Silence False Alarms: Identifying Anti-Reentrancy Patterns on Ethereum to Refine Smart Contract Reentrancy Detection

Qiyang Song, Heqing Huang, Xiaoqi Jia, Yuanbo Xie, and Jiahao Cao

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

CHINESE ACADEMY OF SCIENCES

清華大學
Tsinghua University

NDSS
SYMPOSIUM/2025

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain

  - Since DAO hack (2016)

    $50+ million stolen

    How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

    As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

    Millions of assets stolen

    How Spankchain Got Hacked

    Explained: A Reentrancy attack which drained 165 Ether

    Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

    By CISOMAG - April 20, 2020

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen

    How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

    As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.
  - SpankChain and Lendf.me

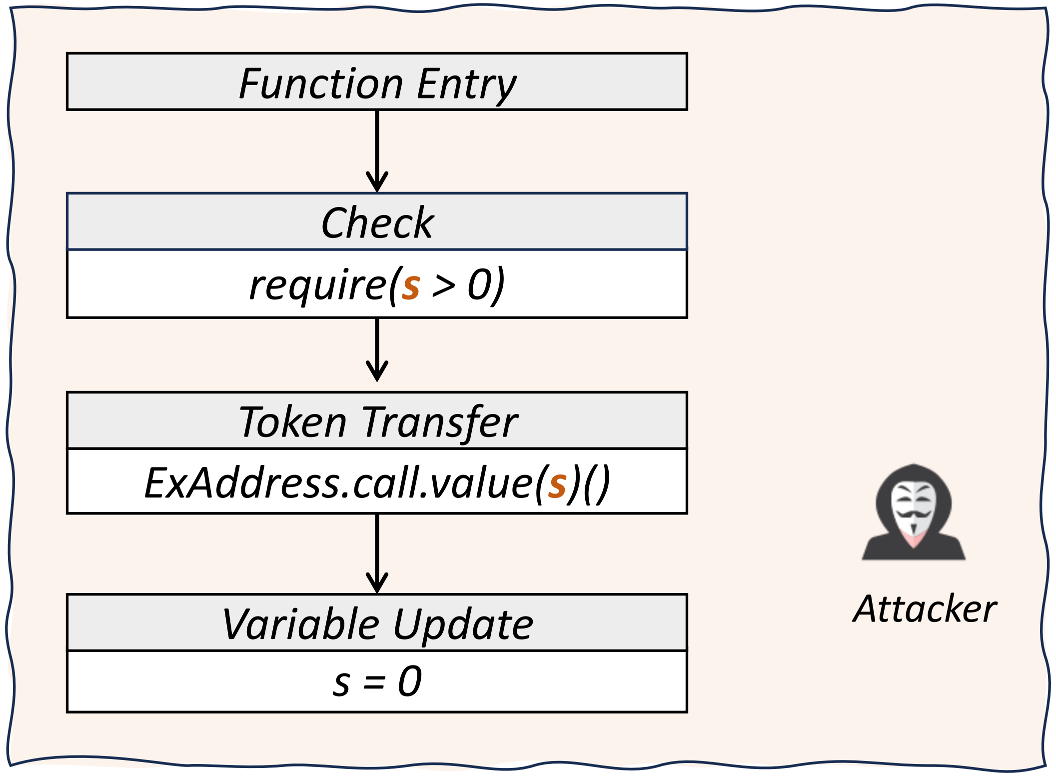
    Millions of assets stolen

    How Spankchain Got Hacked

    Explained: A Reentrancy attack which drained 165 Ether

    Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

    By CISOMAG - April 20, 2020

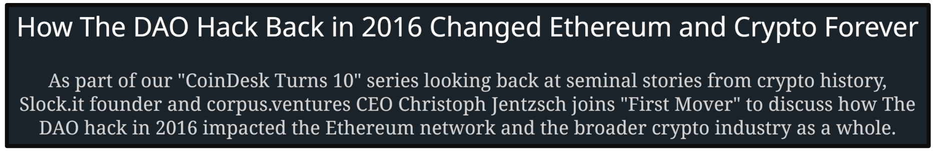


Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen



How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

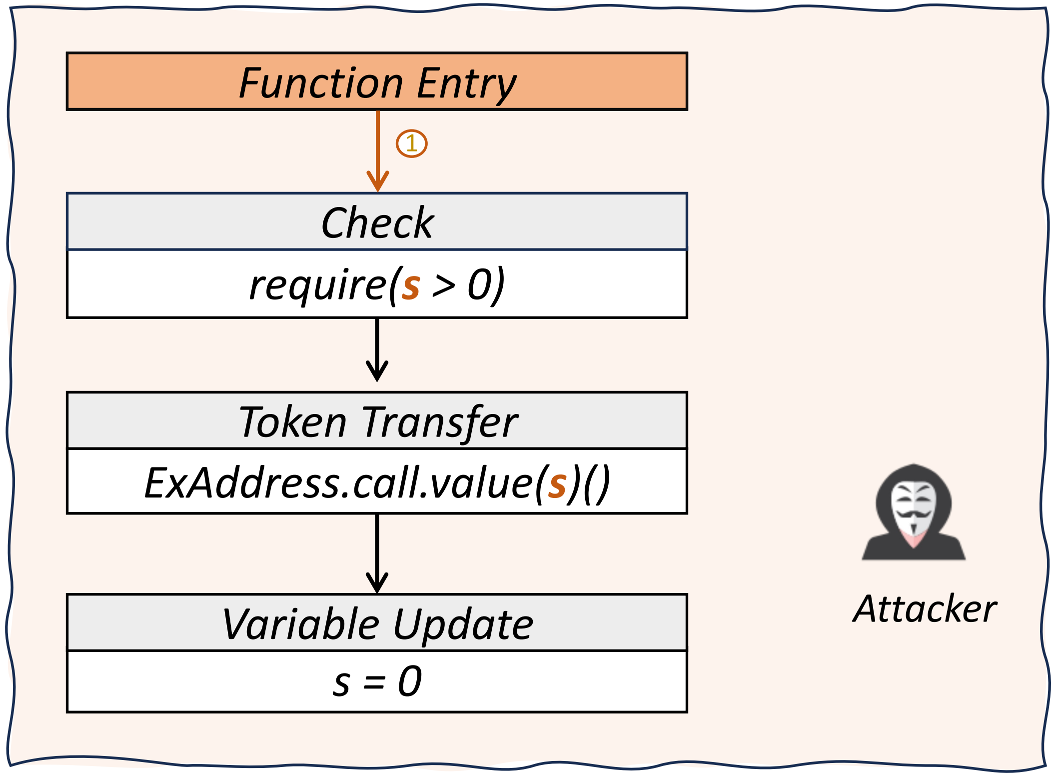
    Millions of assets stolen

How Spankchain Got Hacked

Explained: A Reentrancy attack which drained 165 Ether

Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

By CISOMAG - April 20, 2020



Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen

How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

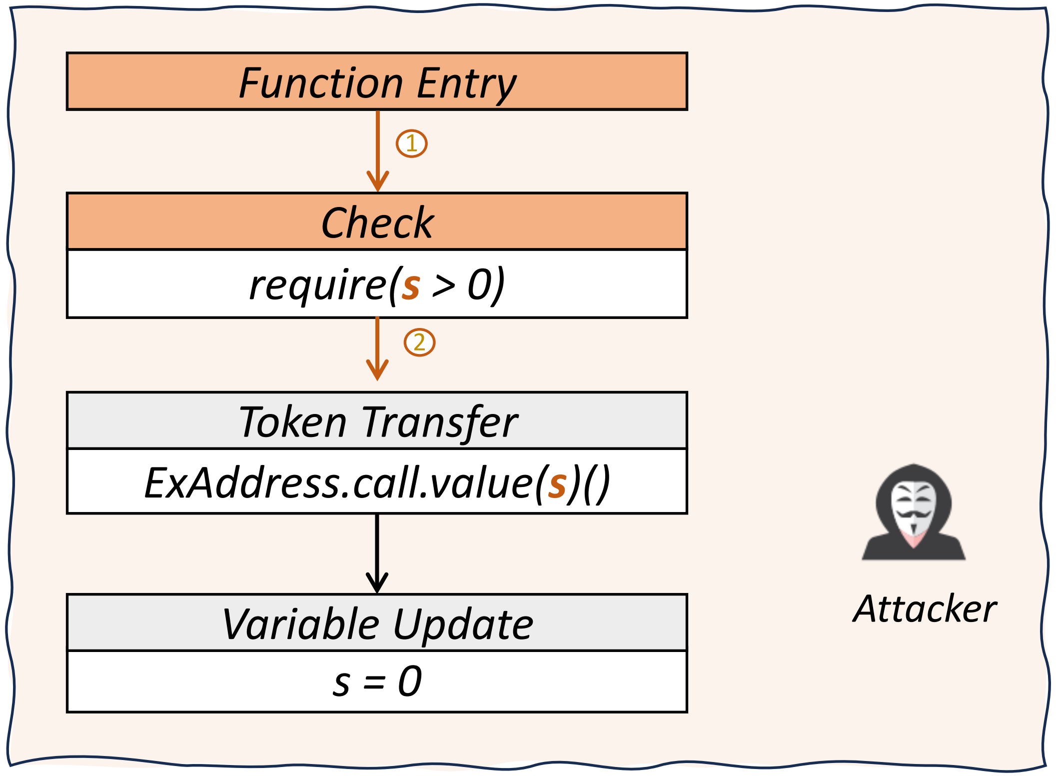
    Millions of assets stolen

How Spankchain Got Hacked

Explained: A Reentrancy attack which drained 165 Ether

Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

By CISOMAG - April 20, 2020



Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen

How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

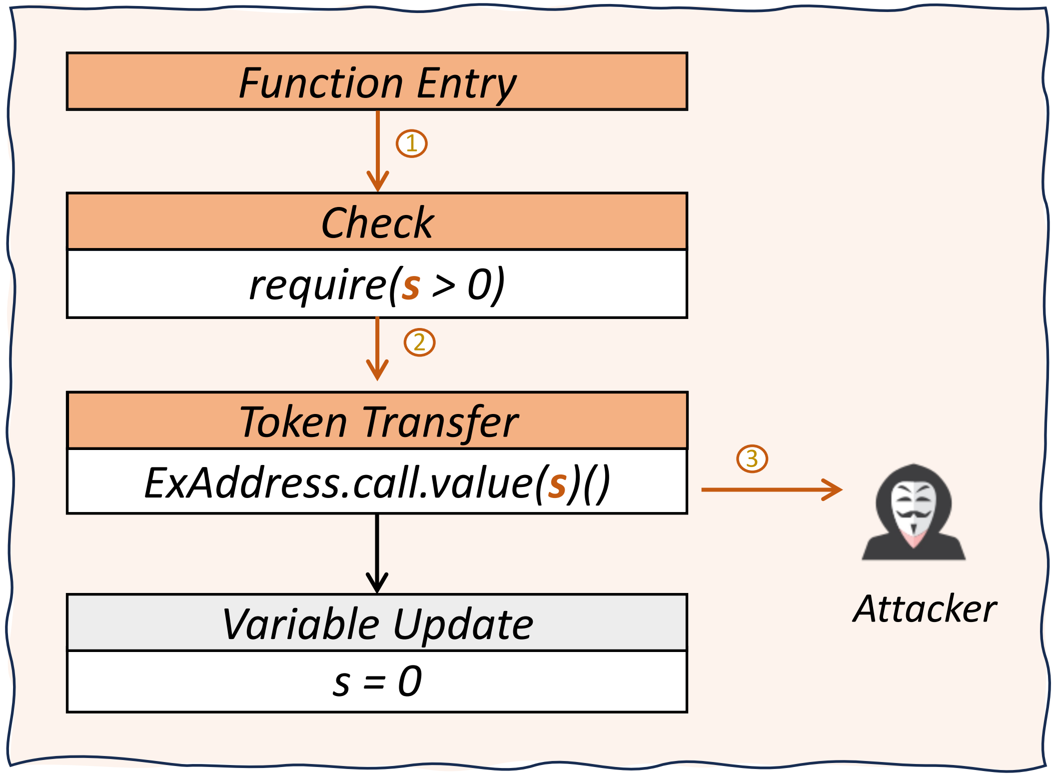
    Millions of assets stolen

How Spankchain Got Hacked

Explained: A Reentrancy attack which drained 165 Ether

Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

By CISOMAG - April 20, 2020



Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen

    How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

    As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

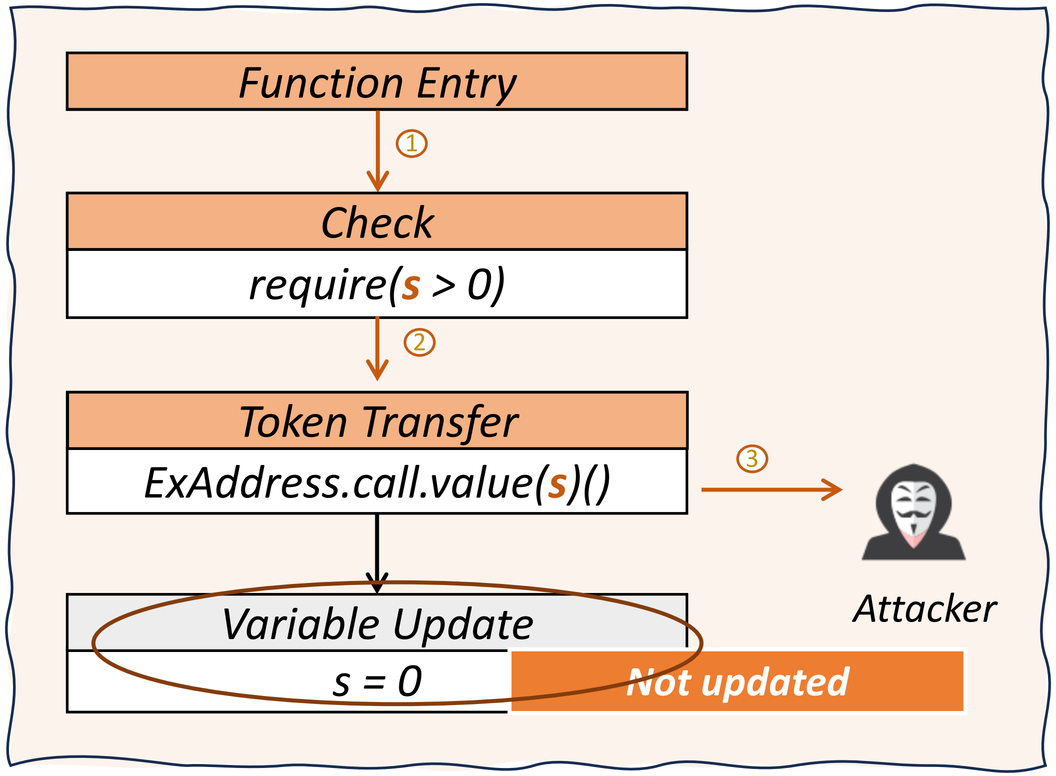
    Millions of assets stolen

    How Spankchain Got Hacked

    Explained: A Reentrancy attack which drained 165 Ether

    Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

    By CISOMAG - April 20, 2020



Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen

How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

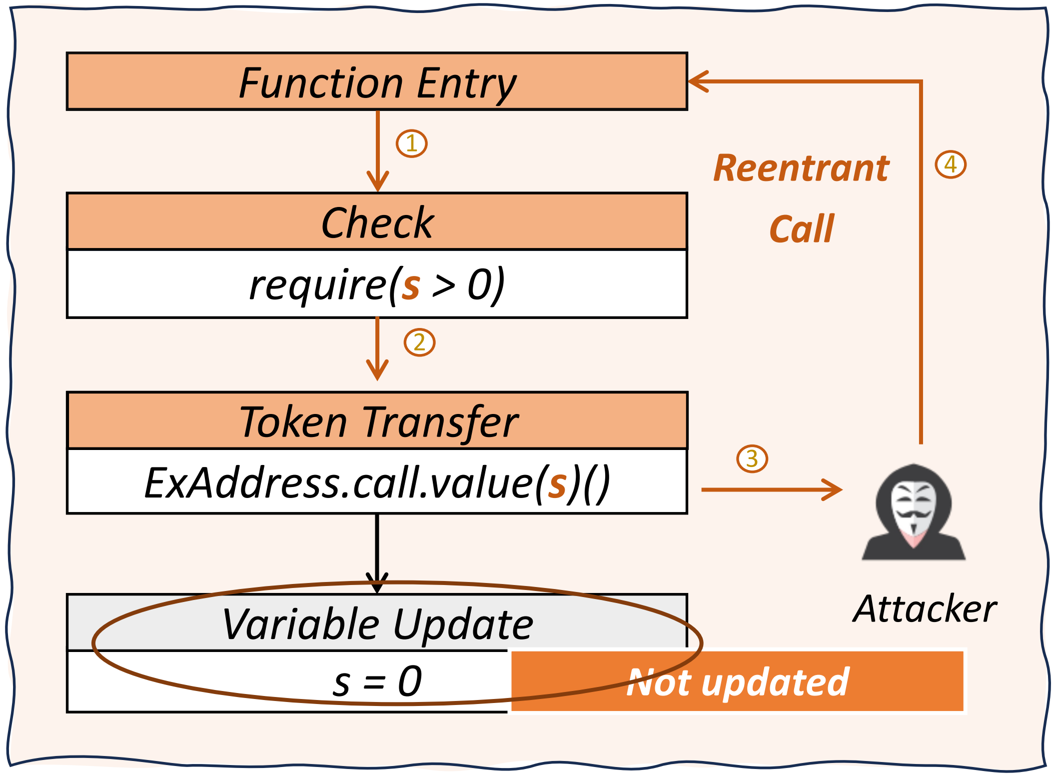
    Millions of assets stolen

How Spankchain Got Hacked

Explained: A Reentrancy attack which drained 165 Ether

Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

By CISOMAG - April 20, 2020



Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen

    How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

    As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

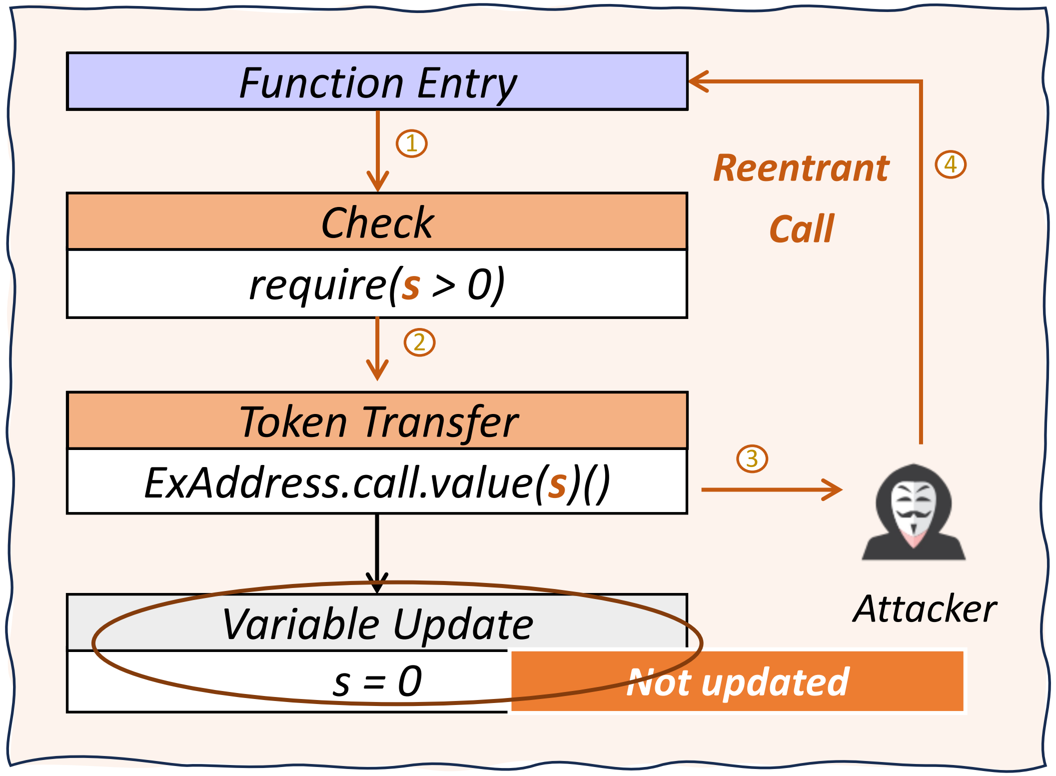
    Millions of assets stolen

    How Spankchain Got Hacked

    Explained: A Reentrancy attack which drained 165 Ether

    Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

    By CISOMAG - April 20, 2020



Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen

    How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

    As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

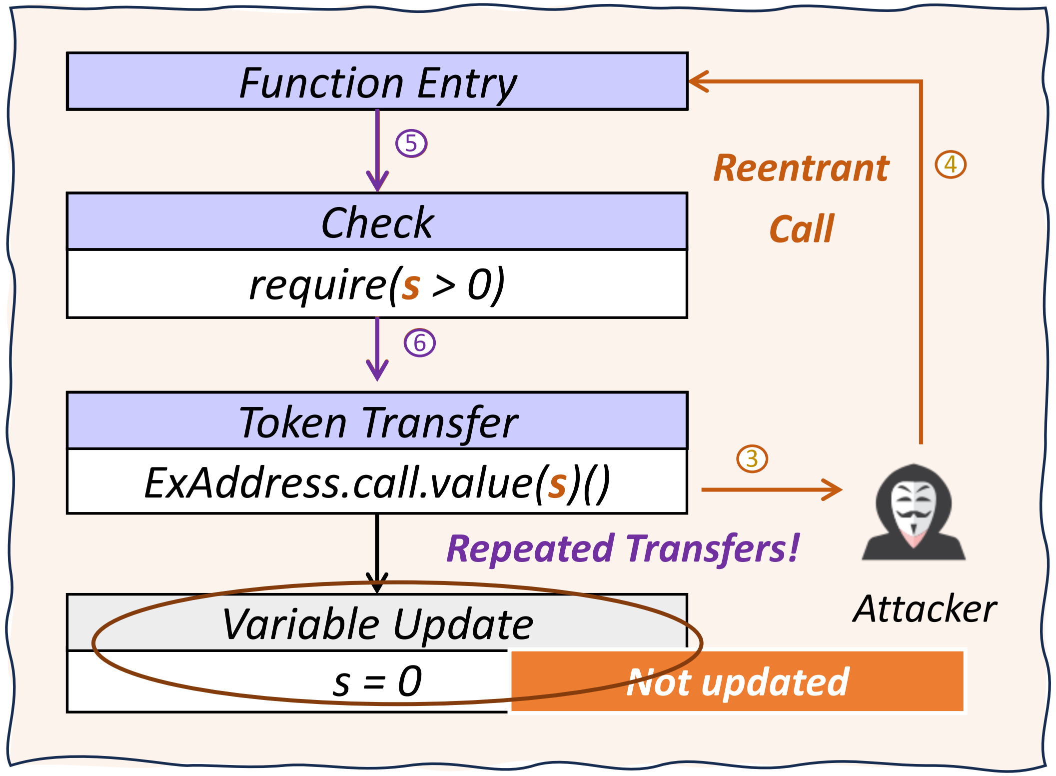
    Millions of assets stolen

    How Spankchain Got Hacked

    Explained: A Reentrancy attack which drained 165 Ether

    Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

    By CISOMAG - April 20, 2020

