University of Luxembourg

Interdisciplinary Centre for Security, Reliability and Trust

ÆGIS: Shielding Vulnerable Smart Contracts Against Attacks

Christof Ferreira Torres, Mathis Baden, Robert Norvill, Beltran Borja Fiz Pontiveros, Hugo Jonker and Sjouke Mauw

Radboud Universiteit

Open Universiteit

SnT

uni.lu UNIVERSITÉ DU LUXEMBOURG

Attacks on Smart Contracts

KLINT FINLEY BUSINESS 06.18.16 04:30 AM

A $50 MILLION HACK JUST SHOWED THAT THE DAO WAS ALL TOO HUMAN

20 JULY 2017 / #ETHEREUM #BLOCKCHAIN #SECURITY

A hacker stole $31M of Ether — how it happened, and what it means for Ethereum

SpankChain, a cryptocurrency project focused on the adult industry, has suffered a breach that saw almost $40,000 in ethereum (ETH) stolen.

Wallet bug freezes more than $150 million worth of Ethereum

A hackers' dream payday: Ledf.Me and Uniswap lose $25 million worth of cryptocurrency

by Alina Bizga on April 21, 2020

Motivation

- Smart contracts repeatedly suffer from exploits costing millions of dollars:
  - 2016: The DAO hack **($50M)**
  - 2017: Parity Wallet hacks
  - 2018: Bancor, Fomo 3D and Spankchain hacks **($40K)**
  - 2019: MakerDAO and bZx hacks
  - 2020: Uniswap and Lendf.me hack **($25M)**
  - 2021: ???

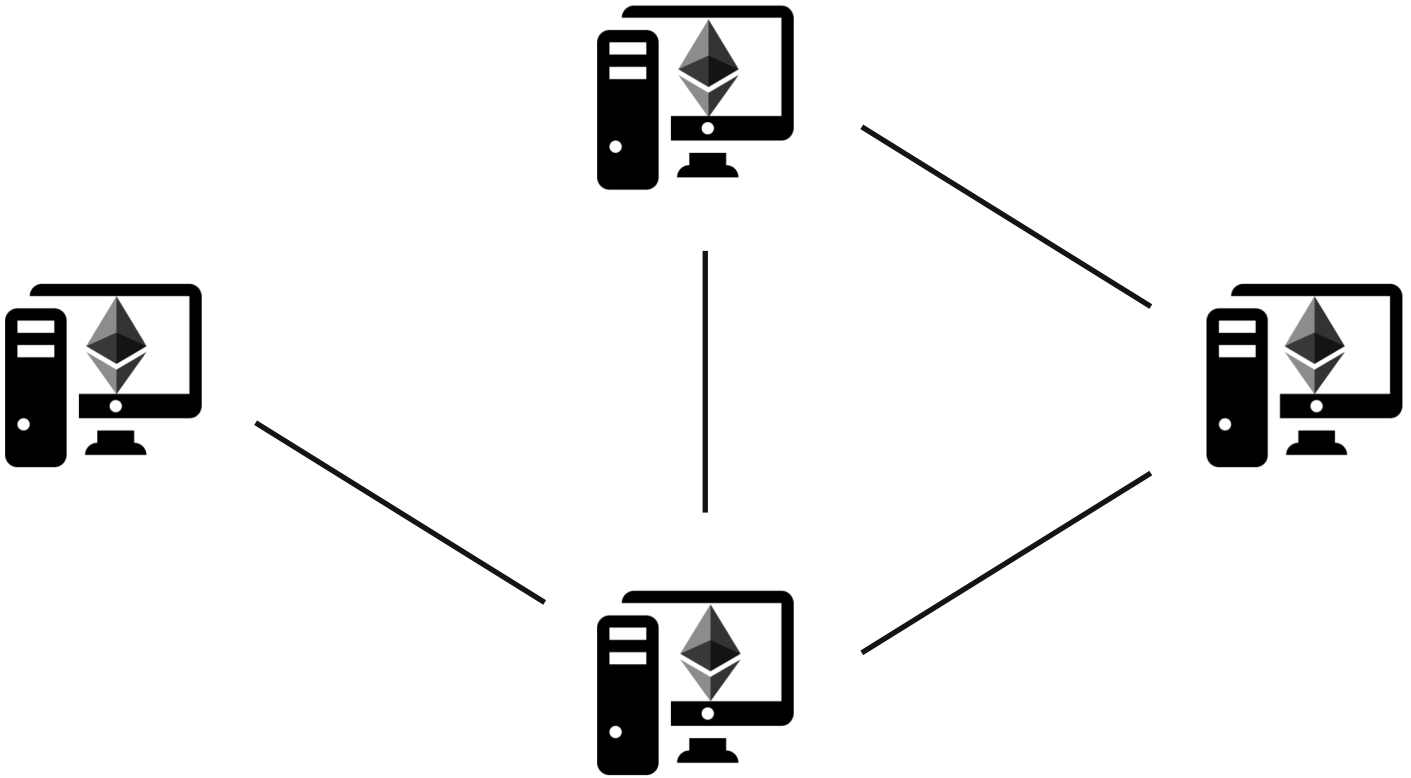
!! Reentrancy Attacks !!

