Zircuit

Temporarily Restricting Solidity Smart Contract Interactions

Valerian Callens, Quantstamp
Zeeshan Meghji, Quantstamp
Jan Gorzny, Zircuit

DAPPS 2024, Shanghai, China
July 2024

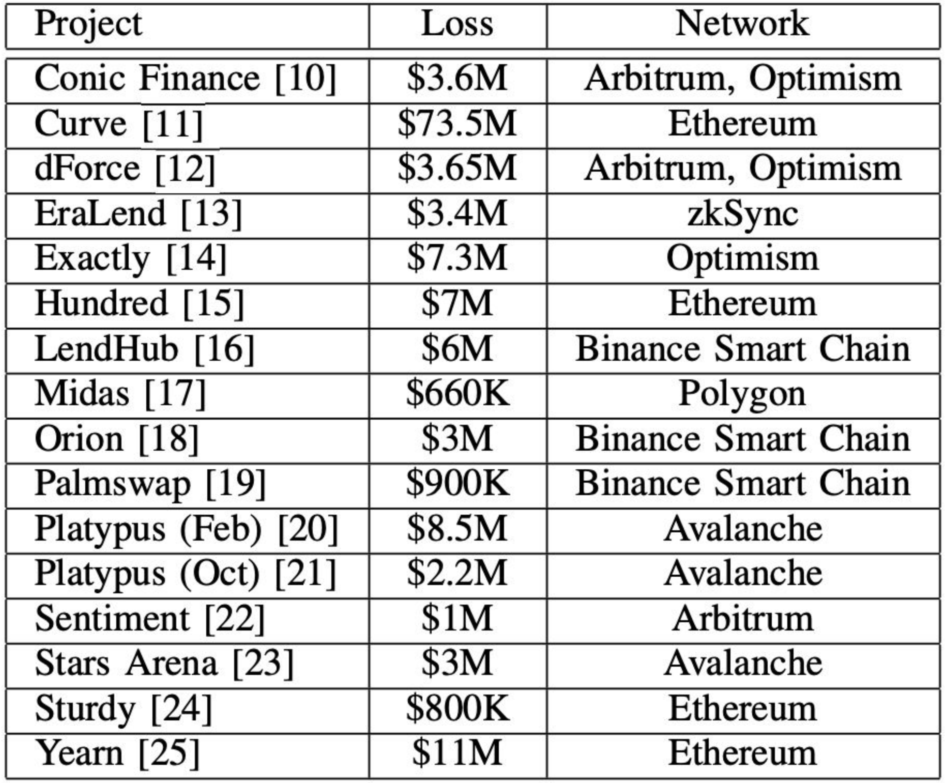
What is this talk about?

dApps, implemented via **smart contracts**, are

- **popular**,
- responsible for **millions of dollars** in cryptocurrencies, and
- **non-trivial to develop**, especially for multiple chains simultaneously

How can we **restrict access to them** and with **what trade-offs**?

Motivation



Project	Loss	Network
Conic Finance [10]	\$3.6M	Arbitrum, Optimism
Curve [11]	\$73.5M	Ethereum
dForce [12]	\$3.65M	Arbitrum, Optimism
EraLend [13]	\$3.4M	zkSync
Exactly [14]	\$7.3M	Optimism
Hundred [15]	\$7M	Ethereum
LendHub [16]	\$6M	Binance Smart Chain
Midas [17]	\$660K	Polygon
Orion [18]	\$3M	Binance Smart Chain
Palmswap [19]	\$900K	Binance Smart Chain
Platypus (Feb) [20]	\$8.5M	Avalanche
Platypus (Oct) [21]	\$2.2M	Avalanche
Sentiment [22]	\$1M	Arbitrum
Stars Arena [23]	\$3M	Avalanche
Sturdy [24]	\$800K	Ethereum
Yearn [25]	\$11M	Ethereum

Concerns for Interactions

- Is reentrancy a thing of the past? **No**:
  - Reentrancy was the cause of **several** of the previous hacks
  - "New" forms, like **read-only reentrancy** that was not explicitly studied before caused others
- Is reentrancy the only problematic interaction? **No**:
  - Non-reentrant exploits exist (e.g., from **flash loans**)
- Are all methods the same on all "EVM-Compatible" / Solidity supporting blockchains? **No**:
  - Various rollups, let alone other layer one blockchains, **change the semantics** of op-codes

ROLLUPCODES

Results

We want to be able to **restrict** interactions at various levels. We…

- Review **existing solutions** for reentrancy
- Generalize approaches to **sets of functions** and **dApps**
- Describe **read-only reentrancy**
- Restrict interactions within the same…
  - **Same transaction**
  - **Block** or **time duration**
- Highlight **future work**

