TxSpector Revisited:

Uncovering Attacks in Ethereum from Transactions

Carla Piazza, Sabina Rossi, Semia Guesmi

UniUd & UniVe
30/05/2025

TxSpector as a Foundation for Future Work

Why This Talk?

- Introduced at **USENIX Security 2020.**
- **TxSpector:** A generic, logic-driven framework for analyzing Ethereum transactions.
- Detects real-world attacks on smart contracts through transaction-level forensic analysis.
- Agenda:

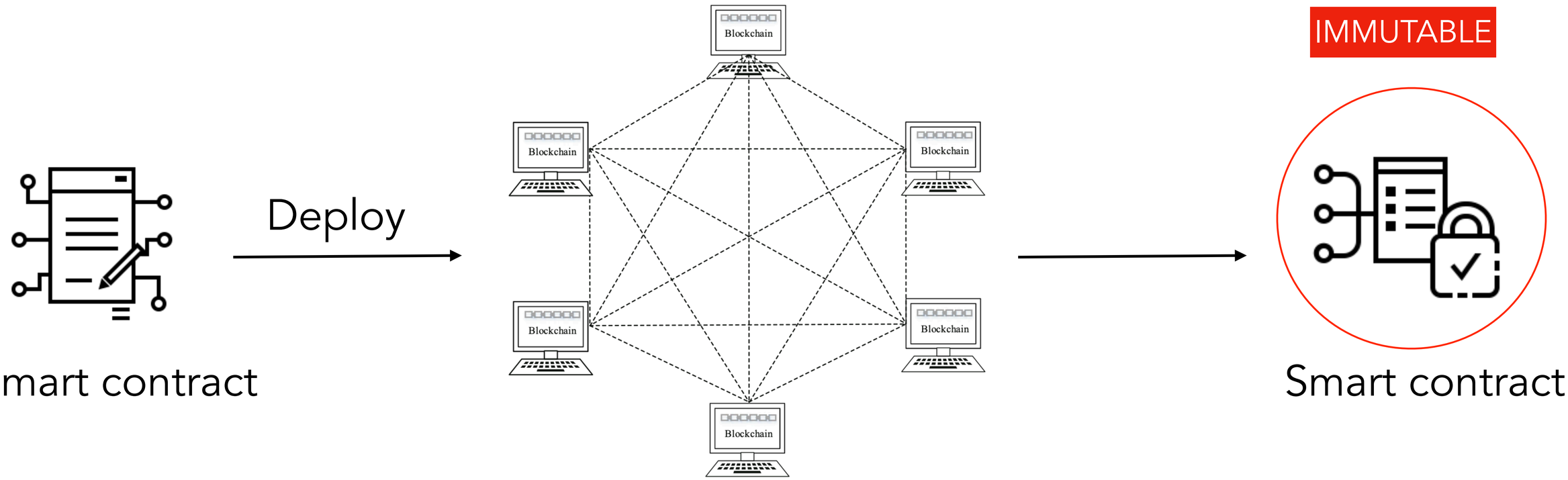
  Explain how TxSpector works.

  Review strengths and weaknesses.

  Present my planned improvements.

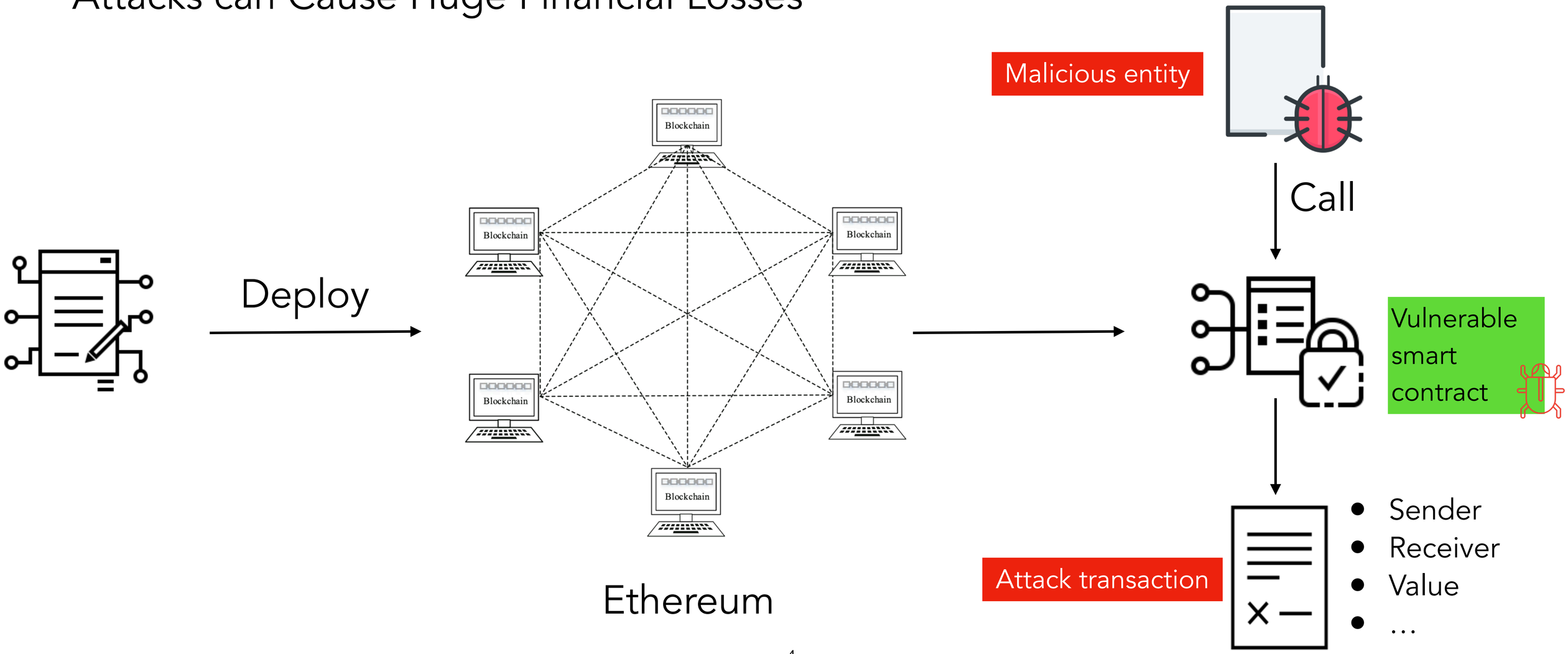
Smart Contract Security: Context and Motivation

