Multiple Withdrawal Attack in ERC20 Tokens

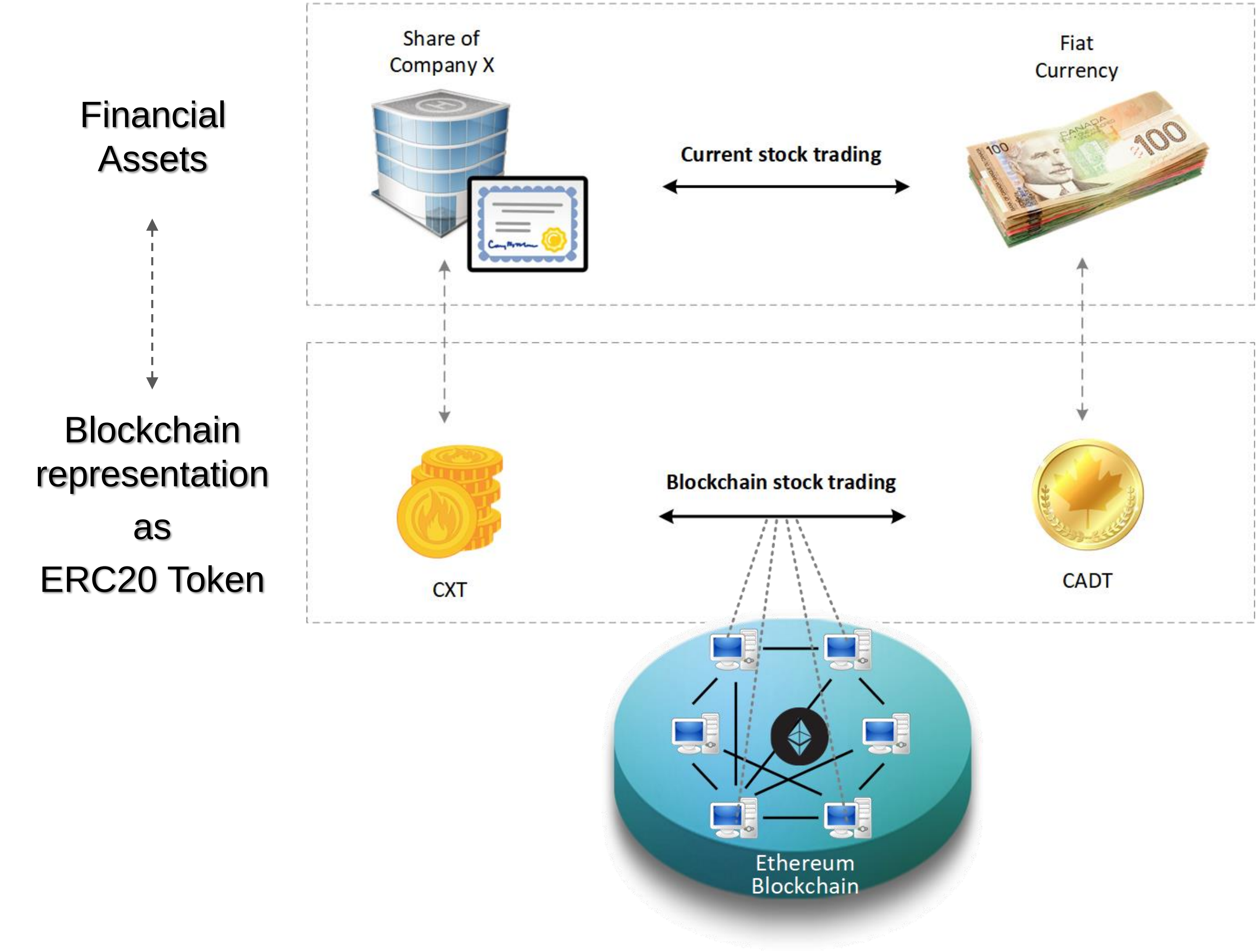
Presented by: Reza Rahimian

Supervised by: Dr. Jeremy Clark

26-Feb-19

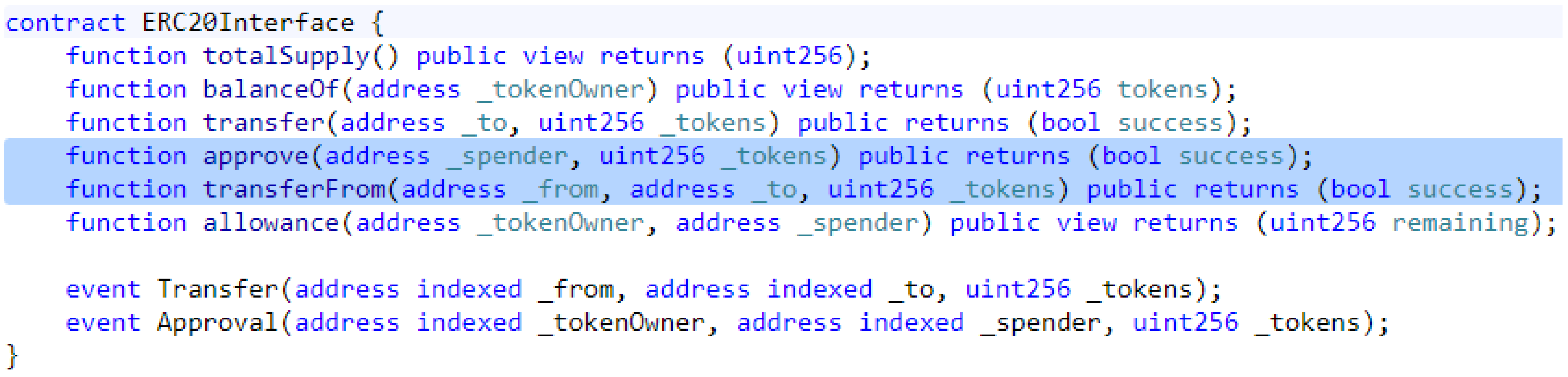
Concordia University

Introduction to ERC20 Tokens



Specifications of ERC20 Standard

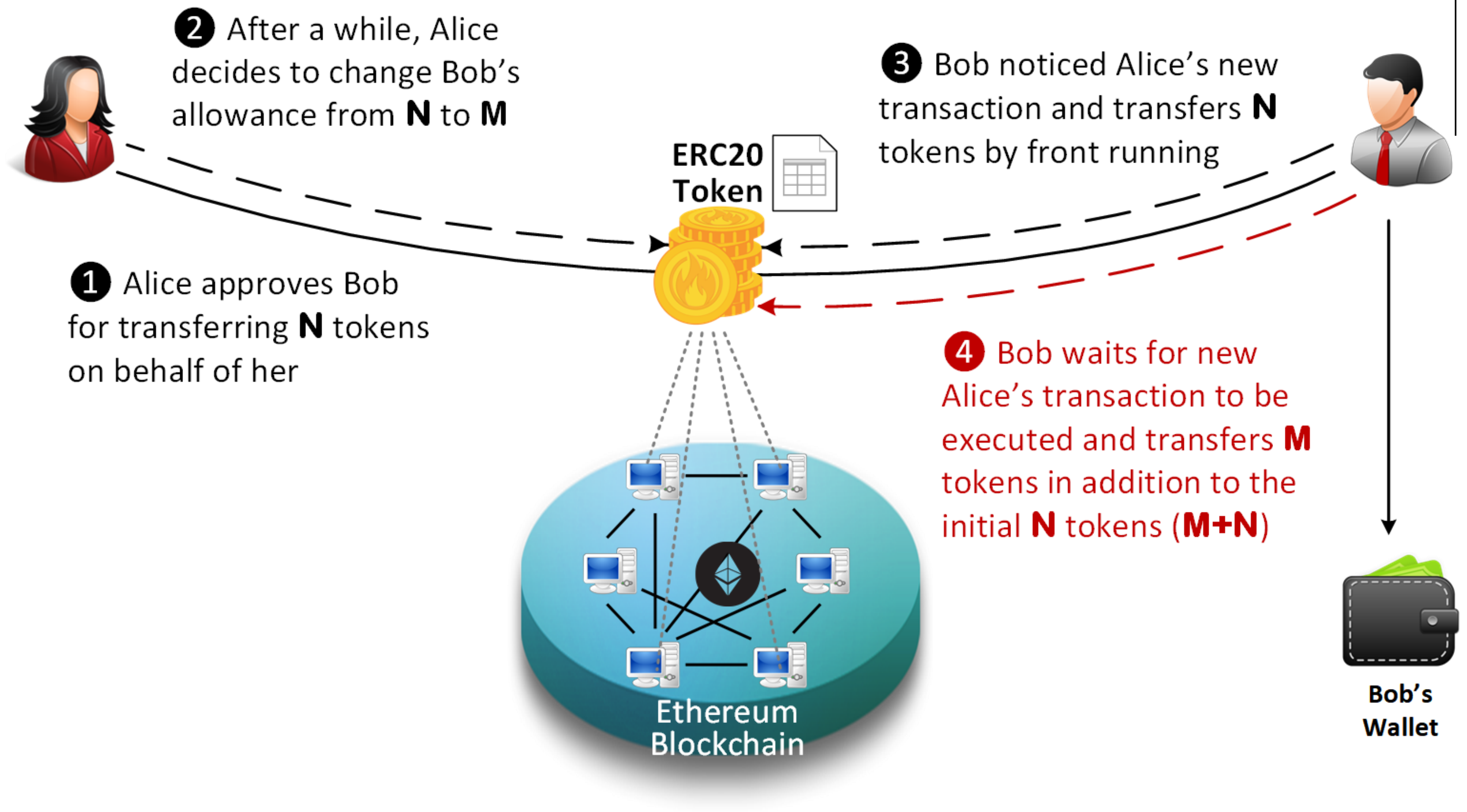
1. Calling approve function has to overwrite current allowance with new allowance.
2. approve method does not adjust allowance, it sets new value of allowance.
3. Transferring 0 values by transferFrom method MUST be treated as normal transfers and fire the Transfer event as non-zero transactions.
4. Introducing new methods violates ERC20 API and it MUST be avoided for having compatible token with already deployed smart contracts.
5. Spender will be allowed to withdraw from approver account multiple times, up to the allowed amount.
6. Transferring initial allowed tokens is considered as legitimate transfer. It could happen right after approval or before changing allowance.
7. Race condition MUST not happen in any case to prevent multiple withdrawal from the account.



```
contract ERC20Interface {
    function totalSupply() public view returns (uint256);
    function balanceOf(address _tokenOwner) public view returns (uint256 tokens);
    function transfer(address _to, uint256 _tokens) public returns (bool success);
    function approve(address _spender, uint256 _tokens) public returns (bool success);
    function transferFrom(address _from, address _to, uint256 _tokens) public returns (bool success);
    function allowance(address _tokenOwner, address _spender) public view returns (uint256 remaining);

    event Transfer(address indexed _from, address indexed _to, uint256 _tokens);
    event Approval(address indexed _tokenOwner, address indexed _spender, uint256 _tokens);
}
```