- Smart contracts cannot be modified once deployed

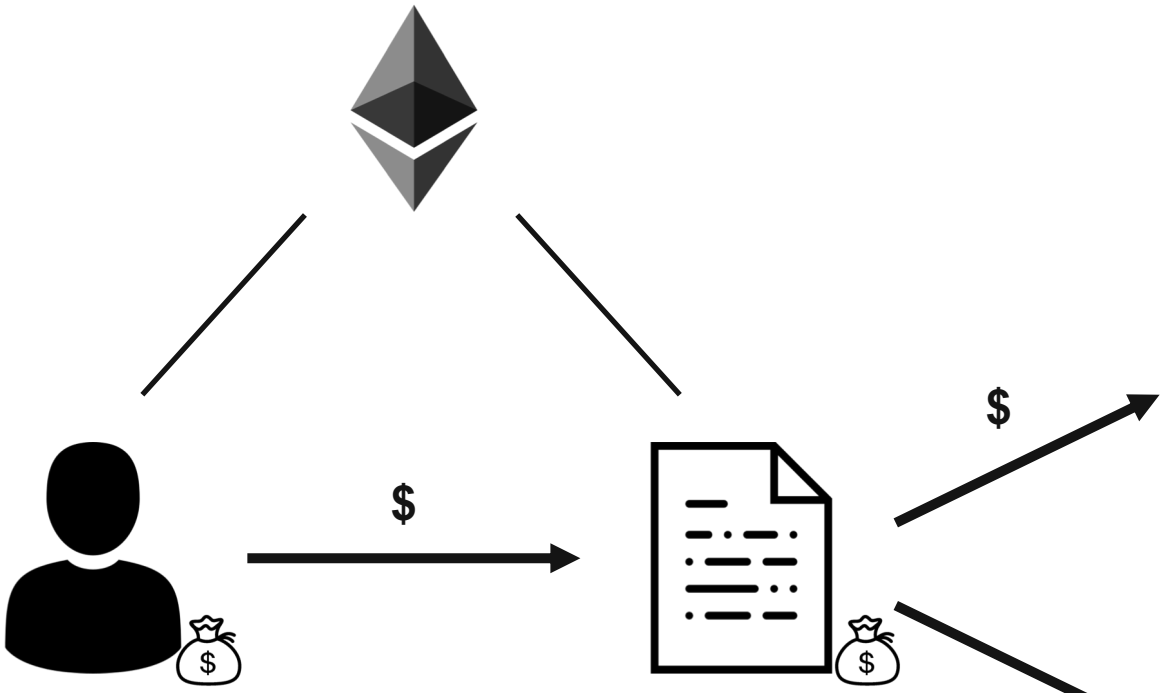
- Existing tools suffer from **low precision** and are **not generic** enough