Deployed Smart Contracts are Immutable



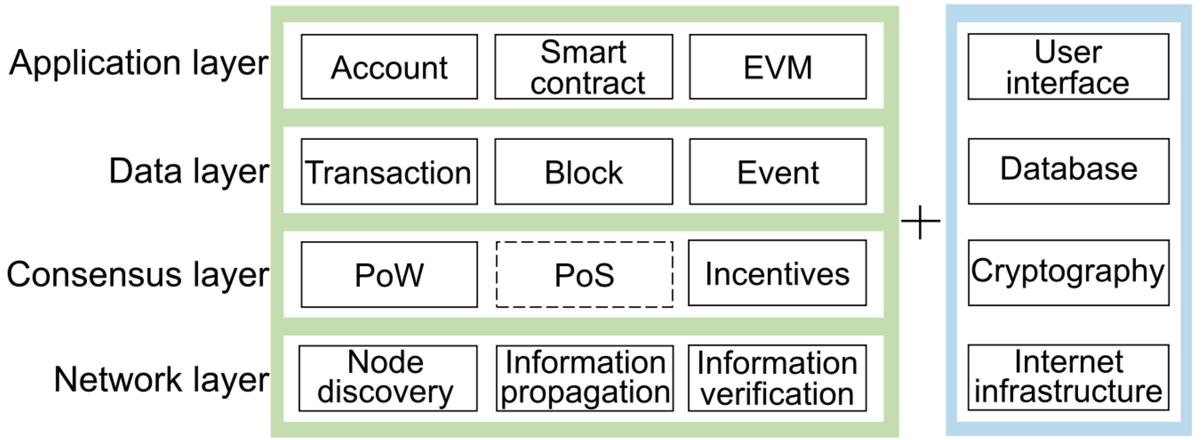
Smart Contract Security: Context and Motivation

Attacks can Cause Huge Financial Losses



Transaction vs. bytecode

- Focus on the **data layer** where real blockchain interactions happen.
- Analyze Reentrancy Attacks through r**eal transaction** behavior.
- Transactions show how smart contracts interact with each other in real use.
- Transaction data reveals important forensic info: attack patterns, attacker & victim addresses.
- If a function is never called, transactions won't reveal vulnerabilities related to it. But if a contract or function is never used in any transaction, its vulnerabilities are not exploited either.