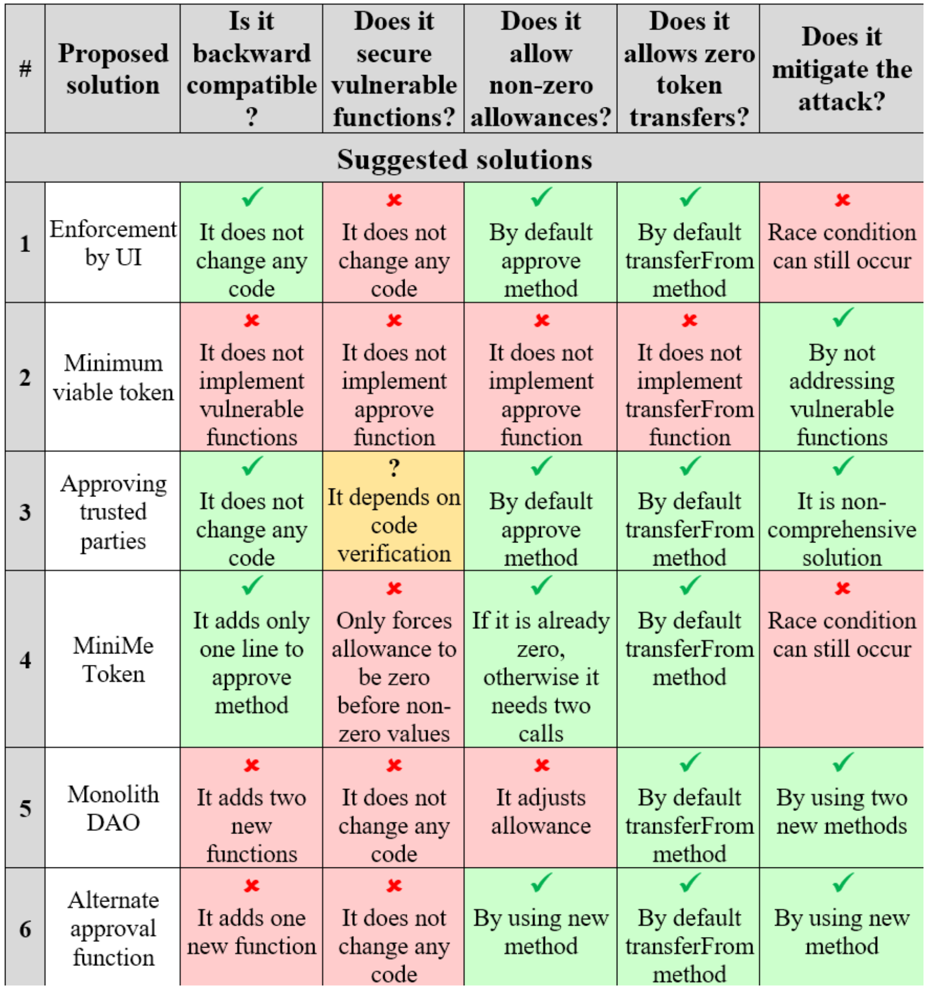
Multiple Withdrawal Attack


❷ After a while, Alice decides to change Bob's allowance from **N** to **M**
❸ Bob noticed Alice's new transaction and transfers **N** tokens by front running
ERC20 Token
❶ Alice approves Bob for transferring **N** tokens on behalf of her
❹ Bob waits for new Alice's transaction to be executed and transfers **M** tokens in addition to the initial **N** tokens (**M+N**)
Ethereum Blockchain
Bob's Wallet