Ethereum
Crash Course

Ethereum Blockchain



Ethereum Accounts



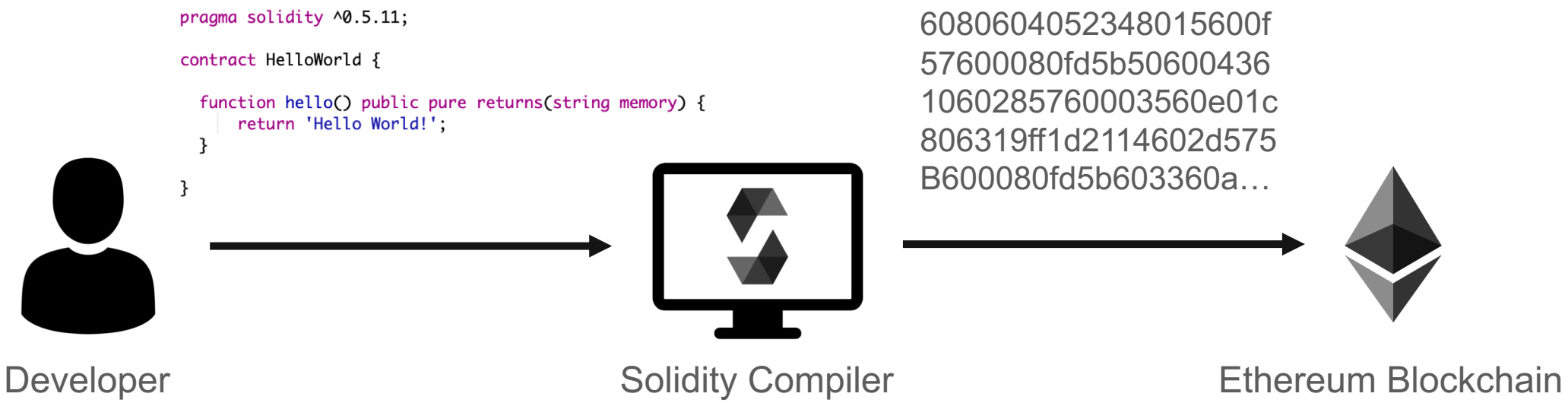
Externally Owned Account

- Address
- Balance

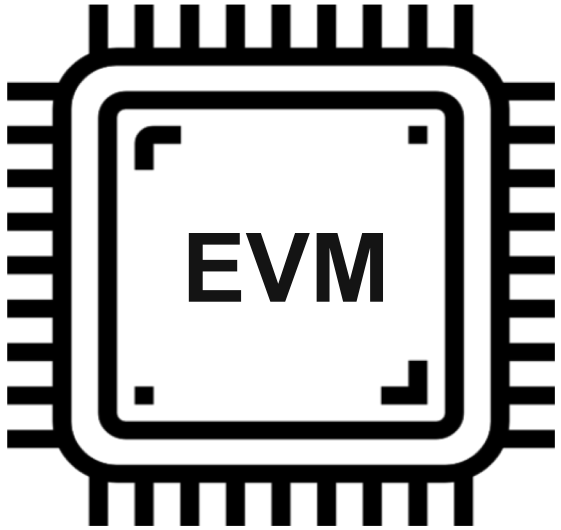
Contract Account

- Address
- Balance
- Code
- Storage

Ethereum Smart Contracts



Ethereum Virtual Machine



- Over **100** instructions:
  - Stack instructions:
    PUSH, SWAP, …
  - Arithmetic instructions:
    ADD, SUB, MUL, …
  - Memory instructions:
    SLOAD, SSTORE, …
  - Control-flow instructions:
    JUMP, JUMPI, …
  - Contract instructions:
    CALL, SELFDESTRUCT, …
  - Error handling instructions:
    REVERT, INVALID, …