Outline

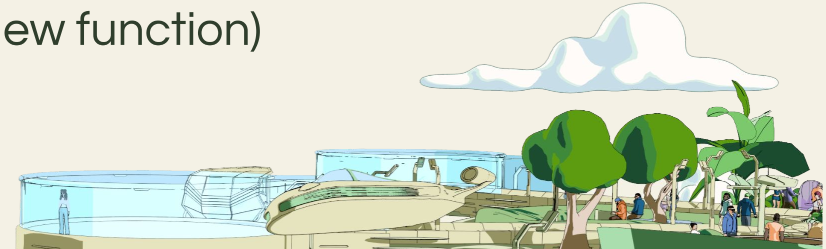
- Existing solutions
- Sets of Functions and dApps
  - Read-only reentrancy example
- Duration-based restrictions
  - Same transactions
  - block or time based duration
- Future work

Outline

- **Existing solutions**
- Sets of Functions and dApps
  - Read-only reentrancy example
- Duration-based restrictions
  - Same transactions
  - block or time based duration
- Future work

Existing Solutions

- **Checks-Effects Interaction** pattern
  - **Design pattern** to mitigate effects of reentrancy, even if it occurs
- **Gas limiting** external calls
  - **Don't supply enough gas** to reenter; hardcoding **values that may change**
- **Non-reentrant modifier** on functions (e.g., from OpenZeppelin)
  - Uses a **mutex** to ensure non-reentrancy
  - **Always writes** (cannot be used for view function)

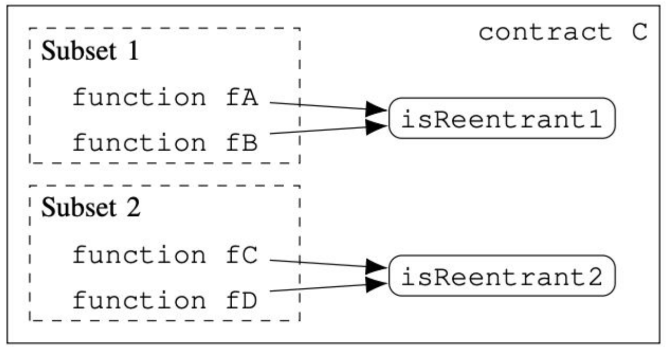


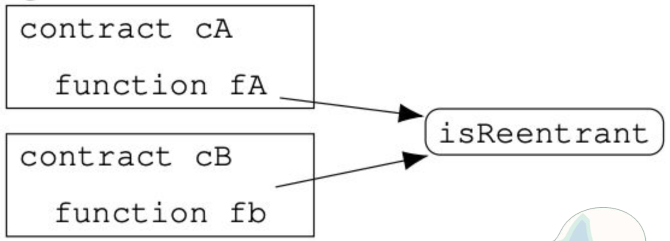
Outline

- Existing solutions
- **Sets of Functions and dApps**
  - Read-only reentrancy example
- Duration-based restrictions
  - Same transactions
  - block or time based duration
- Future work

Sets of Functions and dApps

- Modifiers don't need to be limited to single functions – **they can be shared**
- Shared modifiers mean protection across entire dApps
- **Can use multiple locks to allow some reentrancy**
- **Can use the same lock for multiple contracts**
- Protecting **opposite actions** can be valuable, especially in conjunction with duration-based locks.



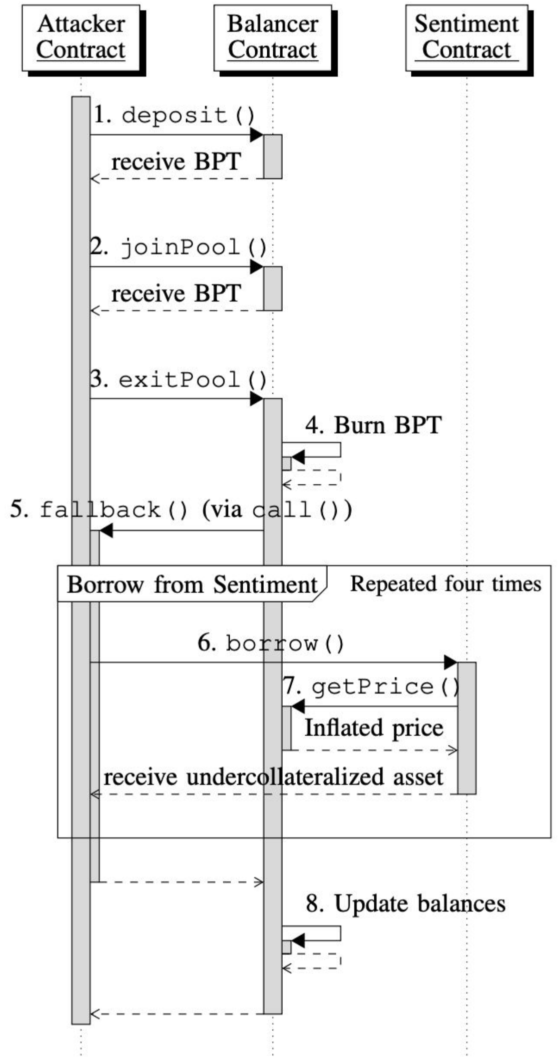


Outline

- Existing solutions
- Sets of Functions and dApps
  - **Read-only reentrancy example**
- Duration-based restrictions
  - Same transactions
  - block or time based duration
- Future work