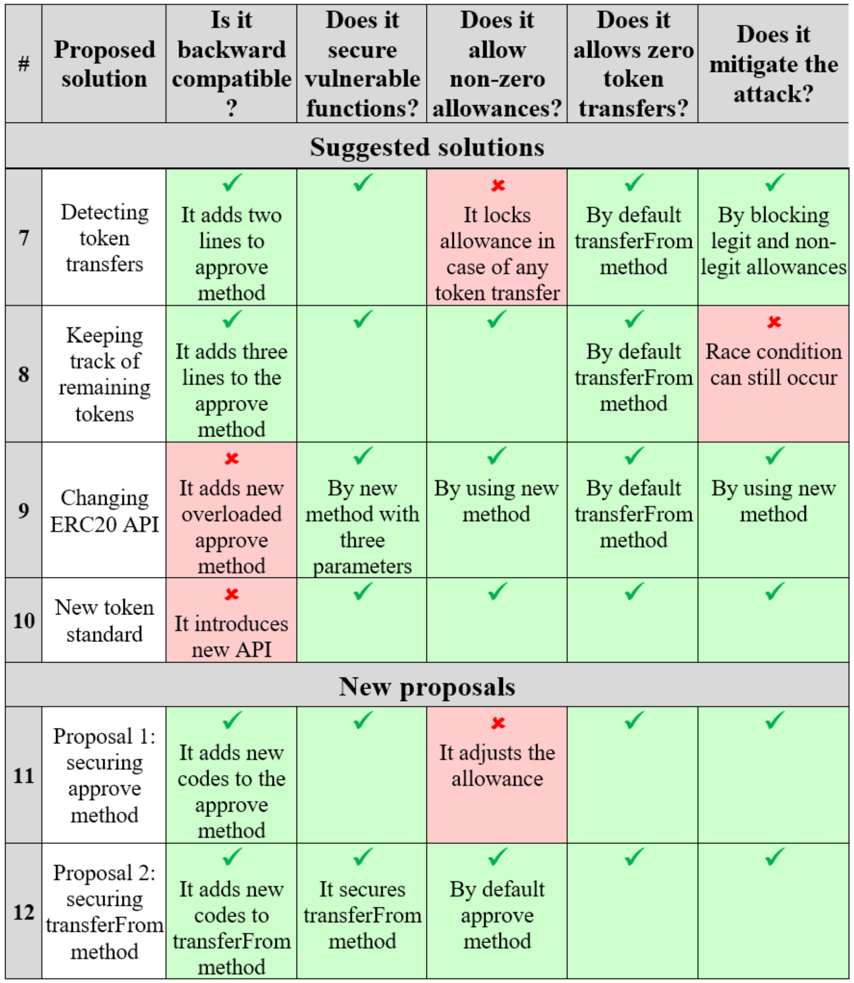
☑ The goal is to prevent Bob from transferring more M tokens