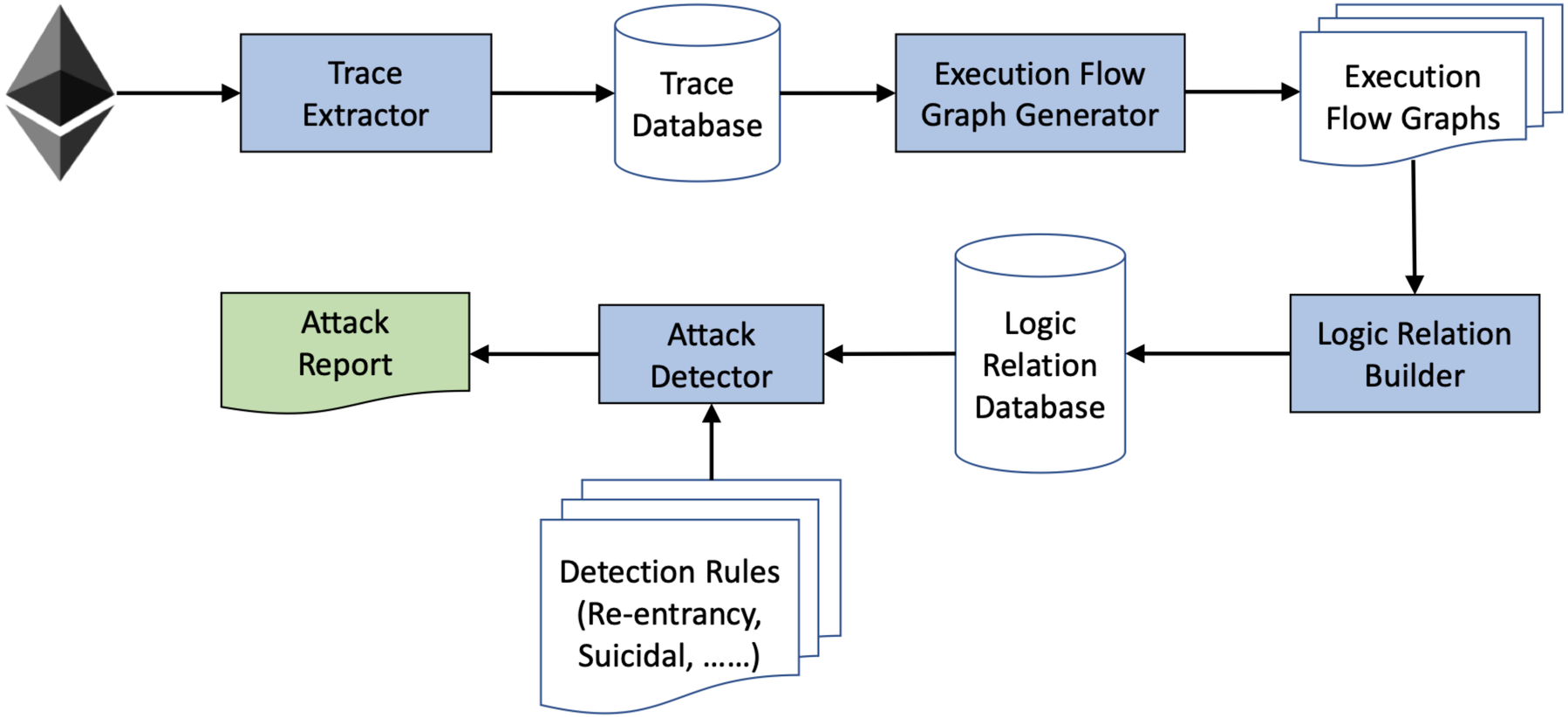


Smart Contract Security: Context and Motivation

Few Works focus on Transactions

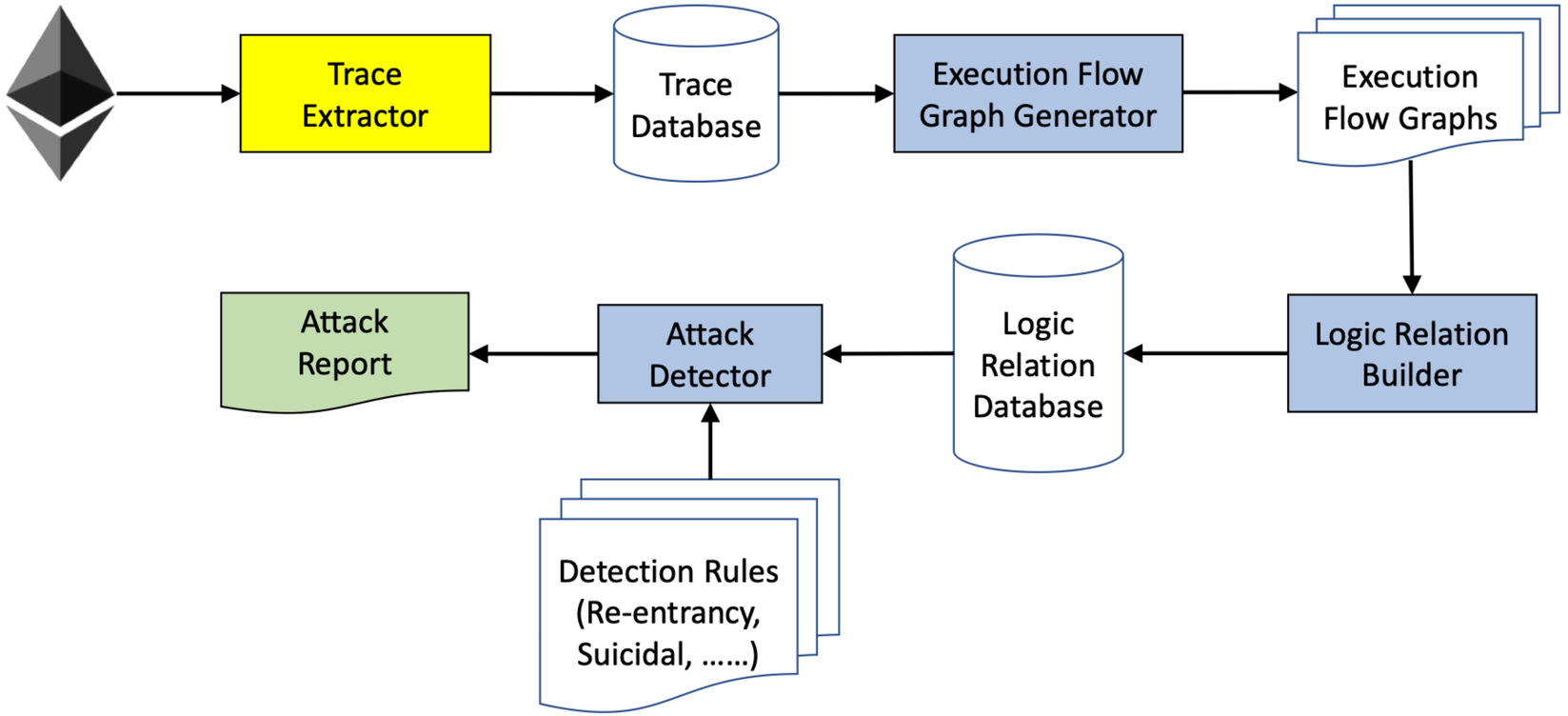
Tools	Transaction	Bytecode	Source code
Oyente [1]		X	
Securify [2]		X	
Vandal [3]		X	
SmartDagger [4]		X	
Mythril [5]			X
Ethor [6]		X	
Sereum [7]	X		
ECFchecker [8]	X		
TxSpector [9]	X		