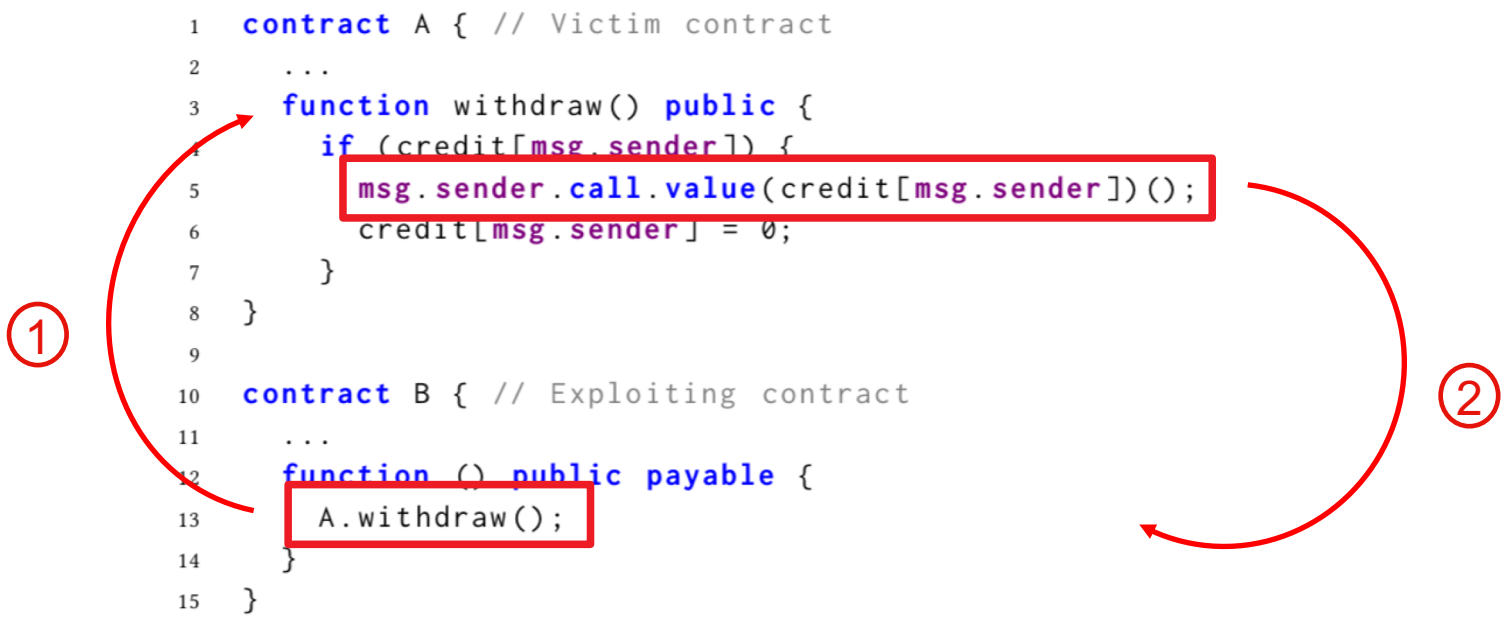
ÆGIS

Smart Shielding of
***(not so)** Smart Contracts*

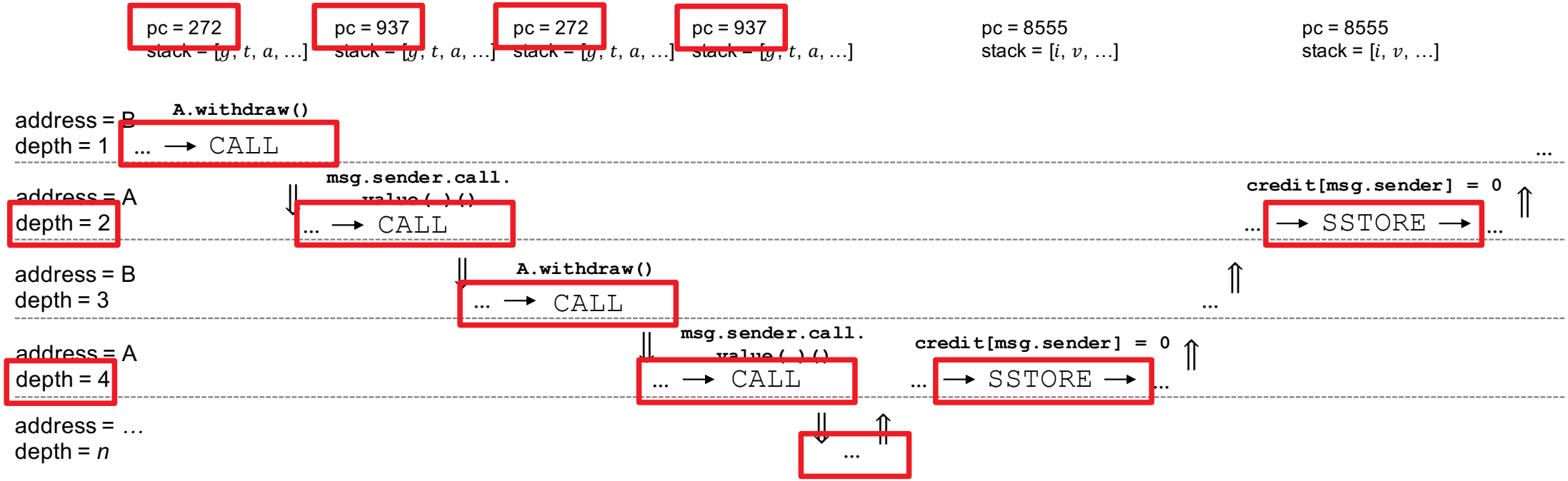
SnT

uni.lu
UNIVERSITÉ DU LUXEMBOURG

Reentrancy Example



Execution Flow of a Reentrancy Attack




```
1  contract A { // Victim contract
2    ...
3    function withdraw() public {
4      if (credit[msg.sender]) {
5        msg.sender.call.value(credit[msg.sender])();
6        credit[msg.sender] = 0;
7      }
8  }
```

```
10 contract B { // Exploiting contract
11   ...
12   function () public payable {
13     A.withdraw();
14   }
15 }
```