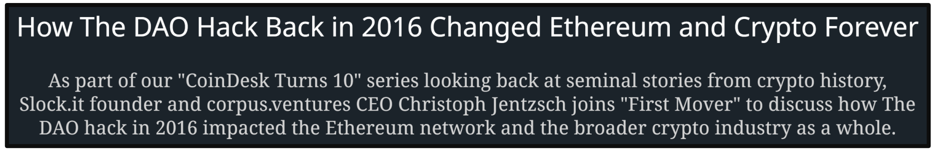


Reentrancy Attack Example

Infamous Smart Contract Bug: Reentrancy

- Reentrancy bugs have caused massive financial losses on the blockchain
- They enable external attackers to **manipulate program execution**
  - Since DAO hack (2016)

    $50+ million stolen



How The DAO Hack Back in 2016 Changed Ethereum and Crypto Forever

As part of our "CoinDesk Turns 10" series looking back at seminal stories from crypto history, Slock.it founder and corpus.ventures CEO Christoph Jentzsch joins "First Mover" to discuss how The DAO hack in 2016 impacted the Ethereum network and the broader crypto industry as a whole.

  - SpankChain and Lendf.me

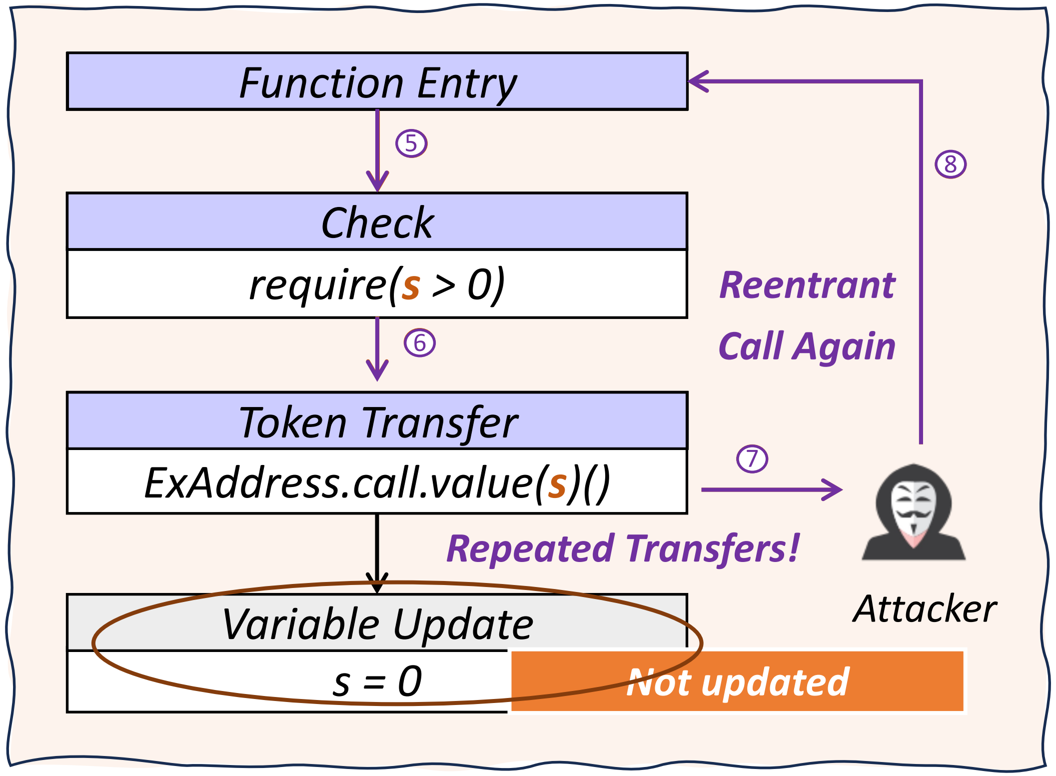
    Millions of assets stolen

How Spankchain Got Hacked

Explained: A Reentrancy attack which drained 165 Ether

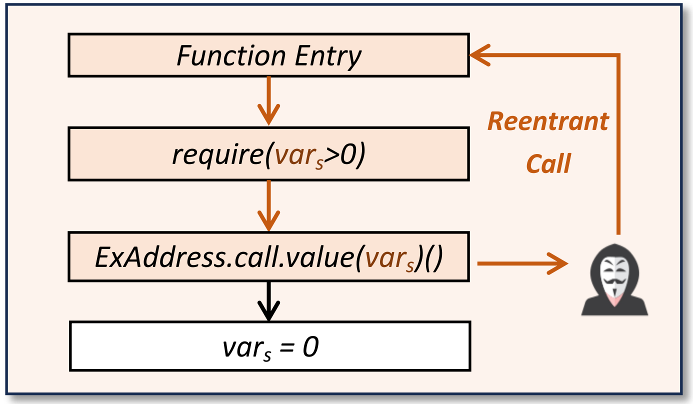
Cryptocurrency Worth $25 Mn Stolen in Lendf.Me and Uniswap Hacking

By CISOMAG - April 20, 2020



Reentrancy Attack Example

Reentrancy Vulnerability Detection



- Existing Reentrancy Detectors
  - According to Basic Reentrancy Patterns

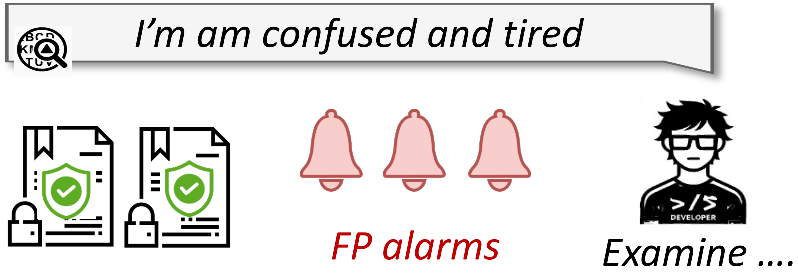
Read Variable X --> External Call --> Write Variable X

  - Based on static analysis/symbolic execution

SLITHER | Mythril | SECURIFY 2.0 new!

2022 IEEE Symposium on Security and Privacy (SP)
SAILFISH: Vetting Smart Contract State-Inconsistency Bugs in Seconds

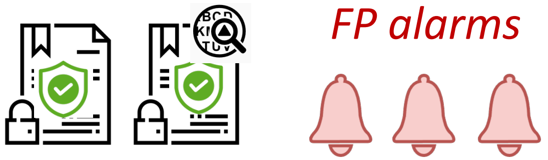
  - **A high rate of false positives (FPs)**
    - Leads to alert fatigue



*Durieux, T et al. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. ICSE'20

False Alarms Caused by Anti-reentrancy Patterns

- Exiting tools ignore **anti-reentrancy patterns**[*]
  - **FPs:** misclassify safe contracts as vulnerable

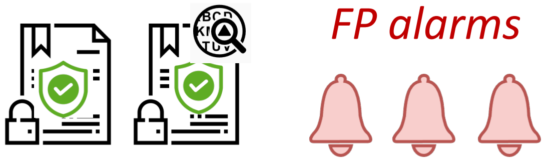


Ignore anti-reentrancy patterns

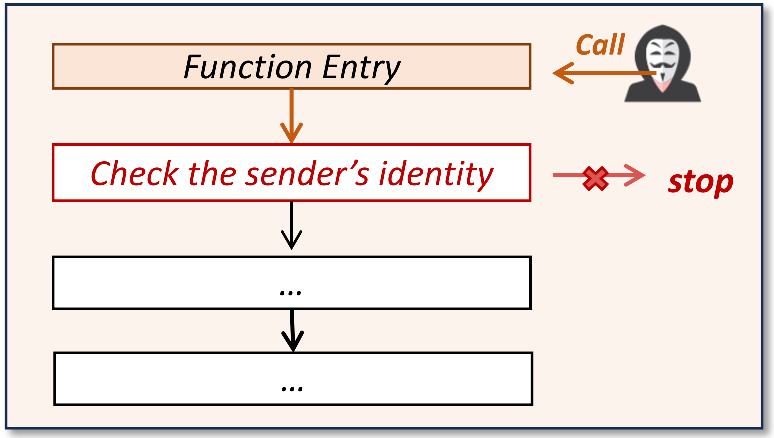
[*] Xue, Y. et al. Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. ASE 2020.