TxSpector overview



Detailed Design - Trace Extractor

- Record bytecode-level trace

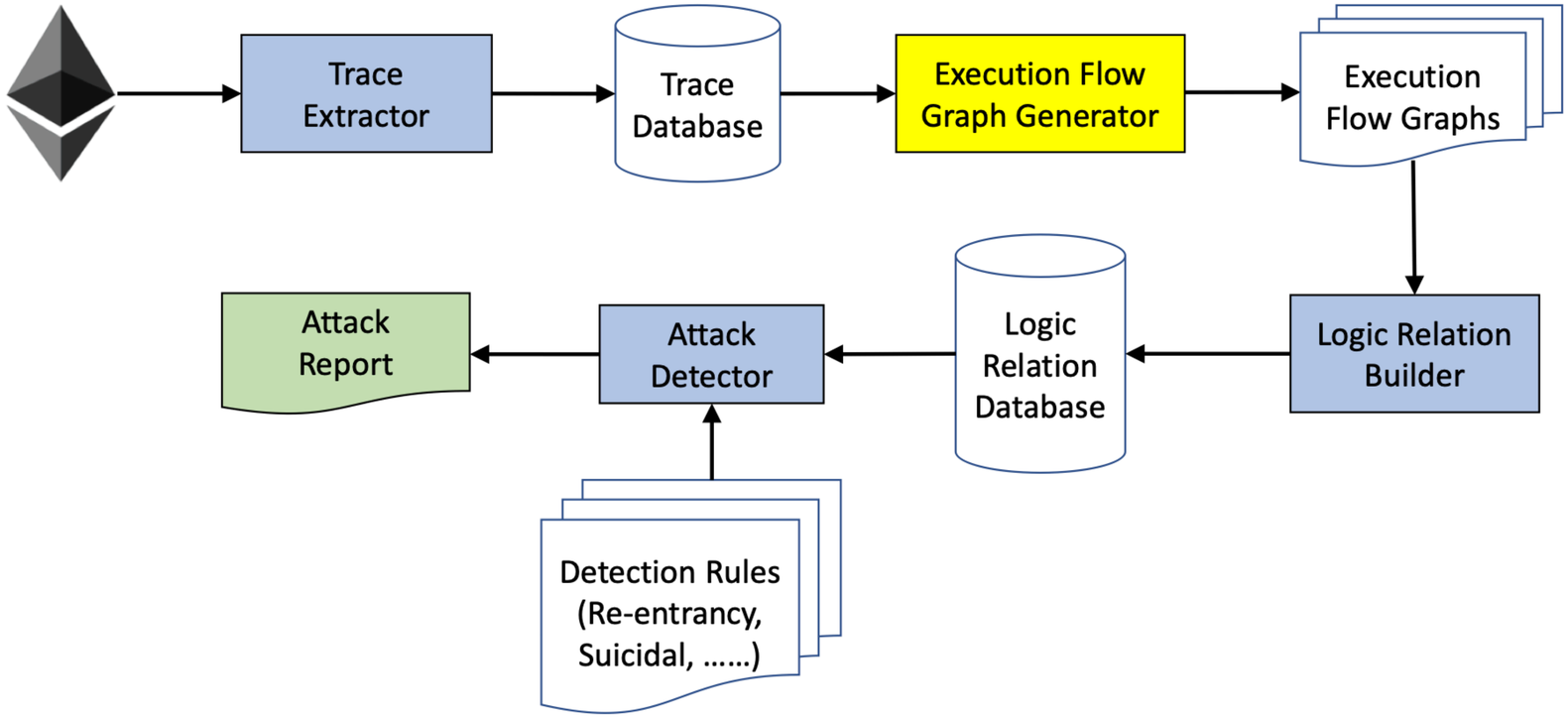


{<pc> ; <OPCODE> ; <args>}

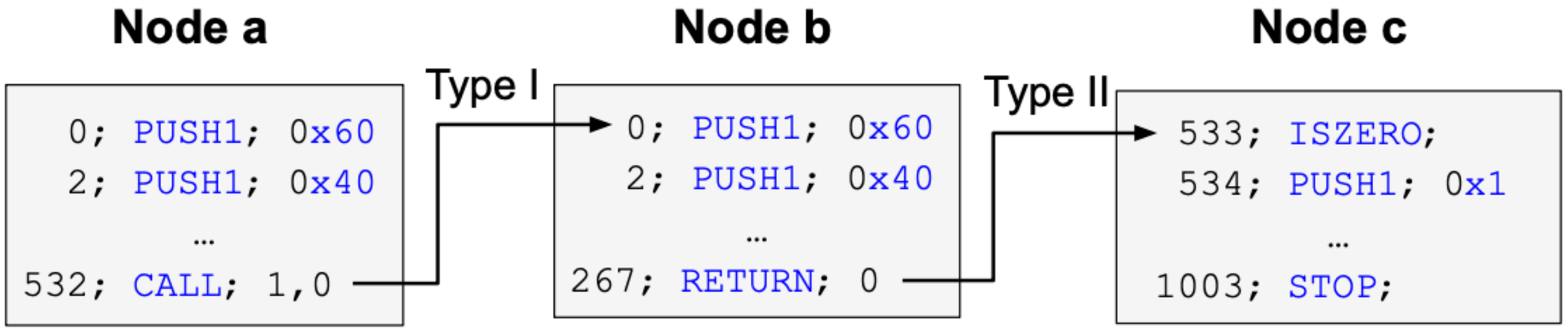
```
 0; PUSH1; 0x60
 2; PUSH1; 0x40
 4; MSTORE
 5; CALLDATASIZE; 0x144
 6; ISZERO
 7; PUSH2; 0x20e
10; JUMPI
```