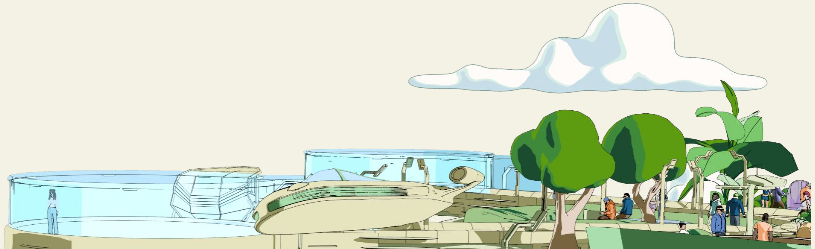
Motivation

- Description of the **Sentiment protocol** incident of 2023
- “Read-only” reentrancy: a **read-only** function of the dApp was entered at a bad time; involved other dApps
- **$1M loss** of cryptocurrency
- **Fixed after** the issue (but that’s too late)



Read-only Reentrancy

- **Not solved properly** ideally we don't want to just write via mutexes everywhere – we want view functions
  - EIP-1153 introduces **transient storage** which is a middle ground; not yet well studied or exemplified.
- Difficult to reason about; often overlooked by auditors
- May need more "heavyweight" properties or invariants; may be most costly in other ways.



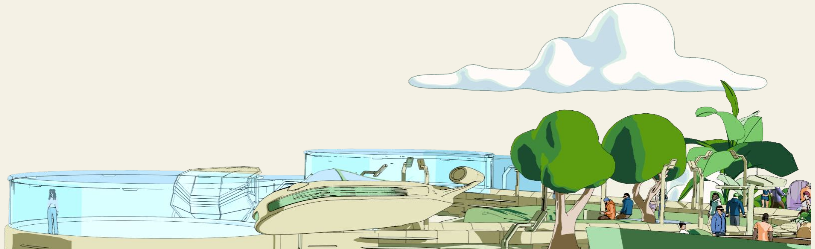
Outline

- Existing solutions
- Sets of Functions and dApps
  - Read-only reentrancy example
- **Duration-based restrictions**
  - **Same transactions**
  - block or time based duration
- Future work

Same Transaction

- Allow **two** or more calls **within the same block** but **not within the same transaction**
- Uses **warm** and **cold** memory access

```
modifier calledMaxOncePerTransaction() {
    address addressToCheck = address(
        uint160(bytes20(blockhash(block.
        number))));
    uint256 initialGas = gasleft();
    uint256 temp = addressToCheck.balance;
    uint256 gasConsumed = initialGas
        - gasleft();
    require(gasConsumed == 2631,
     "already called in this transaction");
    _;
}
```



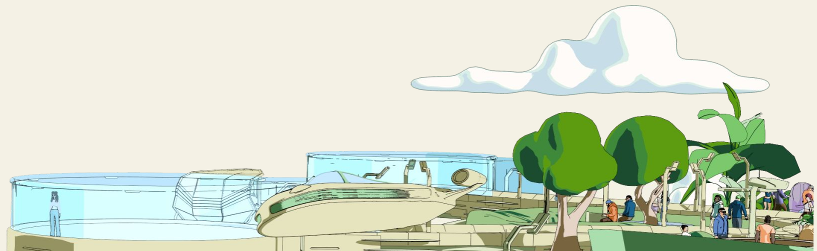
Outline

- Existing solutions
- Sets of Functions and dApps
  - Read-only reentrancy example
- **Duration-based restrictions**
  - Same transactions
  - **block or time based duration**
- Future work

Block or Time-Based Duration

- Disallow two or more transactions **per account** within a **time period** or **number of blocks**
- Can be powerful for "opposite actions" like deposit and withdrawal; **no flash loans**
- No meaningful difference on Ethereum, but **different on some layer two networks**

```
abstract contract ReentrancyGuardDuration {
    uint256 private constant _DELTA = 60
        seconds;
    mapping(address => uint256) public
        latestEntry;
    modifier nonReentrant() {
        require(latestEntry[msg.sender] +
            _DELTA <= block.timestamp,
            "Called again too soon");
        latestEntry[msg.sender] =
            block.timestamp;
        _;
    }
}
```



Outline

- Existing solutions
- Sets of Functions and dApps
  - Read-only reentrancy example
- Duration-based restrictions
  - Same transactions
  - block or time based duration
- **Future work**

Conclusion & Future Work

We explicitly introduced...

- **read-only reentrancy** attacks
- **Same transaction, block,** or **time duration** level restrictions
- Generalize approaches to **sets of functions** and **dApps**

Future work:

- **Analysis of changes for opcodes** on layer two networks
- **Other approaches** to counter read-only reentrancy
- **Empirical analysis** of these approaches (which are more likely to break composability? gas cost trade-offs?)
- EIP-1153 changes and their **security implications**

Zircuit

Thank you!

jan@zircuit.com