False Alarms Caused by Anti-reentrancy Patterns

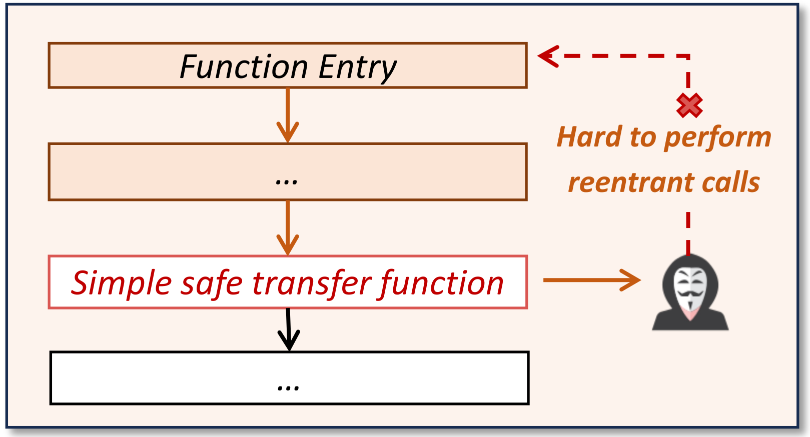
- Exiting tools ignore **anti-reentrancy patterns***
  - **FPs:** misclassify safe contracts as vulnerable
  - Anti-reentrancy patterns prevent illegal users from reentering functions to gain profits



Ignore anti-reentrancy patterns



Example 1: sender check



Example 2: Safe transfer

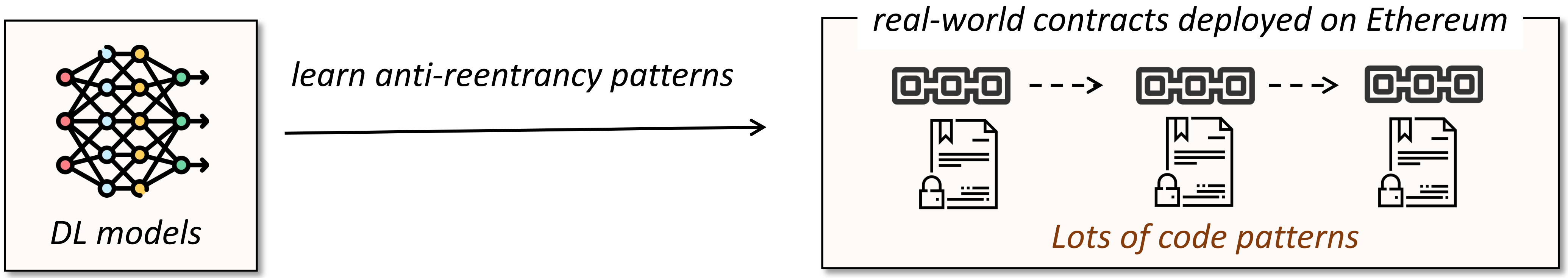
* Xue, Y. et al. Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. ASE 2020.

To reduce false positives, we

develop an automated tool to
identify anti-reentrancy patterns

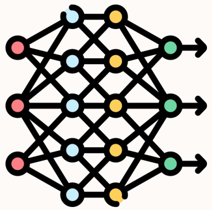
Our Solution

- Use deep learning to learn anti-reentrancy patterns from various contracts



Our Solution

- Use deep learning to learn anti-reentrancy patterns from various contracts



DL models

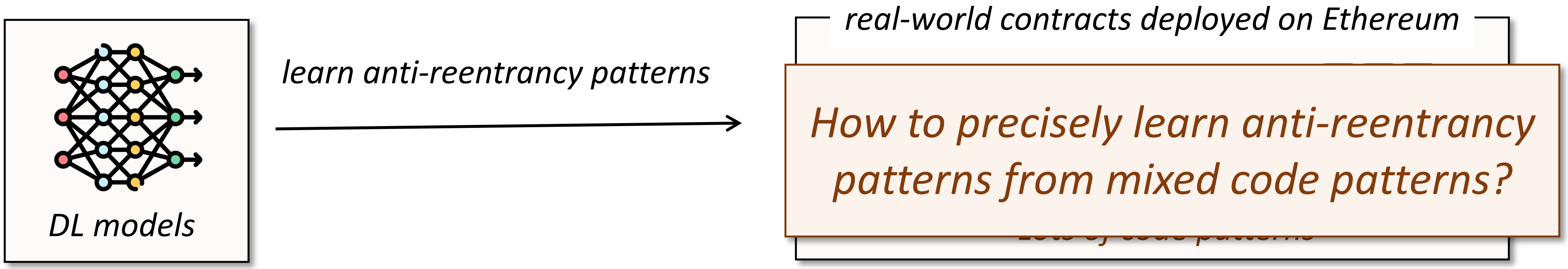
learn anti-reentrancy patterns

real-world contracts deployed on Ethereum

How to precisely learn anti-reentrancy patterns from mixed code patterns?

Our Solution

- Use deep learning to learn anti-reentrancy patterns from various contracts



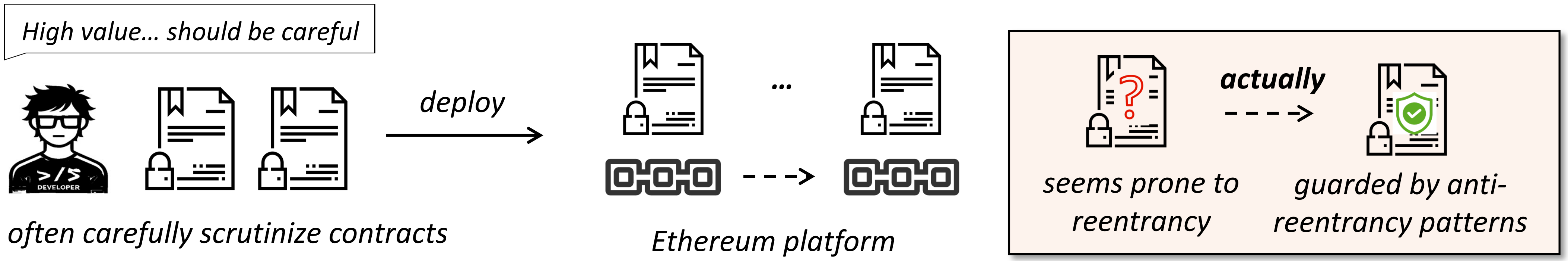
- Design specific methods and data structures to capture related semantics

Step #1:

We begin by filtering contracts **potentially with anti-reentrancy patterns**

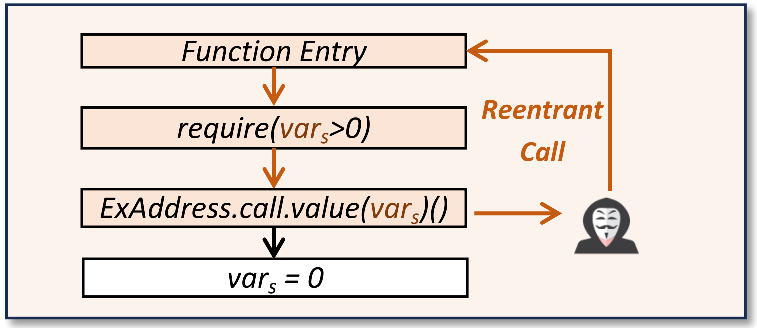
Smart Contract Filtering

- Insight: Ethereum contracts prone to reentrancy often **contain anti-reentrancy patterns***



- Utilize reentrancy knowledge to identify related smart contracts
  - Static analysis

Read Variable X --> External Call --> Write Variable X



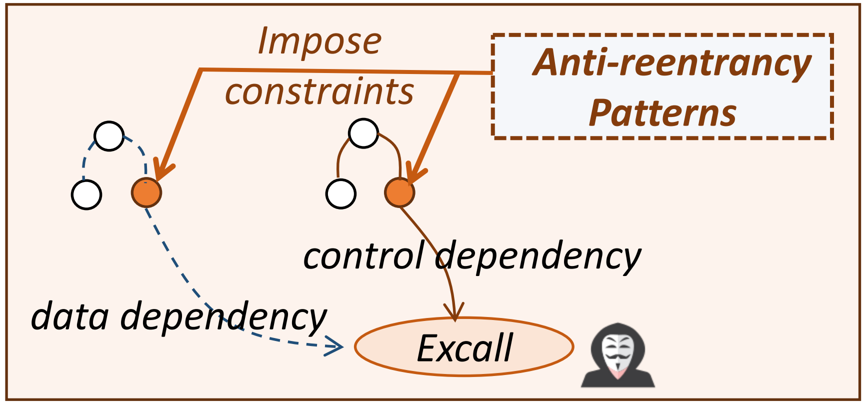
* *According to our investigation and related paper* "Xue, Y. et al. Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. ASE 2020."

Step #2:

Design a data structure to further capture anti-reentrancy semantics from selected contracts

Program Dependency Graph for Anti-reentrancy (RentPDG)

- Observation: anti-reentrancy patterns often impose **data and control dependency constraints** on external calls

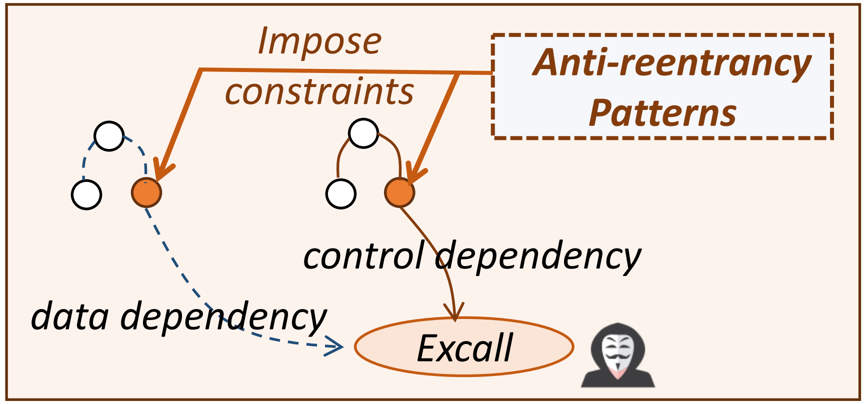


General anti-reentrancy semantics

Program Dependency Graph for Anti-reentrancy (RentPDG)

- Observation: anti-reentrancy patterns often impose **data and control dependency constraints** on external calls

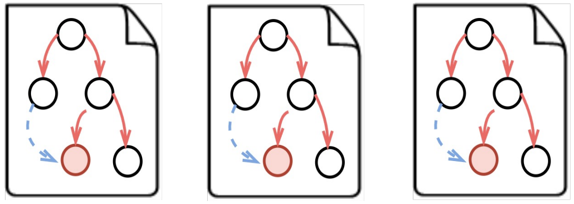
- To capture the semantics, we use program dependency graphs