Detailed Design - Execution Flow Graph Generator

- Construct the Execution Flow Graph



An example of Execution Flow Graph Generator



{<PC>; <OPCODE>; <ARGS>; **<idx>; <depth>; <callnum>**}

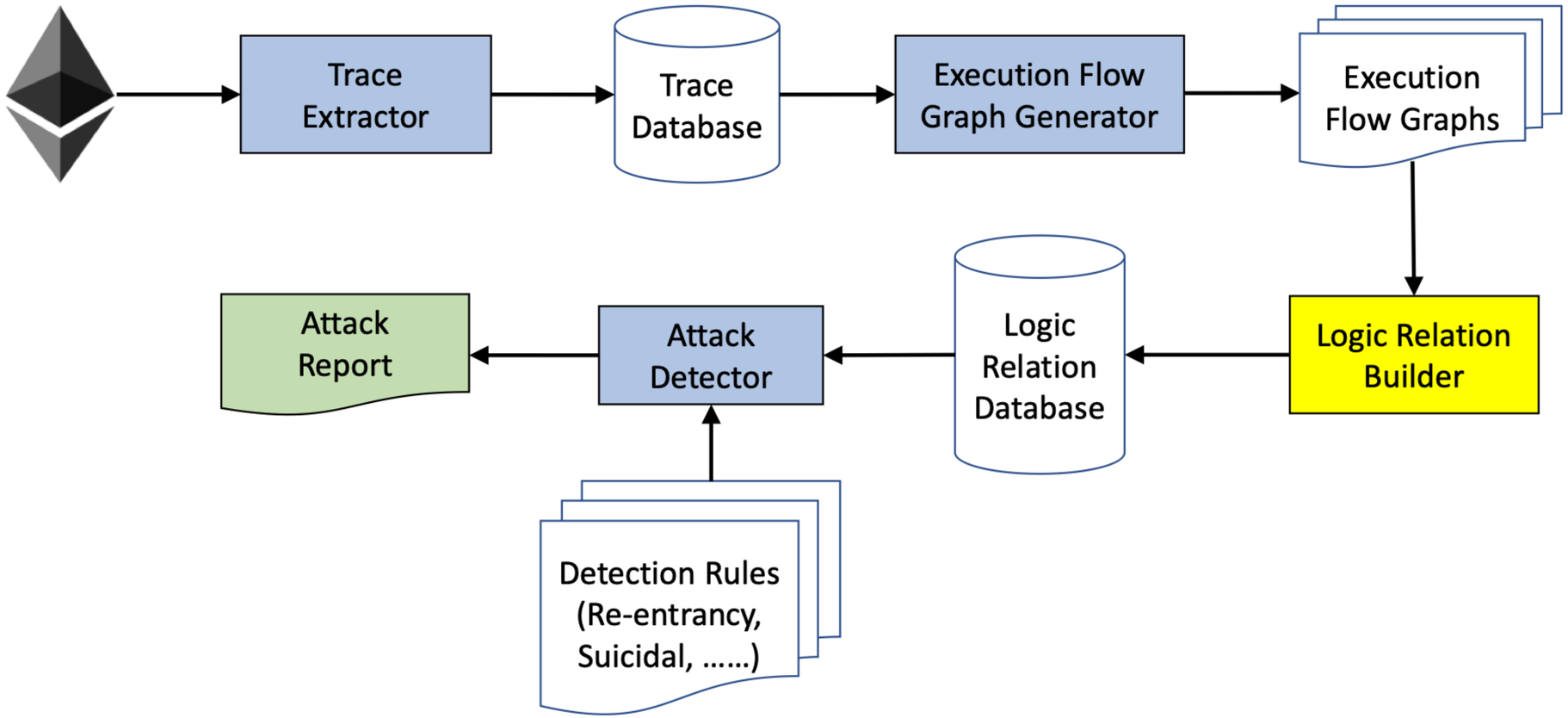
Idx: Index of the current OPCODE in the EFG, used because PC values can repeat across contracts.

Depth: Call depth level of each OPCODE; increases by 1 on a call, decreases by 1 on return.

Callnum: Counts the total number of calls before each OPCODE; increments by 1 with each new call.

Detailed Design - Logic Relation Builder

- Extract logic relations

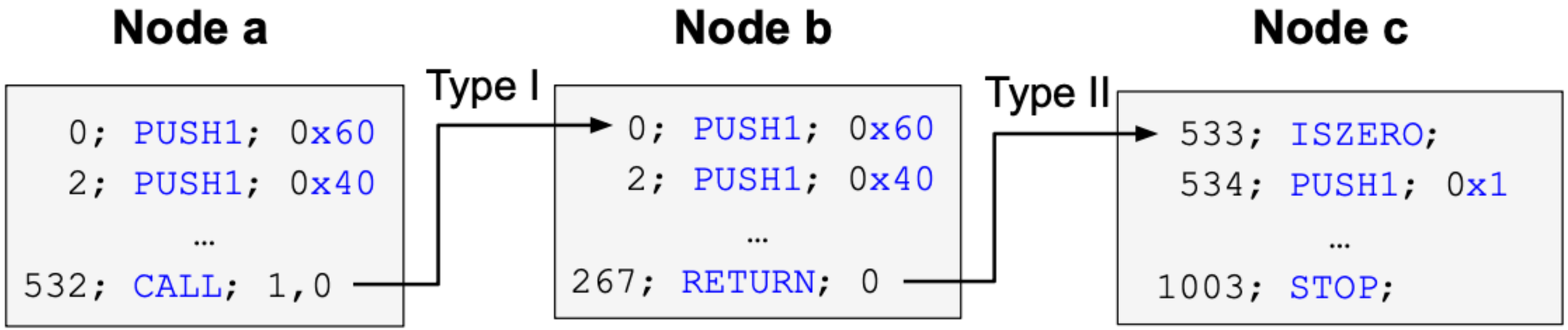


The logic rules used by Logic Relation Builder

```
.type Variable
.type Opcode
.type Value
.decl def(var:Variable, pc:number, idx:number,
 ↪   depth:number, callnum:number)
.decl use(var:Variable, pc:number, i:number, idx:number,
 ↪   depth:number, callnum:number)
.decl op(pc:number, op:Opcode, idx:number)
.decl value(var:Variable, val:Value)
.decl op_OPCODE(pc:number, registers:Variable, idx:number,
 ↪   depth:number, callnum:number)
.input def, use, op, value, op_OPCODE
```