ÆGIS

Generic Attack Detection

SnT

uni.lu
UNIVERSITÉ DU LUXEMBOURG

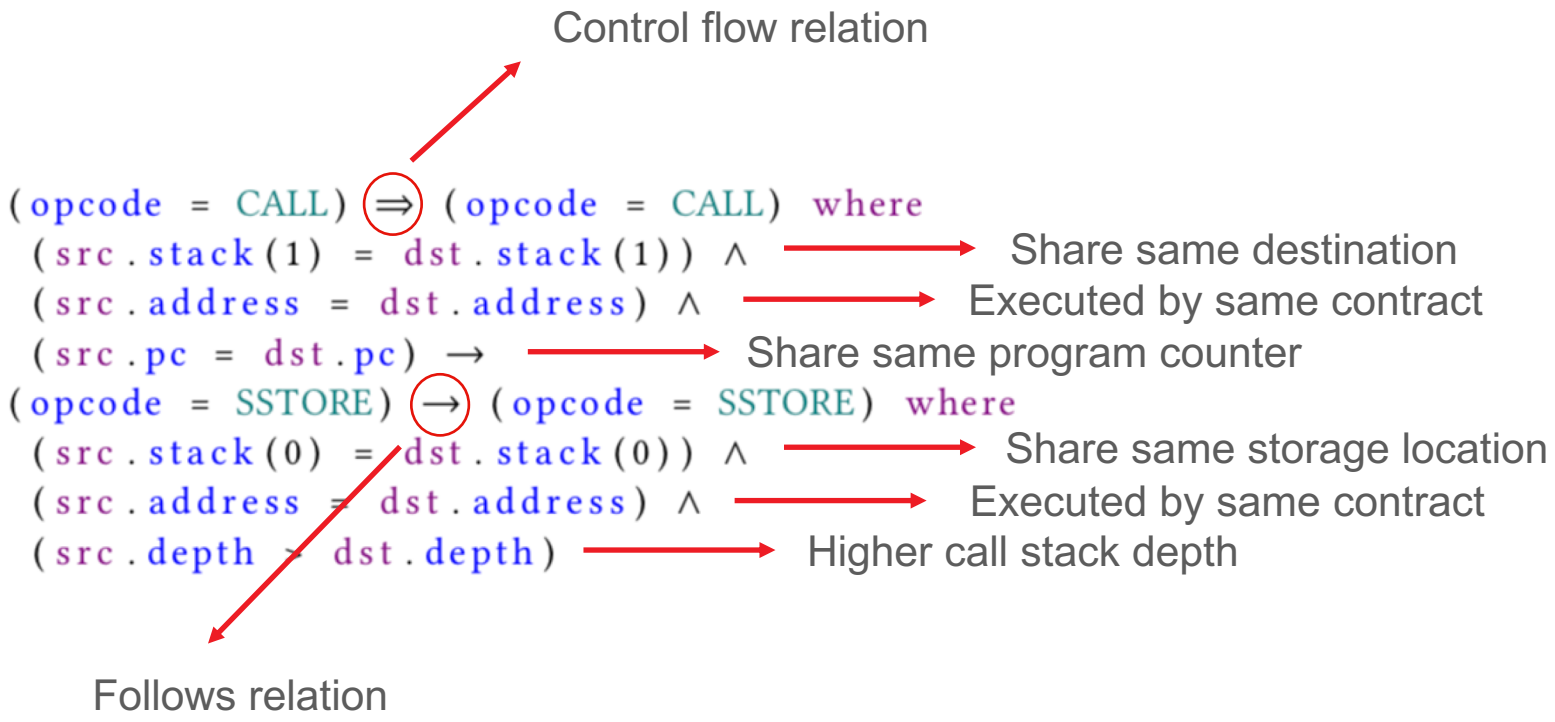
Generic Attack Detection

- We propose a domain-specific language (DSL)
  - Tailored to the execution model of the EVM
  - Describe malicious control and data flows as *attack patterns*
  - **Attack pattern:** Sequence of relations between EVM instructions

- We distinguish between 3 relations:
  - **Control Flow** (⇒)
  - **Data Flow** (⇝)
  - **Follows** (⟶)

$$
\begin{array}{lll}
\langle instr \rangle & ::= & \texttt{CALL} \mid \texttt{CALLDATALOAD} \mid \texttt{SSTORE} \mid \texttt{JUMPI} \mid \ldots \\
\langle exec \rangle & ::= & \textbf{depth} \mid \textbf{pc} \mid \textbf{address} \mid \textbf{stack}(\text{int}) \mid \textbf{stack.result} \mid \\
 & \mid & \textbf{memory}(\text{int}, \text{int}) \mid \textbf{transaction.}\langle trans \rangle \\
 & \mid & \textbf{block.}\langle block \rangle \\
\langle trans \rangle & ::= & \textbf{hash} \mid \textbf{value} \mid \textbf{from} \mid \textbf{to} \mid \ldots \\
\langle block \rangle & ::= & \textbf{number} \mid \textbf{gasUsed} \mid \textbf{gasLimit} \mid \ldots \\
\langle comp \rangle & ::= & < \mid > \mid \leq \mid \geq \mid = \mid \neq \mid + \mid - \mid \cdot \mid / \\
\langle expr \rangle & ::= & \textbf{(src.}\langle exec \rangle\ \langle comp \rangle\ \langle expr \rangle\textbf{)}\ [\wedge\ \langle expr \rangle] \\
 & \mid & \textbf{(}\langle expr \rangle\ \langle comp \rangle\ \textbf{dst.}\langle exec \rangle\textbf{)}\ [\wedge\ \langle expr \rangle] \\
 & \mid & \textbf{(src.}\langle exec \rangle\ \langle comp \rangle\ \textbf{src.}\langle exec \rangle\textbf{)}\ [\wedge\ \langle expr \rangle] \\
 & \mid & \textbf{(src.}\langle exec \rangle\ \langle comp \rangle\ \textbf{dst.}\langle exec \rangle\textbf{)}\ [\wedge\ \langle expr \rangle] \\
 & \mid & \textbf{(dst.}\langle exec \rangle\ \langle comp \rangle\ \textbf{dst.}\langle exec \rangle\textbf{)}\ [\wedge\ \langle expr \rangle] \\
 & \mid & \textbf{(src.}\langle exec \rangle\ \langle comp \rangle\ \text{int}\textbf{)} \mid \textbf{(dst.}\langle exec \rangle\ \langle comp \rangle\ \text{int}\textbf{)} \\
\langle rel \rangle & ::= & \Rightarrow \mid \rightsquigarrow \mid \longrightarrow \\
\langle pattern \rangle & ::= & \textbf{(opcode} = \langle instr \rangle\textbf{)}\ \langle rel \rangle\ \textbf{(opcode} = \langle instr \rangle\textbf{)}\ [\textbf{where}\ \langle expr \rangle] \\
 & \mid & \langle pattern \rangle\ \langle rel \rangle\ \textbf{(opcode} = \langle instr \rangle\textbf{)}\ [\textbf{where}\ \langle expr \rangle] \\
 & \mid & \textbf{(opcode} = \langle instr \rangle\textbf{)}\ \langle rel \rangle\ \langle pattern \rangle\ [\textbf{where}\ \langle expr \rangle]
\end{array}
$$