General anti-reentrancy semantics

To capture the semantics

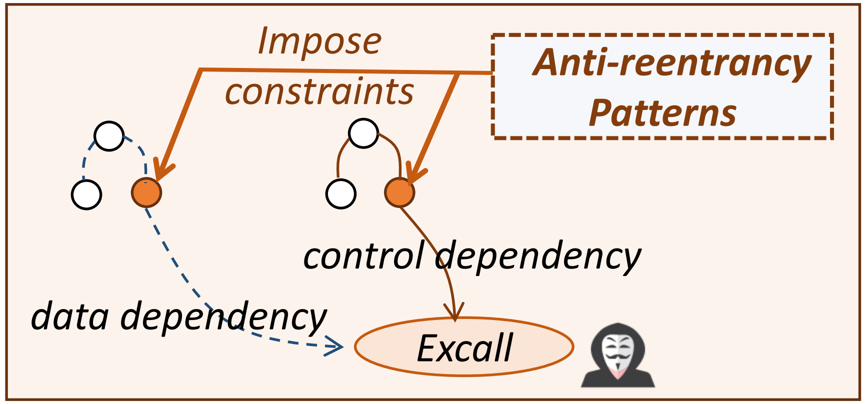
Program dependency graphs (PDG)



Reveal Control and Data dependency

Program Dependency Graph for Anti-reentrancy (RentPDG)

- Observation: anti-reentrancy patterns often impose **data and control dependency constraints** on external calls
- To capture the semantics, we use program dependency graphs



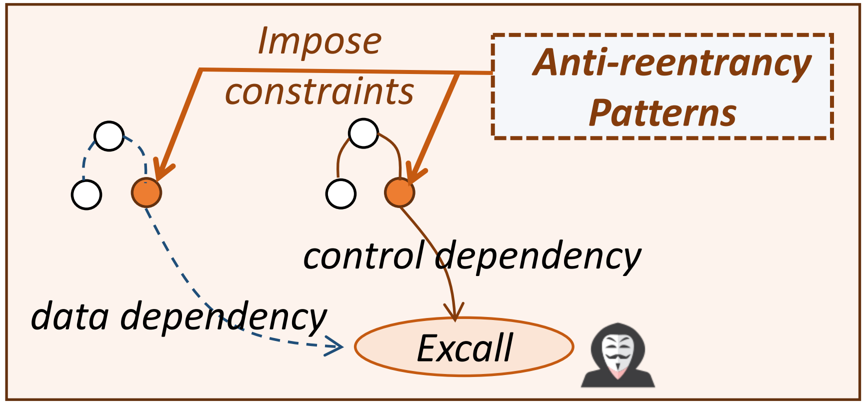
General anti-reentrancy semantics

Reveal Control and Data dependency

Program Dependency Graph for Anti-reentrancy (RentPDG)

- Observation: anti-reentrancy patterns often impose **data and control dependency constraints** on external calls
- To capture the semantics, we use ~~program dependency graphs~~

A variant of program dependency graphs (RentPDG)

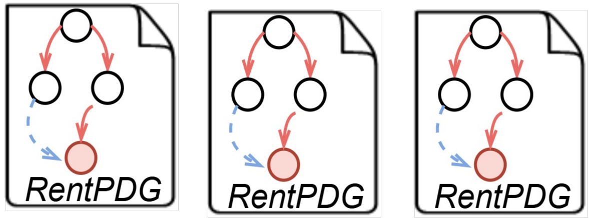


General anti-reentrancy semantics

To capture the semantics

Variant of Program Dependency Graph

(our RentPDG)



Only preserve components

related to external calls

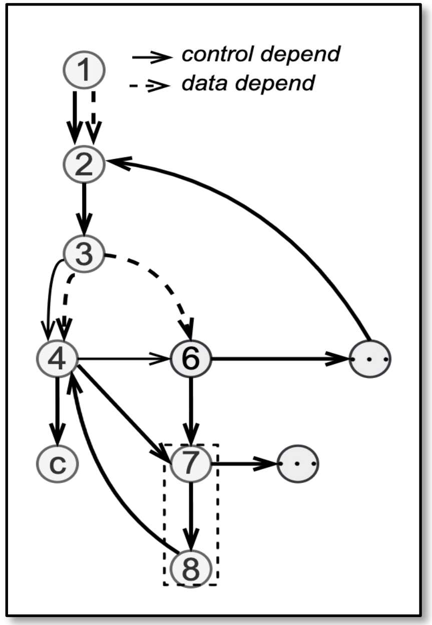
Constructing RentPDGs from Smart Contracts

- Intuitive RentPDG construction

Smart Contract Code

```
1:  function transfer(address to,address[] tokenId){
2:        _transfer(msg.sender,to,tokenId));
}

3:  function _transfer(address from,address to,address[] tokenId){
4:        require(_approve(from,to,tokenId[0]));
c:        marketingAddr.call.value(fee)();     // external call
6:        require(_approve(from, to, tokenId[1])); ...;
      }


7:  function _approve(from, to, tokenId) returns (bool){ ...;
8:        return true;
      }
```

Inter-procedural PDG



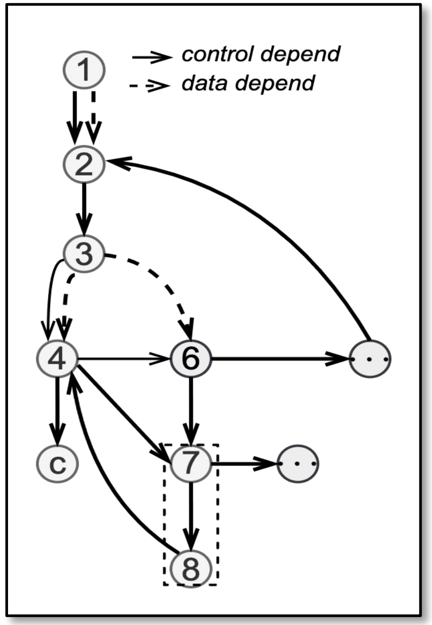
Constructing RentPDGs from Smart Contracts

- Intuitive RentPDG construction
  - use deep-first search (DFS) to extract external-call related PDG components

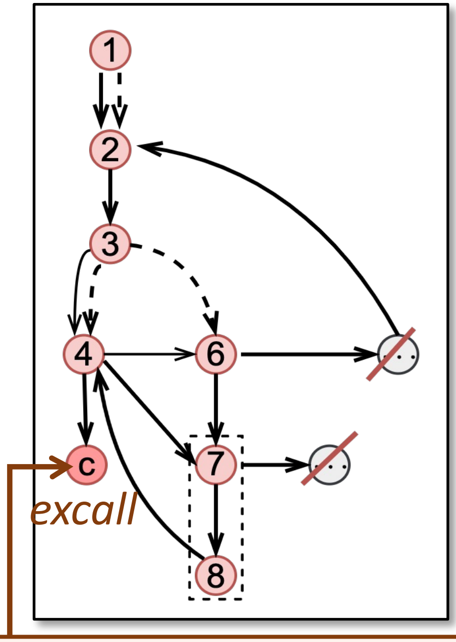
Smart Contract Code

```
1:  function transfer(address to,address[] tokenId){
2:         _transfer(msg.sender,to,tokenId));
}

3:  function _transfer(address from,address to,address[] tokenId){
4:         require(_approve(from,to,tokenId[0]));
c:         marketingAddr.call.value(fee)();     // external call
6:         require(_approve(from, to, tokenId[1])); ...;
    }


7:  function _approve(from, to, tokenId) returns (bool){ ...;
8:         return true;
    }
```

Inter-procedural PDG



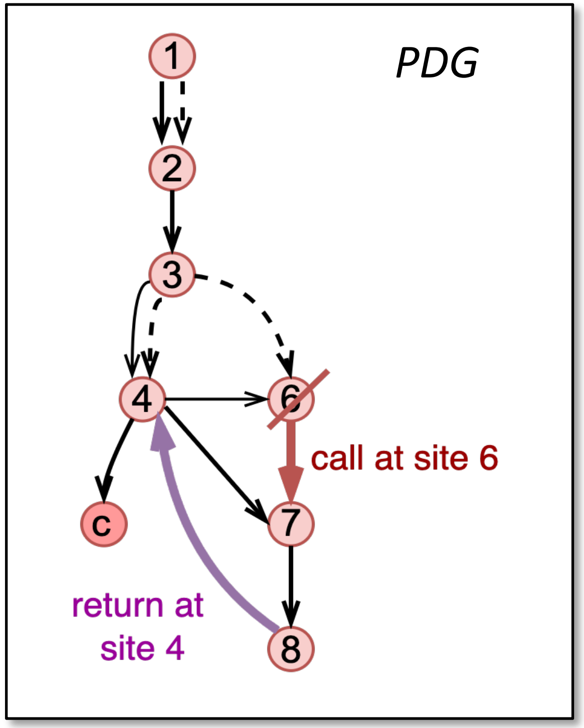
RentPDG



Use DFS to extract call-related components

Constructing RentPDGs from Smart Contracts

- **Issues of DFS**: not consider **inter-procedural call contexts**
  - may falsely include nodes in **infeasible paths,** which are actually not connected to external calls

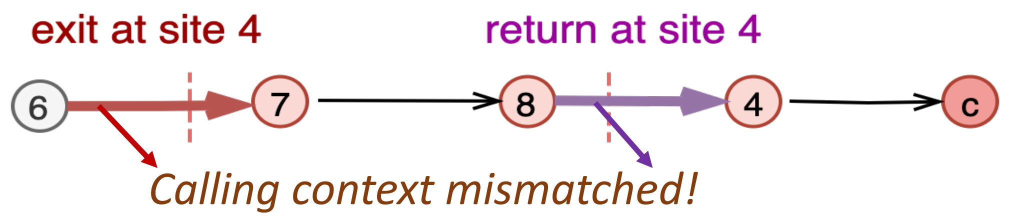


(Example) DFS-based RentPDG Construction

Nodes: {c, 4, 3, 2, 1, 7, 8, ~~6~~} *No feasible paths from 6 to external calls*

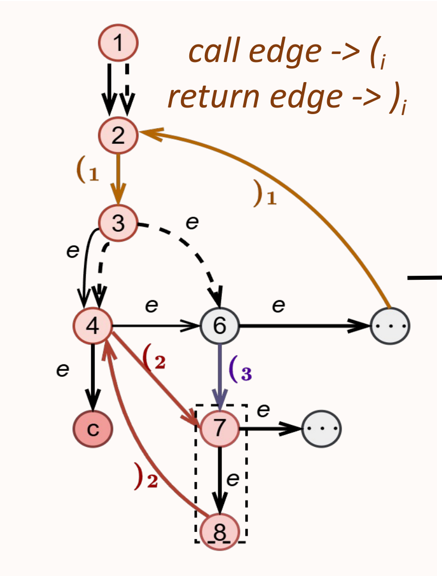
Edges: $\{e_{1\text{->}2},\ e_{2\text{->}3}, \cdots, \cdots\}$