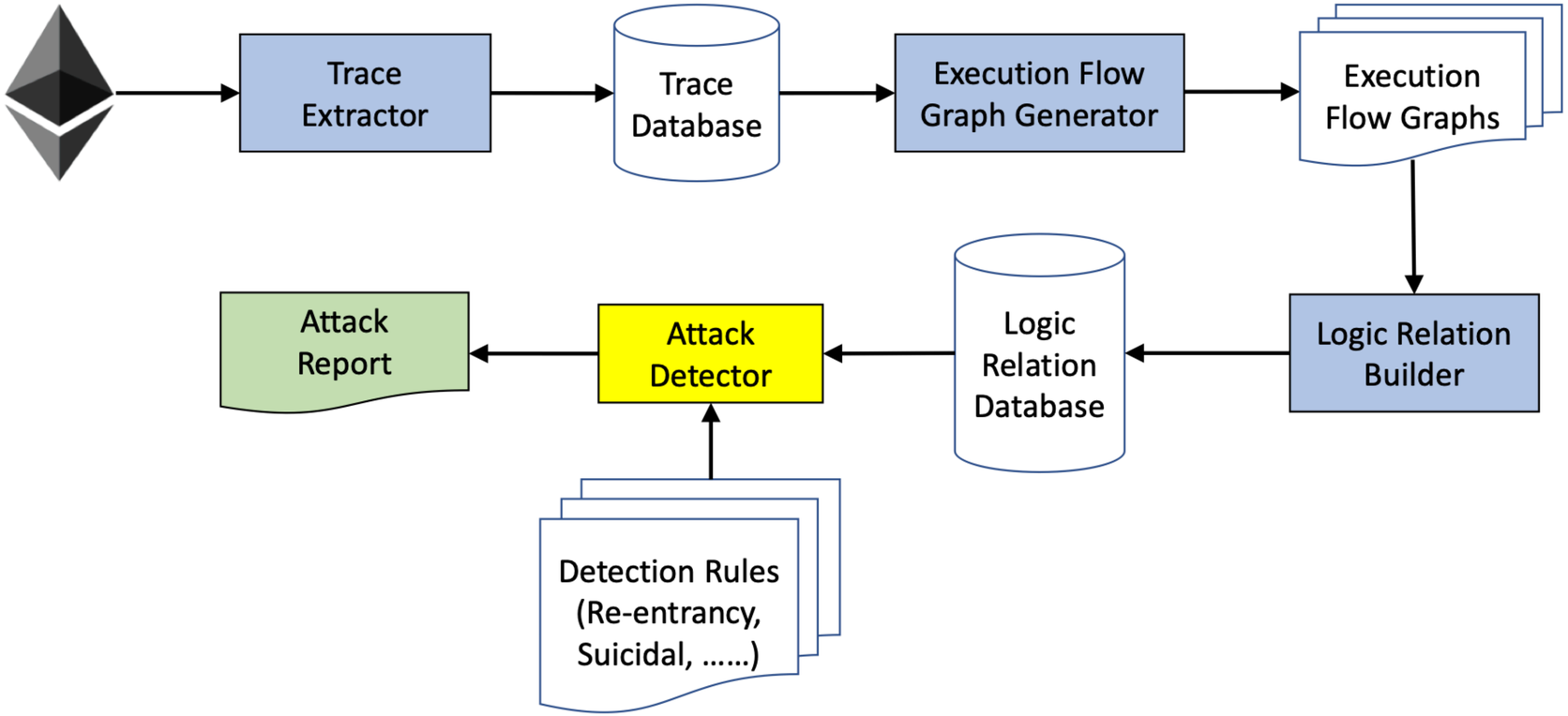
An example of PUSH1 logic relations:

op_OPCODE(**pc**:number, **registers**:Variable, **idx**:number, **depth**:number, **callnum**:number)