Example: Designing an Attack Pattern for Reentrancy



ÆGIS

Decentralized Security Updates

SnT

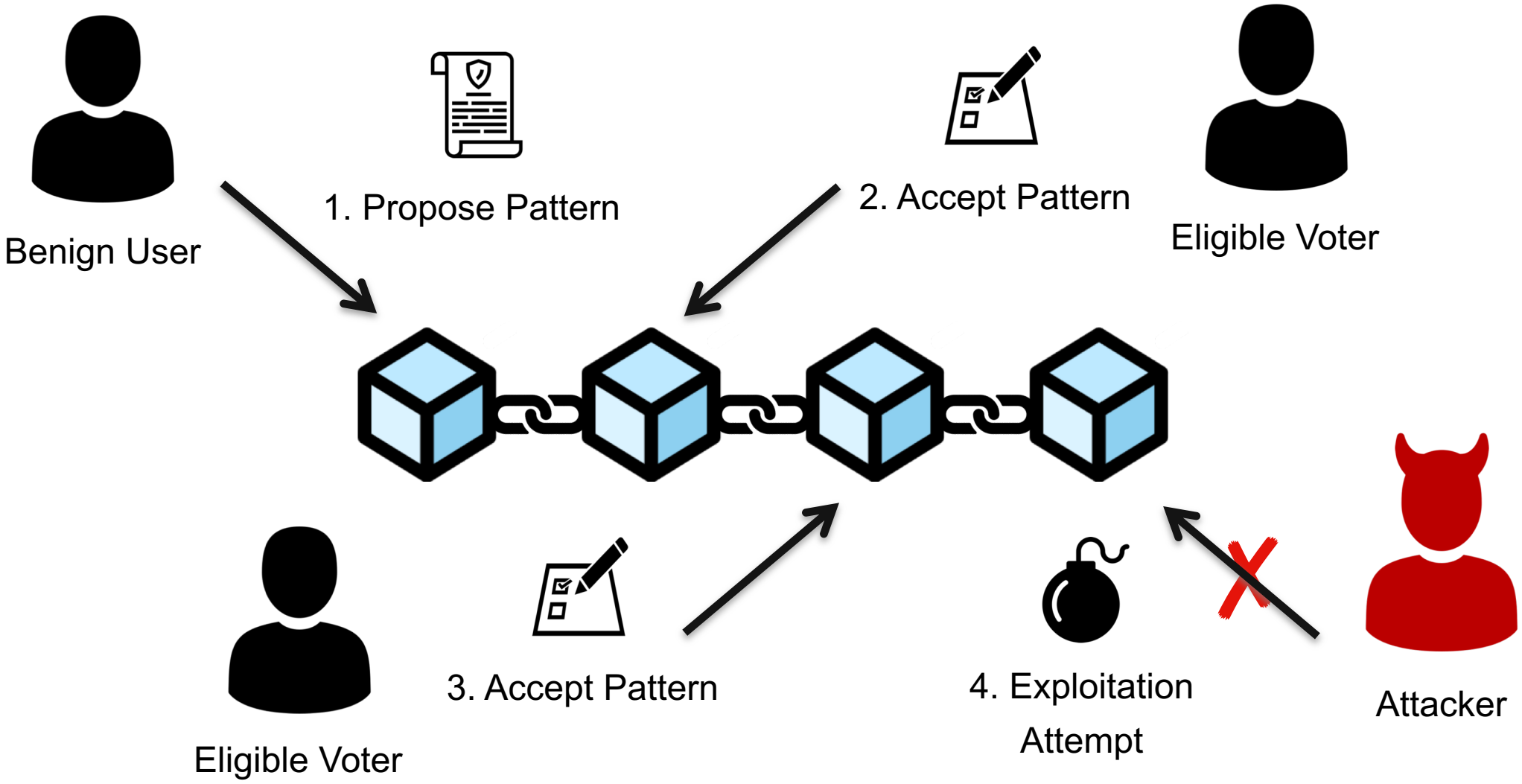
uni.lu
UNIVERSITÉ DU LUXEMBOURG

Decentralized Security Updates

- **Two questions remain open:**
  - How to distribute and enforce same patterns across all clients?
  - How to prevent a single entity from deciding which patterns are added or removed?

- **Solution:**
  - Store patterns inside a smart contract