Infeasible Path: 6 to c



Constructing RentPDGs from Smart Contracts

- Context-sensitive reachability analysis
  - Symbolize edges via a **context-free language** (CFL) => analyze path feasibility
  - Combine CFL with adjacency-matrix-based reachability analysis



Edge Symbolization

$$
\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & c \\
\hline
1 & \emptyset & (_1 & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
2 & \emptyset & \emptyset & e & e & \emptyset & \emptyset & \emptyset \\
3 & \emptyset & \emptyset & \emptyset & e & (_2 & \emptyset & e \\
4 & \emptyset & \emptyset & \emptyset & \emptyset & (_3 & \emptyset & \emptyset \\
5 & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & e & \emptyset \\
6 & \emptyset & \emptyset &)_2 &)_3 & \emptyset & \emptyset & \emptyset \\
c & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset
\end{array}
\longrightarrow
\underbrace{\begin{pmatrix} \emptyset \\ \emptyset \\ e \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{pmatrix}}_{\boldsymbol{\mu}_1^c}
+
\underbrace{\begin{pmatrix} \emptyset \\ ee \\ \emptyset \\ \emptyset \\ \emptyset \\ )_2e \\ \emptyset \end{pmatrix}}_{\boldsymbol{\mu}_2^c = \mathbf{A} \cdot \boldsymbol{\mu}_1^c}
+ \cdots =
\underbrace{\begin{pmatrix} \{(_1ee, \cdots\} \\ \{ee, \cdots\} \\ \{e, (_2e)_2e, \cdots\} \\ \cancel{(_3e)_2e} \\ \{e)_2e\} \\ \{)_2e, \cdots\} \\ \emptyset \end{pmatrix}}_{\boldsymbol{\mu}^c}
$$

Infeasible

Symbolic Adjacency Matrix **A**

1-order reachability

2-order reachability

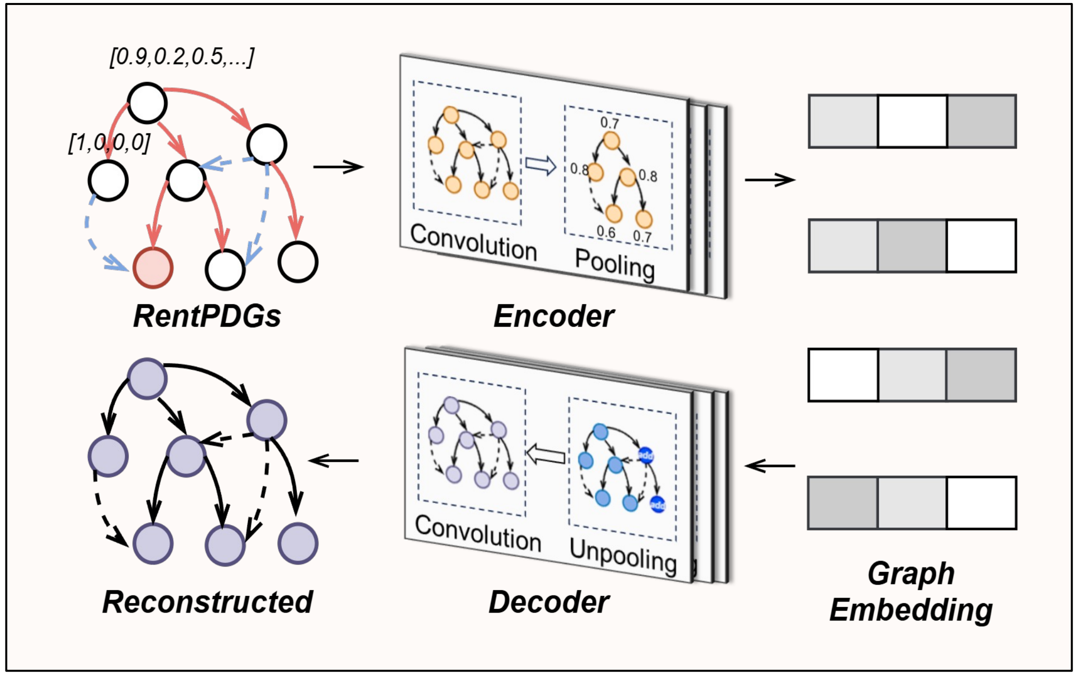
complete reachability

Step #3:

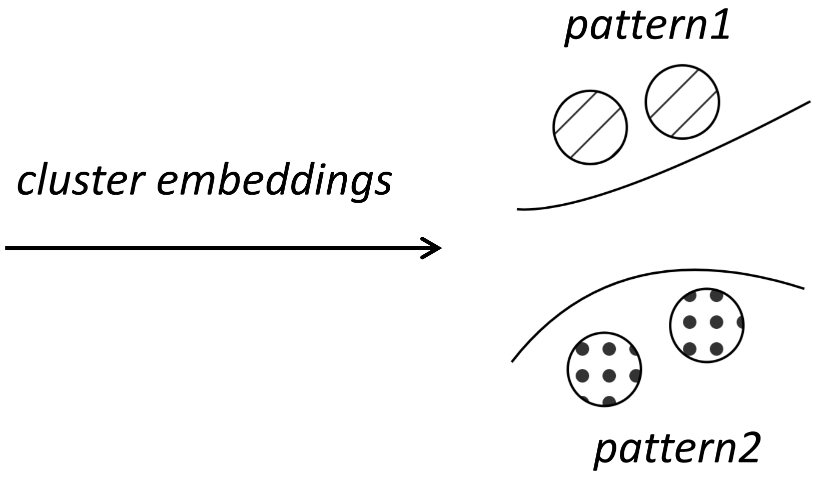
Use **a recognition model** to automatically learn anti-reentrancy semantics inherent in RentPDGs

Anti-Reentrancy Recognition Model

- We train a graph autoencoder
  - To capture semantics into **graph embedding vectors**
- Cluster embedding vectors => find typical anti-reentrancy patterns

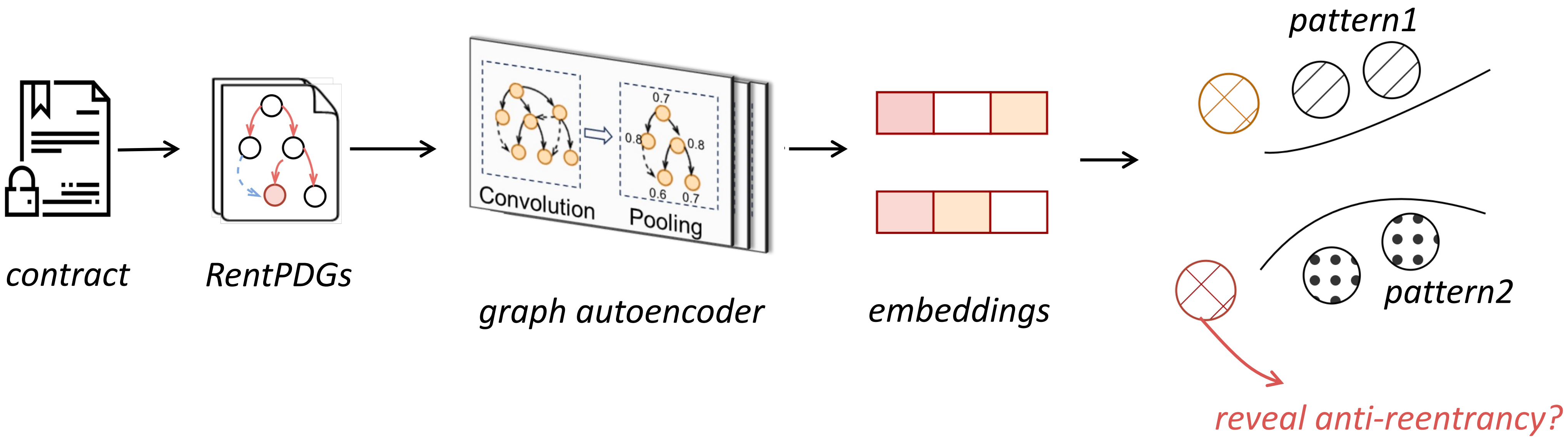


Training a graph auto-encoder



Anti-reentrancy Recognition Model

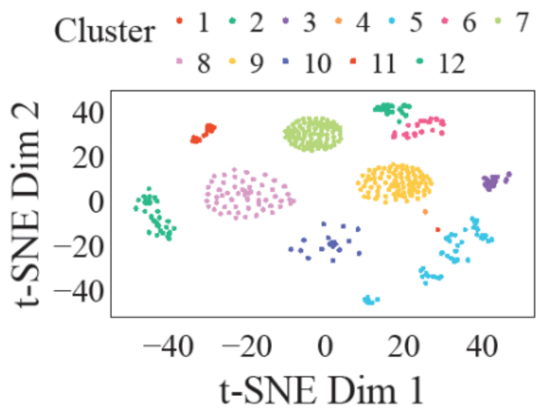
- Recognizing anti-reentrancy patterns
  - If **RentPDG embeddings** fall within learned clusters => protected with anti-reentrancy patterns



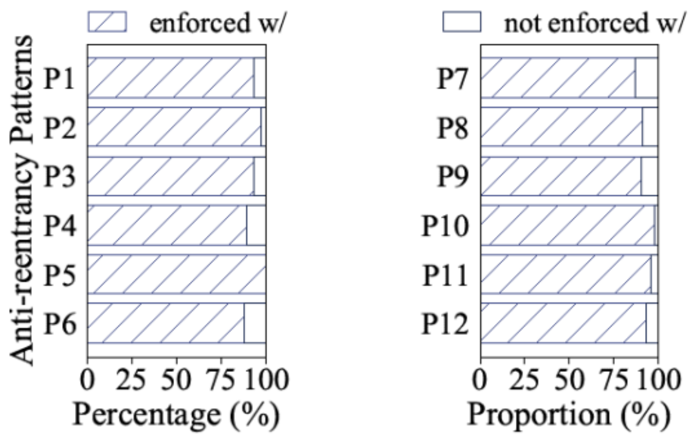
Experiment Evaluation

- Dataset: **40K real-world** smart contracts on Ethereum
  - Diverse types: ERC721, ERC777, ERC 1155, etc
- Clustering result: 12 clusters
  - For each cluster, we randomly select some contracts to review code patterns