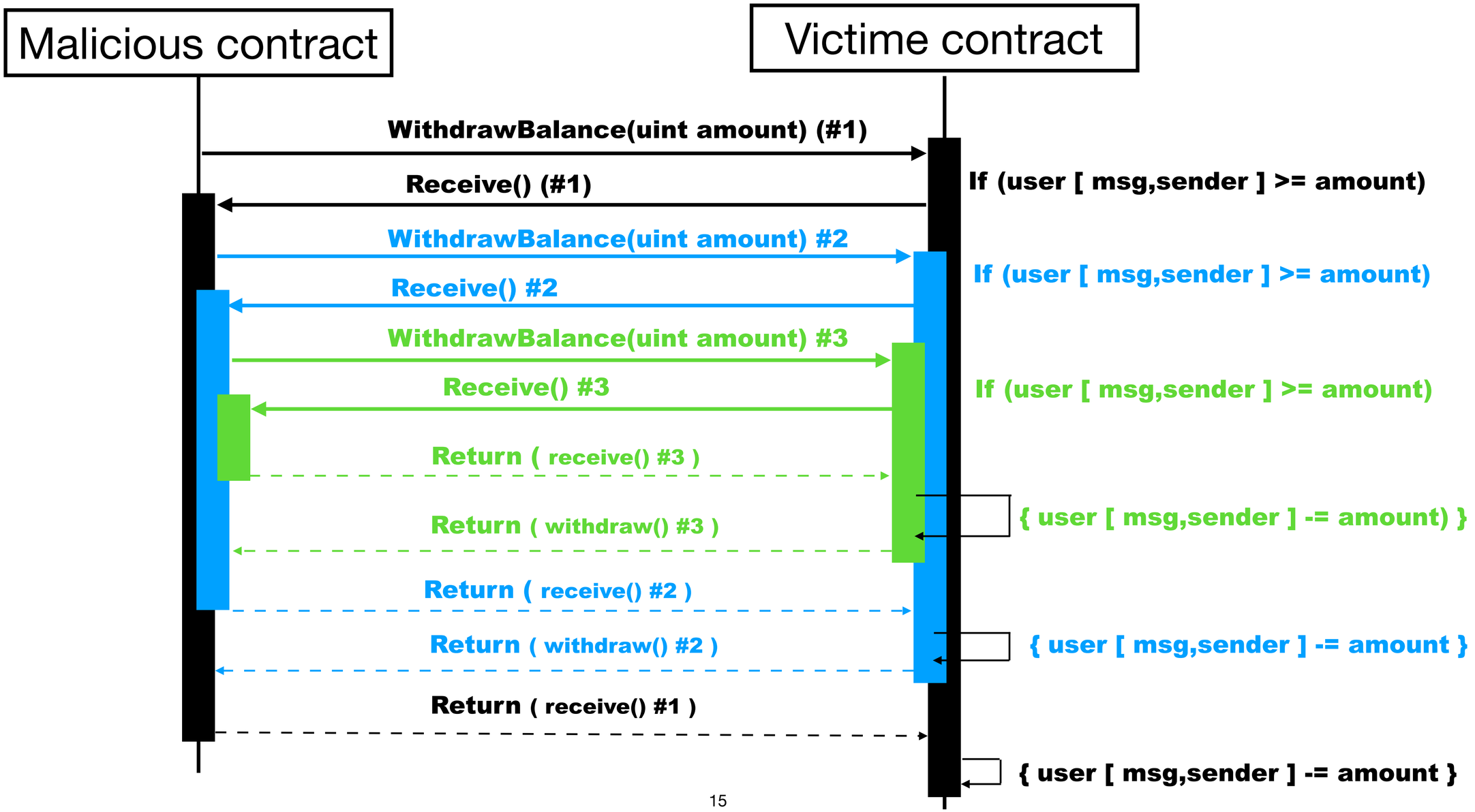


PC	Register	Idx	Depth	Callnum
0	V1	1	1	0
2	V2	2	1	0
0	V89	245	2	1
2	V90	246	2	1
534	V285	1,072	1	1

Detailed Design - Attack Detector



Single function Reentrancy Attacks - An Example



Single function Reentrancy Attacks - Detection Rules

Detection rules principes:

"If there is a state change (i.e., updates of a storage variable) after the Victim Contract is re-entered and returned, and this storage variable affects a control-flow decision when re-entering the Victim Contract, it will result in an **inconsistent state.**"

SLOAD-JUMPI Dependency

op_SLOAD (_, sloadAddr, sloadVal, sloadIdx, sloadDp, cn),

op_JUMPI (_, _, jumpiCond, _, sloadDp, cn)

depends (jumpiCond, sloadVal),

Single function Reentrancy Attacks - Detection Rules

Detection rules principes:

"If there is a state change (i.e., updates of a storage variable) after the Victim Contract is re-entered and returned, and this storage variable affects a control-flow decision when re-entering the Victim Contract, it will result in an **inconsistent state**."