  - Blockchain protocol guarantees that every client uses the same patterns
  - Governance of patterns is decentralized by allowing users to propose and vote for patterns

Decentralized Security Updates



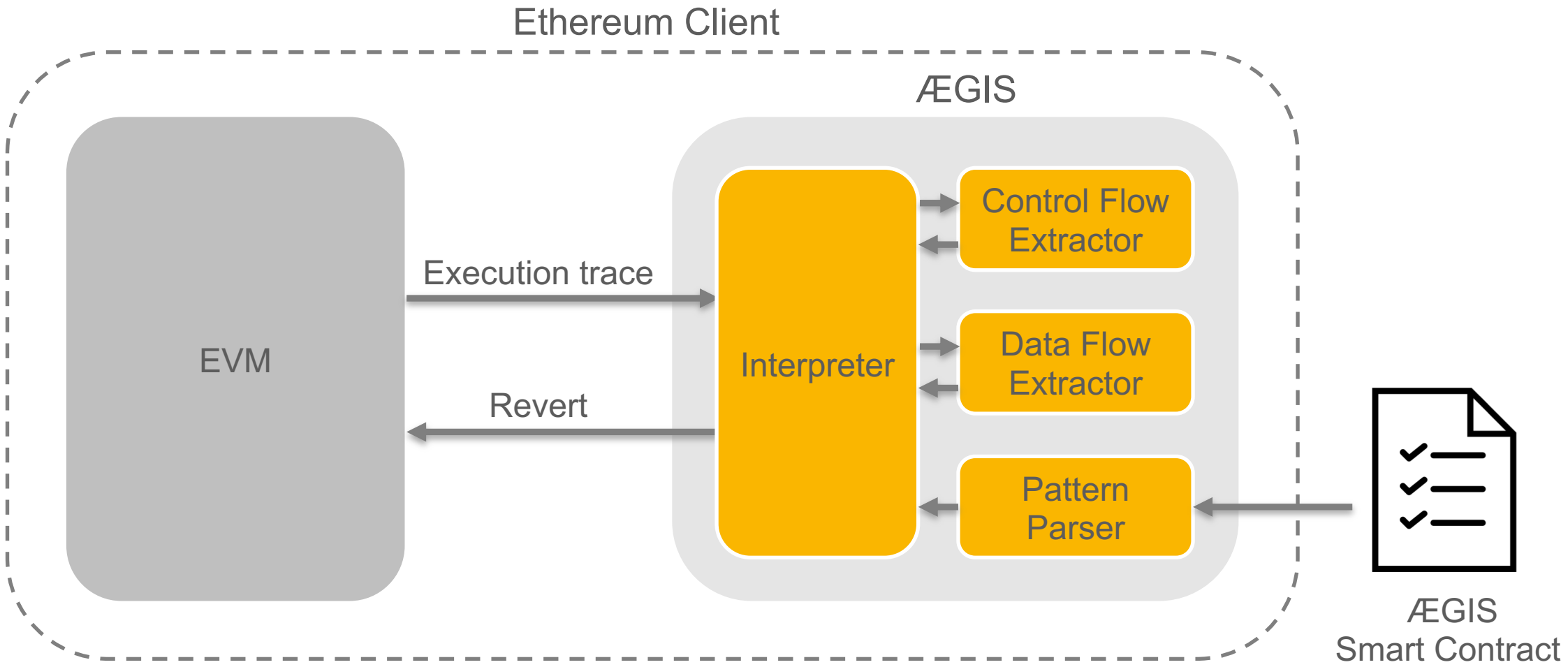
ÆGIS

Putting it all together…

SnT

uni.lu
UNIVERSITÉ DU LUXEMBOURG

ÆGIS's System Architecture



Evaluation

Experiments

- We compared ÆGIS to state-of-the-art runtime reentrancy detection tools:
  - ECFChecker [POPL'18]
  - Sereum [NDSS'19]