Visualized Clustering Result

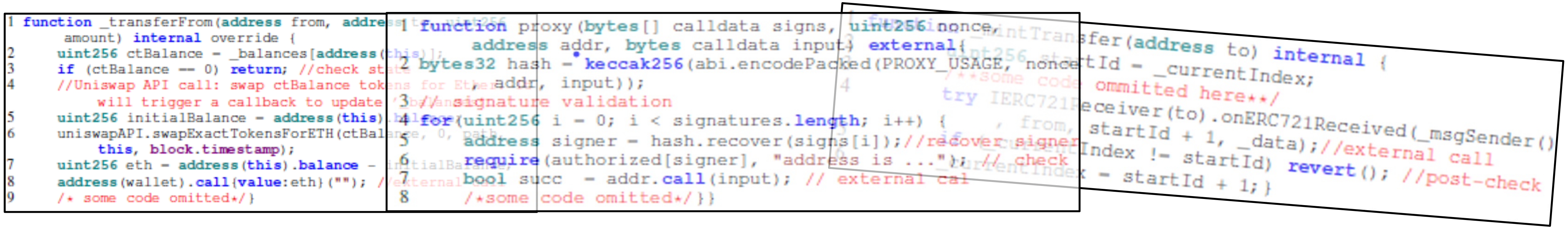


Clustering Statistics



Exp 1: Anti-reentrancy Patterns Learned

- By manually inspecting, we found 12 anti-reentrancy patterns
  - ➢ reentrancy guard, EOA restriction, ... (see details in our paper)



- Out of 12 patterns, 8 patterns are newly explored

Anti-reentrancy Type	Literature		
	Research	Blog	Official Document
Safe Ether Transfer (P1)	✓	✓	✓
Mutex Variable (P2)	✓	✓	-
Sender Check (P3)	✓	-	-
Reentrancy Guard (P6)	✓	✓	✓
P4-5, P7-12	-	-	-

Literature Review

Exp 1: Anti-reentrancy Patterns Learned (Examples)

- External owned account (EOA) restriction
  - EOA does not have any code
  - If caller is EOA => cannot make a reentrant call

variable 'tx.origin' denotes EOA

```
modifier callerIsUser() {
    require(tx.origin == msg.sender, "...");// require caller is user
}
function mint(uint256 _mintAmount) public payable callerIsUser {
    /* some code omitted */
}
```

The anti-reentrancy patterns are rarely discussed in the literature

Exp 1: Anti-reentrancy Patterns Learned (Examples)

- External owned account (EOA) restriction
  - EOA does not have any code
  - If caller is EOA => cannot make a reentrant call

variable ‘tx.origin’ denotes EOA

```
modifier callerIsUser() {
    require(tx.origin == msg.sender, "...");// require
        caller is user
}
function mint(uint256 _mintAmount) public payable
      callerIsUser {
    /* some code omitted */
}
```

- Access Frequency Limitation
  - Attackers cannot reenter a function in a time frame

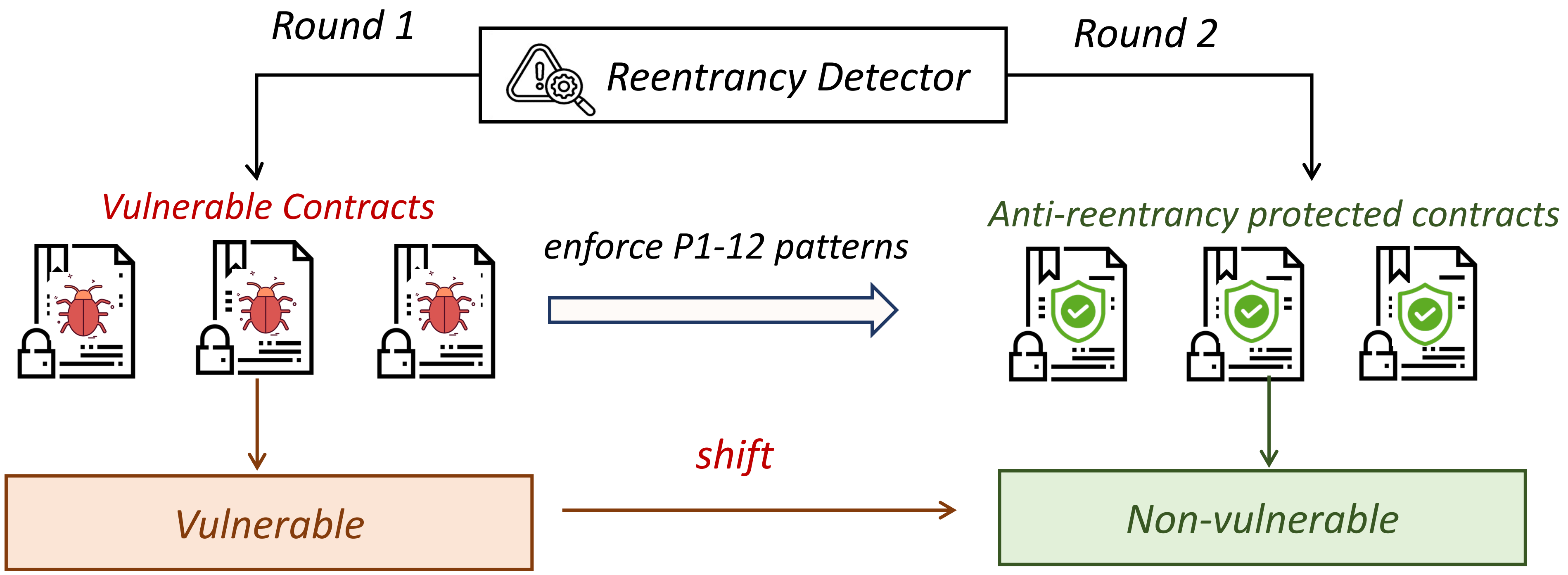
```
function _transfer(address from, address to, uint256 amount)
      internal override {
    if (block.timestamp > lastBurnTime + BurnFreq)
        //check access frequency
        autoBurnLPTokens();
}
function autoBurnLPTokens() internal returns (bool) {
    lastBurnTime=block.timestamp; //record last access time
    pair.sync(); //external call
}
```

Control the access frequency

The anti-reentrancy patterns are rarely discussed in the literature

Exp 2: Can Existing Tools Detect the Learned Patterns?

- For reliable evaluation, we conduct scanning comparison experiments



We say the detector can identify anti-reentrancy patterns

Exp 2: Can Existing Tools Detect the Learned Patterns?

- For reliable evaluation, we conduct scanning comparison experiments

TABLE I: Comparison Experiments. Here, 6 tools are applied to scan contracts before and after anti-reentrancy enforcement.

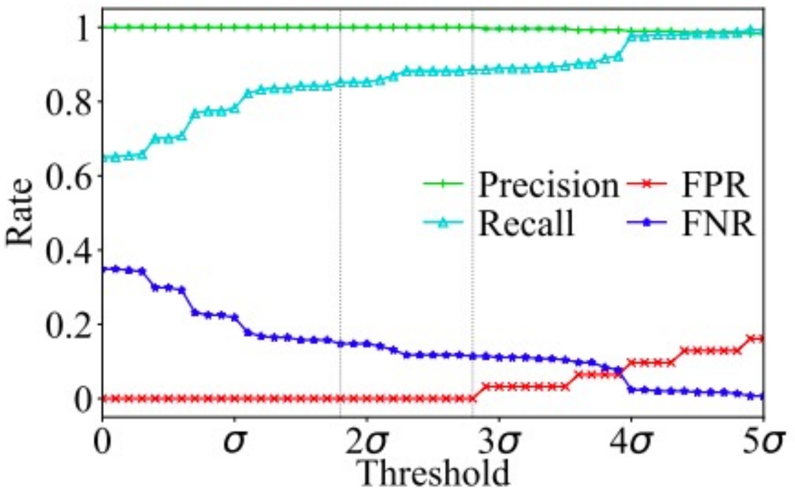
Setup		Slither	Securify	Mythril	Conkas	Smartian	Sailfish
Detection Round #1	Original*	31	29	10	31	13	28
Detection Round #2	w/ P1	**0/31**	29/29	10/10	31/31	**0/13**	**0/28**
	w/ P2	31/31	29/29	10/10	**4/31**	**0/13**	**2/28**
	w/ P3	31/31	29/29	10/10	31/31	13/13	28/28
	w/ P4	31/31	29/29	10/10	31/31	13/13	28/28
	w/ P5	31/31	29/29	10/10	31/31	13/13	28/28
	w/ P6	31/31	29/29	10/10	**4/31**	**0/13** *shift*	**2/28**
	w/ P7	31/31	29/29	10/10	31/31	13/13	28/28
	w/ P8	31/31	29/29	10/10	31/31	13/13	28/28
	w/ P9	31/31	29/29	10/10	31/31	13/13	28/28
	w/ P10	31/31	29/29	10/10	31/31	**0/13**	28/28
	w/ P11	31/31	29/29	10/10	31/31	13/13	28/28
	w/ P12	31/31	29/29	10/10	31/31	13/13	28/28

* It refers to original, vulnerable contracts without anti-reentrancy patterns enforced.

