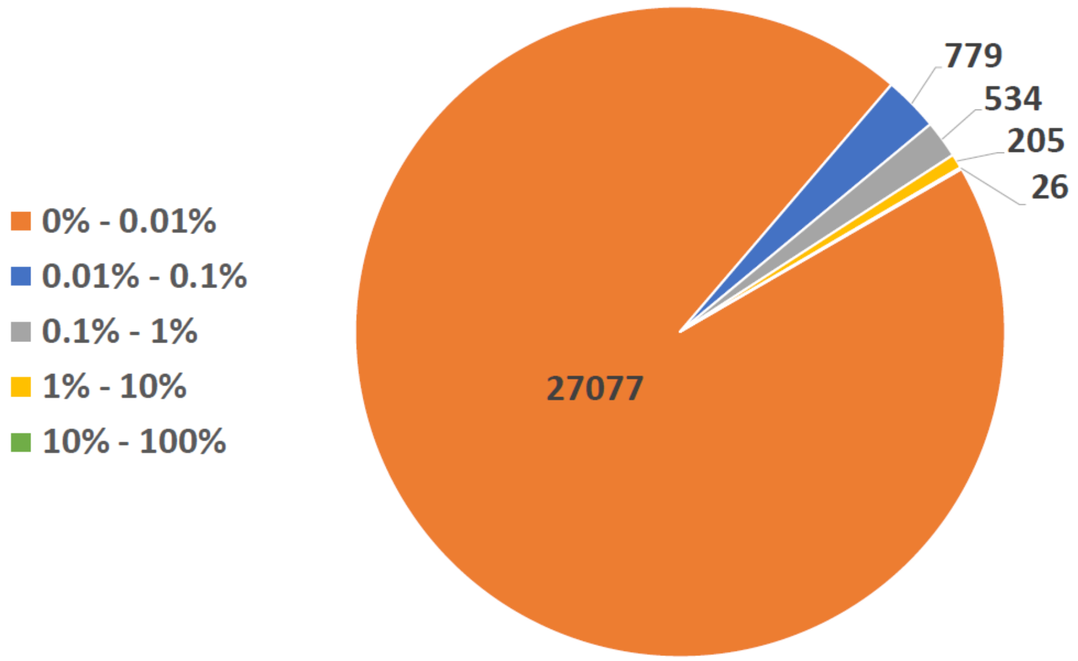
CP.1: State Changes after External Calls
CP.2: Missing Checks for Out-of-Bound Arithmetic Ops
CP.3: Missing Checks for Failing External Calls

RQ2: Correctness

- **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**.



Outline

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[1], Siqi Ma[2], Juanru Li[1], Kailai Li[1], Surya Nepal[2], Dawu Gu[1]

[1]Shanghai Jiao Tong University, Shanghai, China

[2]Data61, CSIRO, Sydney, Australia

Questions?