1. Comparison to Sereum

- Sereum: 16 suspect contracts, 14 false positives
- **ÆGIS** on same 16 contracts:
  - No false positives
  - No false negatives

	CCRB	DAO	0x7484a1	proxyCC	DAC	DSEthToken	0x695d73	EZC	0x98D8A6	WEI	0xbD7CeC	0xF4ee93	Alarm	0x771500	KissBTC	LotteryGameLogic
SEREUM	FP	TP	FP	FP	FP	TP	FP	FP	FP	FP	FP	FP	FP	FP	FP	FP
ÆGIS	TN	TP	TN	TN	TN	TP	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN

Table 1: Comparison between SEREUM and ÆGIS on the effectiveness of detecting reentrancy attacks.

2. Detecting Reentrancy With Manual Locks

- ECFChecker has difficulties in detecting cross-function reentrancy
- Sereum has difficulties in detecting manual locks
- **ÆGIS correctly identifies cross-function reentrancy and distinguishes between manual locks and reentrancy**

Smart Contract	Reentrancy Type	ECFChecker	Sereum	ÆGIS
VulnBankNoLock	Same-Function	TP	TP	TP
	Cross-Function	FN	TP	TP
VulnBankBuggyLock	Same-Function	TN	FP	TN
	Cross-Function	FN	TP	TP
VulnBankSecureLock	Same-Function	TN	FP	TN
	Cross-Function	TN	FP	TN

Table 2: Comparison between ECFChecker, Sereum and ÆGIS on the effectiveness of detecting same-function and cross-function reentrancy attacks with manually introduced locks.

3. Detecting Unconditional Reentrancy

- Sereum does not detect unconditional reentrancy

  → Authors assume reentrancy is always guarded

- **ÆGIS detects unconditional reentrancy**

  → **Attack pattern does not rely on conditions**

```
1  contract VulnBank {
2    mapping (address => uint) public userBalances;
3
4    function deposit() public payable {
5      userBalances[msg.sender] += msg.value;
6    }
7
8    function withdrawAll() public {
9      uint amountToWithdraw = userBalances[msg.sender];
10     msg.sender.call.value(amountToWithdraw)("");
11     userBalances[msg.sender] = 0;
12   }
13 }
```

Unconditional Reentrancy

4. Comparison with Sereum's Large-Scale Blockchain Analysis

- On the same 4.5 million blocks:
  - Sereum detects 2 reentrant contracts
  - **ÆGIS detects 7 reentrant contracts**

Vulnerability	Contracts	Transactions
Same-Function Reentrancy	7	822
Cross-Function Reentrancy	5	695
Delegated Reentrancy	0	0
Create-Based Reentrancy	0	0
Parity Wallet Hack 1	3	80
Parity Wallet Hack 2	236	236
Total Unique	248	1118

Table 3: Number of vulnerable contracts detected by ÆGIS.

Detected only by ÆGIS

Detected by Sereum + ÆGIS

Contract Address	Block Range
0xd654bdd32fc99471455e86c2e7f7d7b6437e9179	1680024 - 1680238
0xbb9bc244d798123fde783fcc1c72d3bb8c189413	1718497 - 2106624
0xf01fe1a15673a5209c94121c45e2121fe2903416	1743596 - 1743673
0x304a554a310c7e546dfe434669c62820b7d83490	1881284 - 1881284
0x59752433dbe28f5aa59b479958689d353b3dee08	3160801 - 3160801
0xbf78025535c98f4c605fbe9eaf672999abf19dc1	3694969 - 3695510
0x26b8af052895080148dabbc1007b3045f023916e	4108700 - 4108700

Table 4: Same-function reentrancy vulnerable contracts detected by ÆGIS. Contracts highlighted in gray have only been detected by ÆGIS and not by SEREUM.

Conclusion

Conclusion

- Smart contract protection of exisiting tools is **insufficient** or **requires client updates**
- ÆGIS detects and blocks attacks at runtime via **generic attack patterns**
- Compared to Sereum and ECFChecker:
  - More attacks identified
  - No false positives
- New mechanism for quick, transparent and **decentralized security updates**

Why You Should Read The Paper?

- It's a fun paper!
- It has things not shown in the presentation, e.g.:
  - How to specify attack patterns for other types of attacks?
  - How to pick eligible voters for the selection of new patterns?
  - How to provide incentives for voting?
  - How to prevent attackers from exploiting contracts before pattern is accepted?
  - …

Questions?

All **code** & **data** is available on GitHub:

https://github.com/christoftorres/Aegis

Contact information:

christof.torres@uni.lu

Supported by:

Luxembourg National Research Fund