SLOAD-SSTORE Dependency

op_SSTORE (_, sstoreAddr, _, sstoreIdx, sstoreDp, _),

filterByDepth (sloadDp, sstoreDp), sloadDp > sstoreDp

filterByIdx (sloadIdx, sstoreIdx), sloadIdx < sstoteIdx

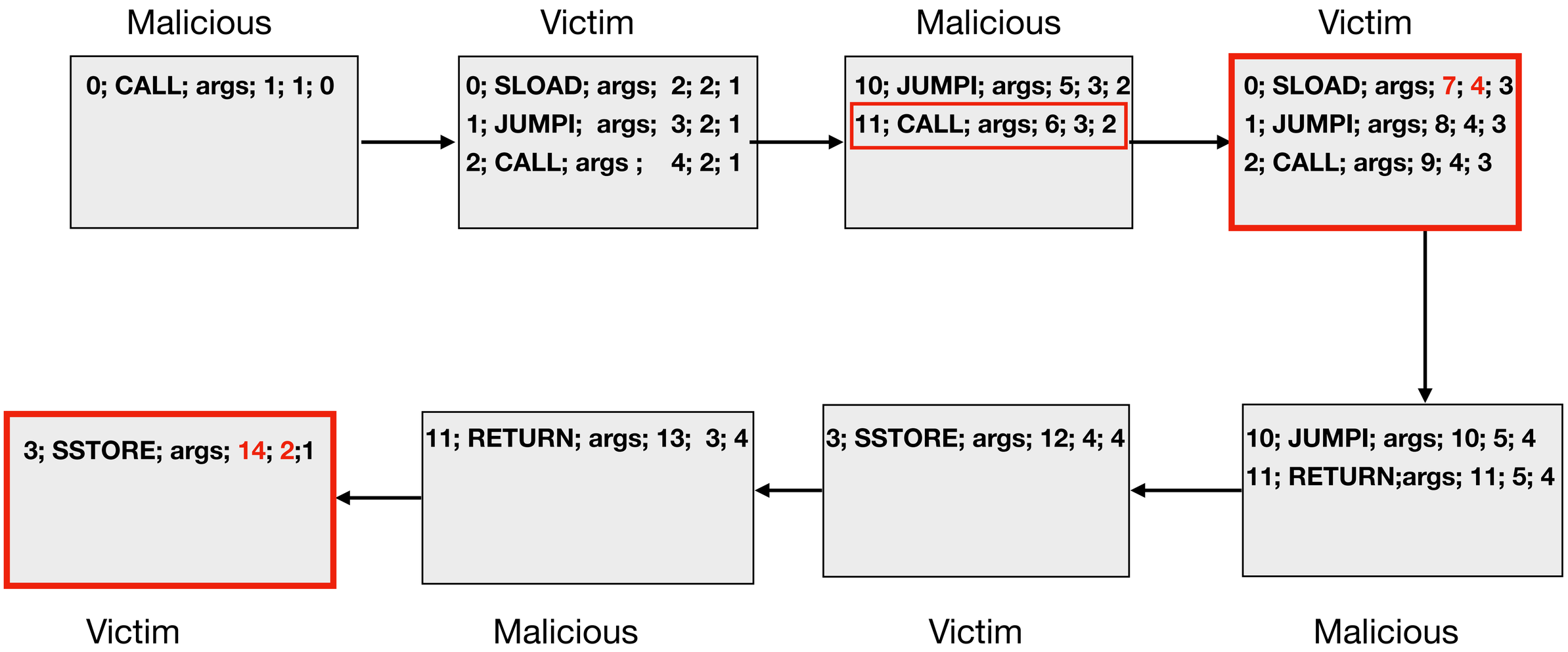
checkSameAddr (sloadAddr, sstoreAddr),

checkSameContract (sloadAddr, jumpiCond, sstoreAddr)

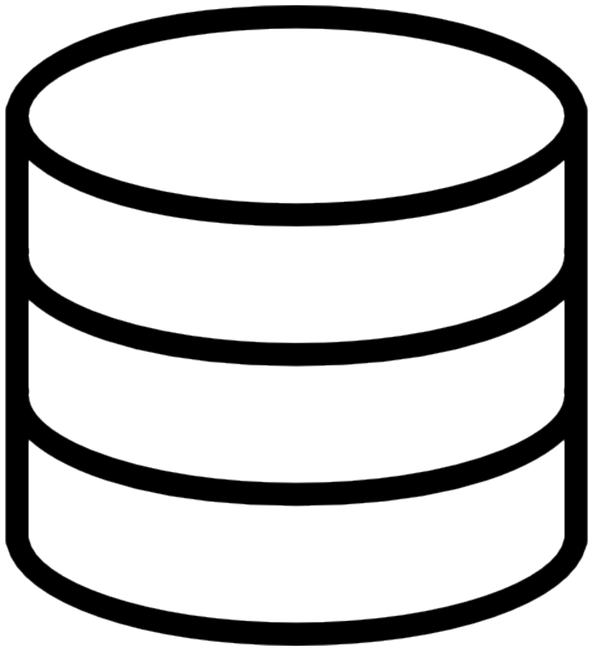
Single function Reentrancy Attacks - 1 Inconsistent state

An Example:

Execution flow graph: {<PC>; <OPCODE>; <ARGS>; **<idx>; <depth>; <callnum>**}



Experiment Setup



January 2019 - February 2019

16,485,279 Transactions

9,321,684 Transactions

3357 Transactions

30 Vulnerable smart contracts

TxSpector: Strengths and Limitations

Strengths

- Supports custom Detection Rules.
- Performs forensic-level analysis.
- Captures dynamic behavior during real execution.
- Open-source and actively extensible.
- Generic framework.

Limitations

- Only detects single-function re-entrancy.
- Requires a full Ethereum node (resource-intensive).
- New EVM opcodes must be added manually.
- Detection results are not easy to understand.
- False Positives: TxSpector fails to detect locking mechanisms that prevent unauthorized reentry.

Extending TxSpector: Work Done and What's Ahead

Main Goal

- Use TxSpector to label smart contracts with re-entrancy vulnerabilities.

Work Done

- Studied and deployed the original TxSpector tool.
- Fixed compatibility issues with current full-node setups.
- Added missing EVM opcodes.
- Tested the tool on real Ethereum transactions.

Extending TxSpector: Work Done and What's Ahead

🚀 Ongoing & Future Objectives

- Extend logic rules to cover more types of vulnerabilities.
- Improve the clarity and context of detection results for better analysis.
- Optimize the Trace Extractor — explore new methods that don't require syncing the entire blockchain.
- Automate the process between components using RPA tools like RPA UIPATH.

🙏 Thank you for your attention!

📧 Contact: semia.guesmi@unicam.it

📄 Original Paper: TxSpecTor: Uncovering Attacks in Ethereum from Transactions

📦 Original Repository: TxSpector github repository

🔧 Our Revised Version: revised TxSpector github repository

References

[1] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security ACM, 2016.

[2] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, M. Vechev, Securify: Practical security analysis of smart contracts, in: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, 2018, pp. 67–82.

[3] L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz, "Vandal: A scalable security analysis framework for smart contracts," arXiv preprint arXiv:1809.03981, 2018.

[4] Z. Liao, Z. Zheng, X. Chen, Y. Nan, Smartdagger: a bytecode-based static analysis approach for detecting cross-contract vulnerability, in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022, pp. 752–764.

[5] B. Mueller, Smashing smart contracts, in: 9th HITB Security Conference, 2018.

[6] C. Schneidewind, I. Grishchenko, M. Scherer, M. Maffei, ethor: Practical and provably sound static analysis of ethereum smart contracts, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 621–640.

[7] M.Rodler,W.Li,G.Karame,andL.Davi,"Sereum:Protectingexisting smart contracts against re-entrancy attacks," in Proceedings of the 26th Network and Distributed System Security Symposium, 2019

[8] S.Grossman,I.Abraham,G.Golan-Gueta,Y.Michalevsky,N.Rinet- zky, M. Sagiv, and Y. Zohar, "Online detection of effectively callback free objects with applications to smart contracts," Proceedings of the ACM on Programming Languages, 2017.

[9] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2020. TXSPECTOR: Uncovering attacks in Ethereum from transactions. In 29th USENIX Security Symposium (USENIX Security'20). 2775–2792.