Evaluating Proposed Solutions

ERC20 specifications



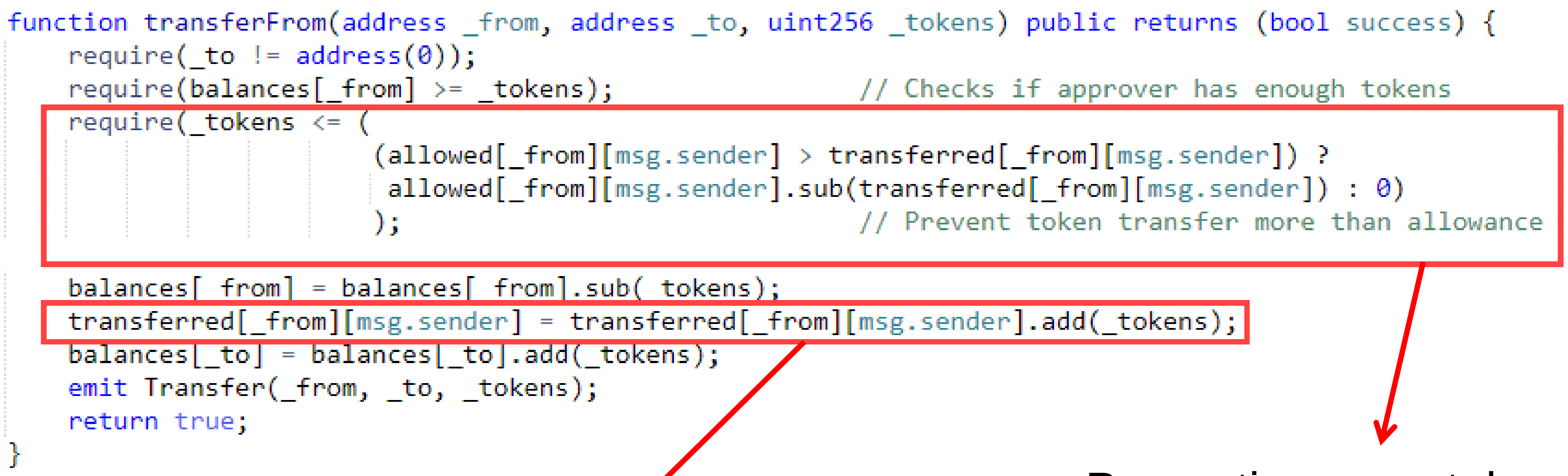
#	Proposed solution	Is it backward compatible ?	Does it secure vulnerable functions?	Does it allow non-zero allowances?	Does it allows zero token transfers?	Does it mitigate the attack?
	Suggested solutions					
1	Enforcement by UI	✓ It does not change any code	✗ It does not change any code	✓ By default approve method	✓ By default transferFrom method	✗ Race condition can still occur
2	Minimum viable token	✗ It does not implement vulnerable functions	✗ It does not implement approve function	✗ It does not implement approve function	✗ It does not implement transferFrom function	✓ By not addressing vulnerable functions
3	Approving trusted parties	✓ It does not change any code	? It depends on code verification	✓ By default approve method	✓ By default transferFrom method	✓ It is non-comprehensive solution
4	MiniMe Token	✓ It adds only one line to approve method	✗ Only forces allowance to be zero before non-zero values	✓ If it is already zero, otherwise it needs two calls	✓ By default transferFrom method	✗ Race condition can still occur
5	Monolith DAO	✗ It adds two new functions	✗ It does not change any code	✗ It adjusts allowance	✓ By default transferFrom method	✓ By using two new methods
6	Alternate approval function	✗ It adds one new function	✗ It does not change any code	✓ By using new method	✓ By default transferFrom method	✓ By using new method



#	Proposed solution	Is it backward compatible ?	Does it secure vulnerable functions?	Does it allow non-zero allowances?	Does it allows zero token transfers?	Does it mitigate the attack?
	Suggested solutions					
7	Detecting token transfers	✓ It adds two lines to approve method	✓	✗ It locks allowance in case of any token transfer	✓ By default transferFrom method	✓ By blocking legit and non-legit allowances
8	Keeping track of remaining tokens	✓ It adds three lines to the approve method	✓	✓	✓ By default transferFrom method	✗ Race condition can still occur
9	Changing ERC20 API	✗ It adds new overloaded approve method	✓ By new method with three parameters	✓ By using new method	✓ By default transferFrom method	✓ By using new method
10	New token standard	✗ It introduces new API	✓	✓	✓	✓
	New proposals					
11	Proposal 1: securing approve method	✓ It adds new codes to the approve method	✓	✗ It adjusts the allowance	✓	✓
12	Proposal 2: securing transferFrom method	✓ It adds new codes to transferFrom method	✓ It secures transferFrom method	✓ By default approve method	✓	✓

Proposed solution

Securing **transferFrom** method instead of **approve** method



```
function transferFrom(address _from, address _to, uint256 _tokens) public returns (bool success) {
    require(_to != address(0));
    require(balances[_from] >= _tokens);                    // Checks if approver has enough tokens
    require(_tokens <= (
                        (allowed[_from][msg.sender] > transferred[_from][msg.sender]) ?
                         allowed[_from][msg.sender].sub(transferred[_from][msg.sender]) : 0)
                        );                                  // Prevent token transfer more than allowance

    balances[_from] = balances[_from].sub(_tokens);
    transferred[_from][msg.sender] = transferred[_from][msg.sender].add(_tokens);
    balances[_to] = balances[_to].add(_tokens);
    emit Transfer(_from, _to, _tokens);
    return true;
}
```

Keep tracking of transferred tokens

Preventing more token transfer than allowed