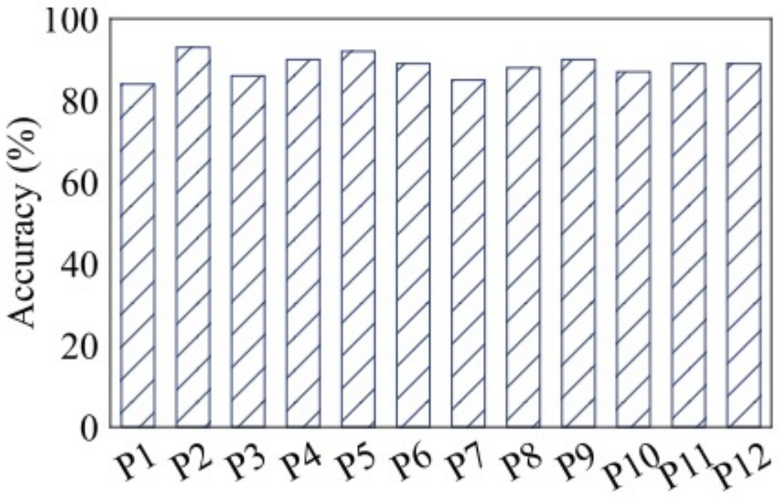
Existing tools only detect 4 patterns at most

Exp 3: Anti-reentrancy Recognition Performance

- Our system can detect anti-reentrancy patterns with recall rates over 85% and 100% precision



(a) Precision, Recall, FNR, and FPR by Varying Detection Thresholds



(b) Anti-reentrancy Recognition Accuracy w/ 2.3σ Threshold

Exp 4: Integrated with Existing Detection Tools

- Integrate our system into the workflow of existing tools
  - Reduce FPs by at least 85%
  - Not compromise their original detection capability

TABLE II: Integrating AutoAR with 6 Tools to Scan 31 Vulnerable and 298 Non-Vulnerable Contracts

Detectors		Recall	Precision	#TPs	#FPs	FNR	FPR
Slither	Original	1	0.128	31	211	0	0.708
	w/ AutoAR	1	0.596	31	21	0	0.070 ↓**(90%)**
Securify	Original	0.935	0.184	29	129	0.065	0.433
	w/ AutoAR	0.935	0.644	29	16	0.065	0.054 ↓**(88%)**
Mythril	Original	0.323	0.161	10	52	0.677	0.174
	w/ AutoAR	0.323	0.588	10	7	0.677	0.023 ↓**(87%)**
Conkas	Original	1	0.164	31	158	0	0.530
	w/ AutoAR	1	0.564	31	24	0	0.081 ↓**(85%)**
Smartian	Original	0.419	0.283	13	33	0.581	0.111
	w/ AutoAR	0.419	0.867	13	2	0.581	0.007 ↓**(94%)**
Sailfish	Original	0.903	0.184	28	124	0.097	0.416
	w/ AutoAR	0.903	0.636	28	16	0.097	0.054 ↓**(87%)**

Conclusion

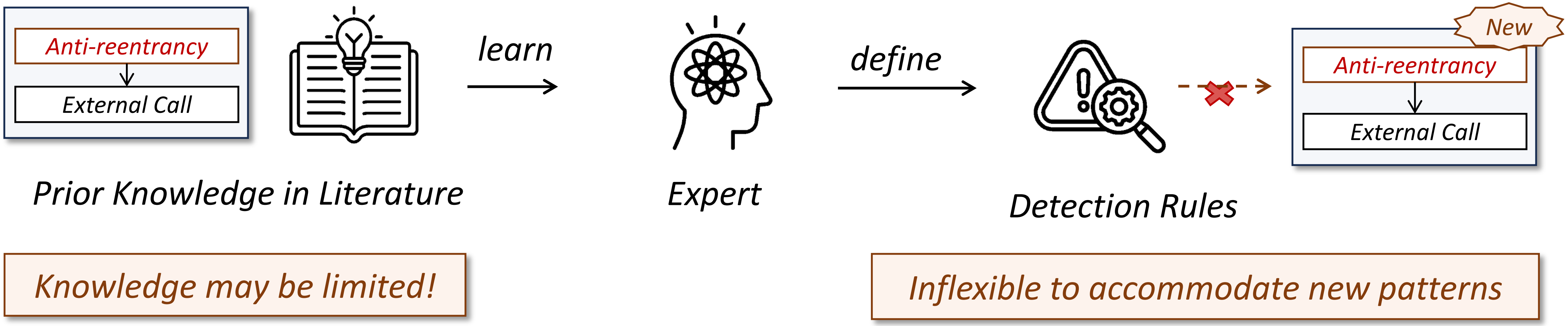
- An automated tool for identifying anti-reentrancy patterns on Ethereum
  - Help refine existing reentrancy detectors
- Utilize deep learning with a specialized data structure to precisely capture anti-reentrancy semantics
- Experimental evaluation shows our tool can significantly reduce FPs from existing reentrancy detectors

Thank You!

Q & A

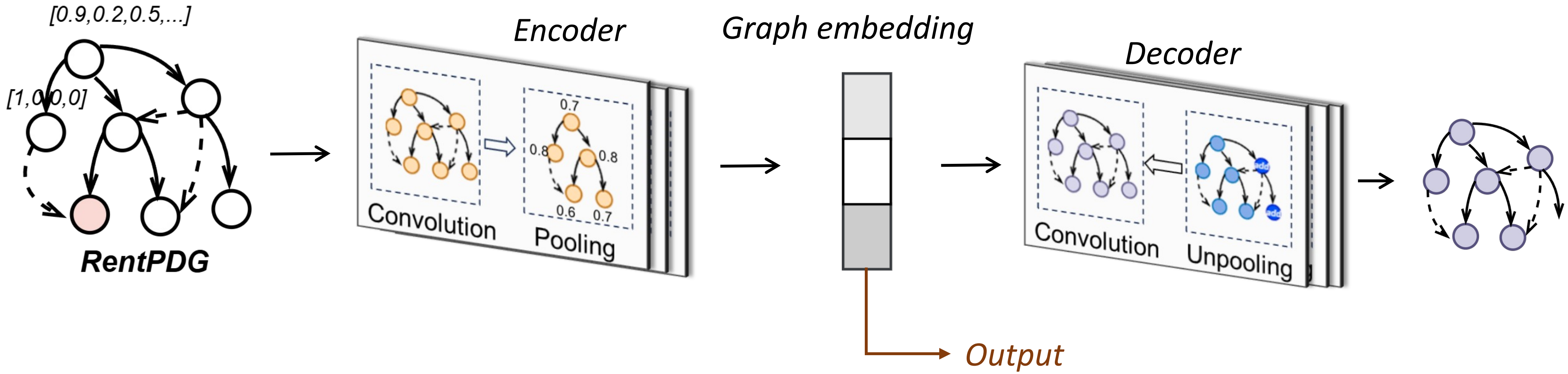
Backup: Intuitive Anti-reentrancy Detection Method

- Intuitive: **manually defining detection rules** with prior knowledge
  - Challenge 1: prior knowledge may not cover all anti-reentrancy patterns
  - Challenge #2: cannot swiftly accommodate new patterns



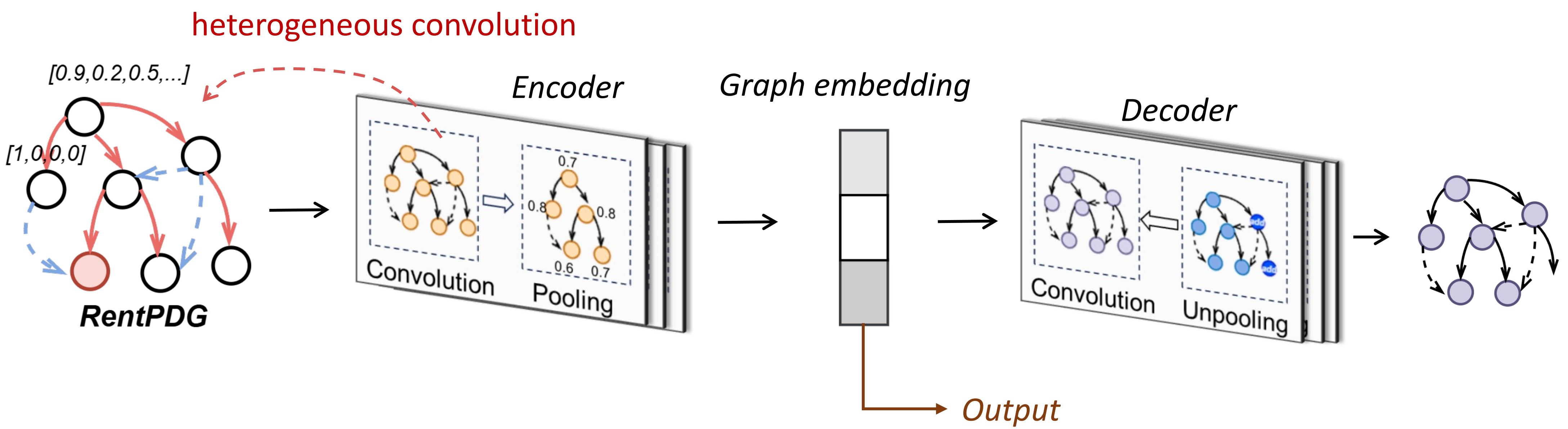
Backup: Graph AutoEncoder

- Graph auto-encoder automatically learn semantics from RentPDGs



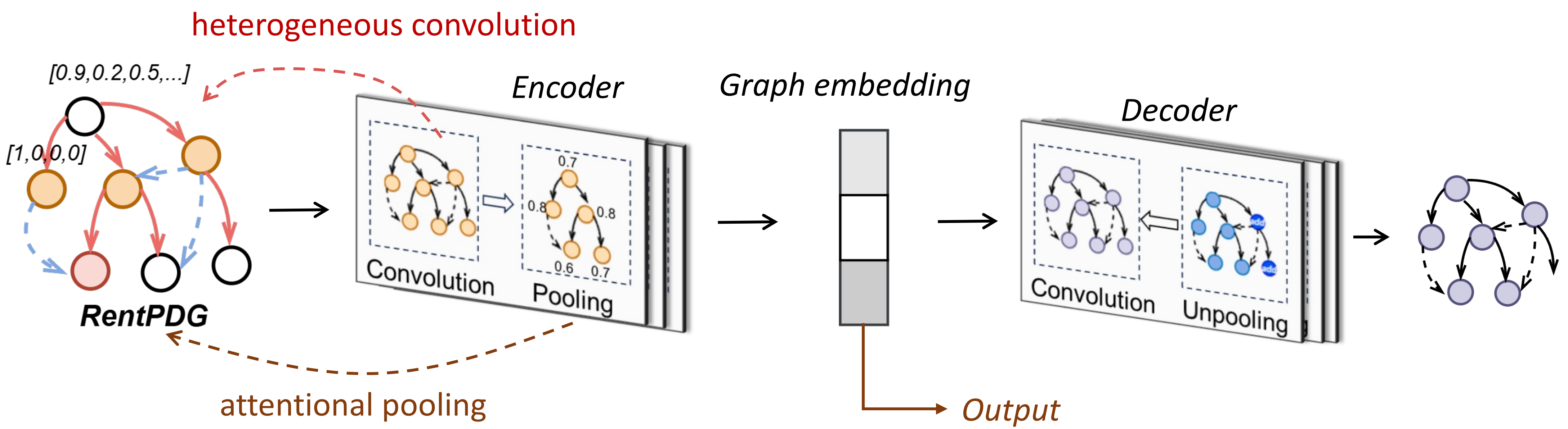
Backup: Graph AutoEncoder

- Graph auto-encoder automatically learn semantics from RentPDGs
  - Heterogeneous graph convolution => manages different types of edges



Backup: Graph AutoEncoder

- Graph auto-encoder automatically learn semantics from RentPDGs
  - Heterogeneous graph convolution => manages different types of edges
  - graph attentional pooling => capture crucial nodes



Backup: Anti-reentrancy Detection

- Clustering-based detection
  - Use cluster centroids to detect if anti-reentrancy semantics are within RentPDG embeddings
  - Set a distance detection threshold τ