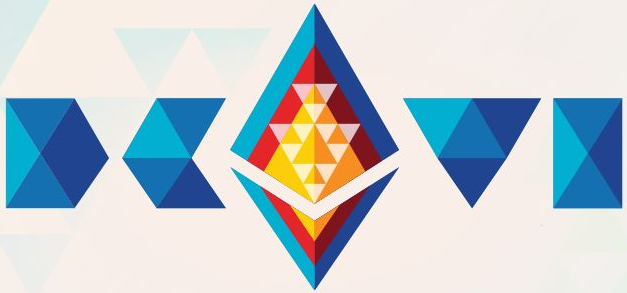


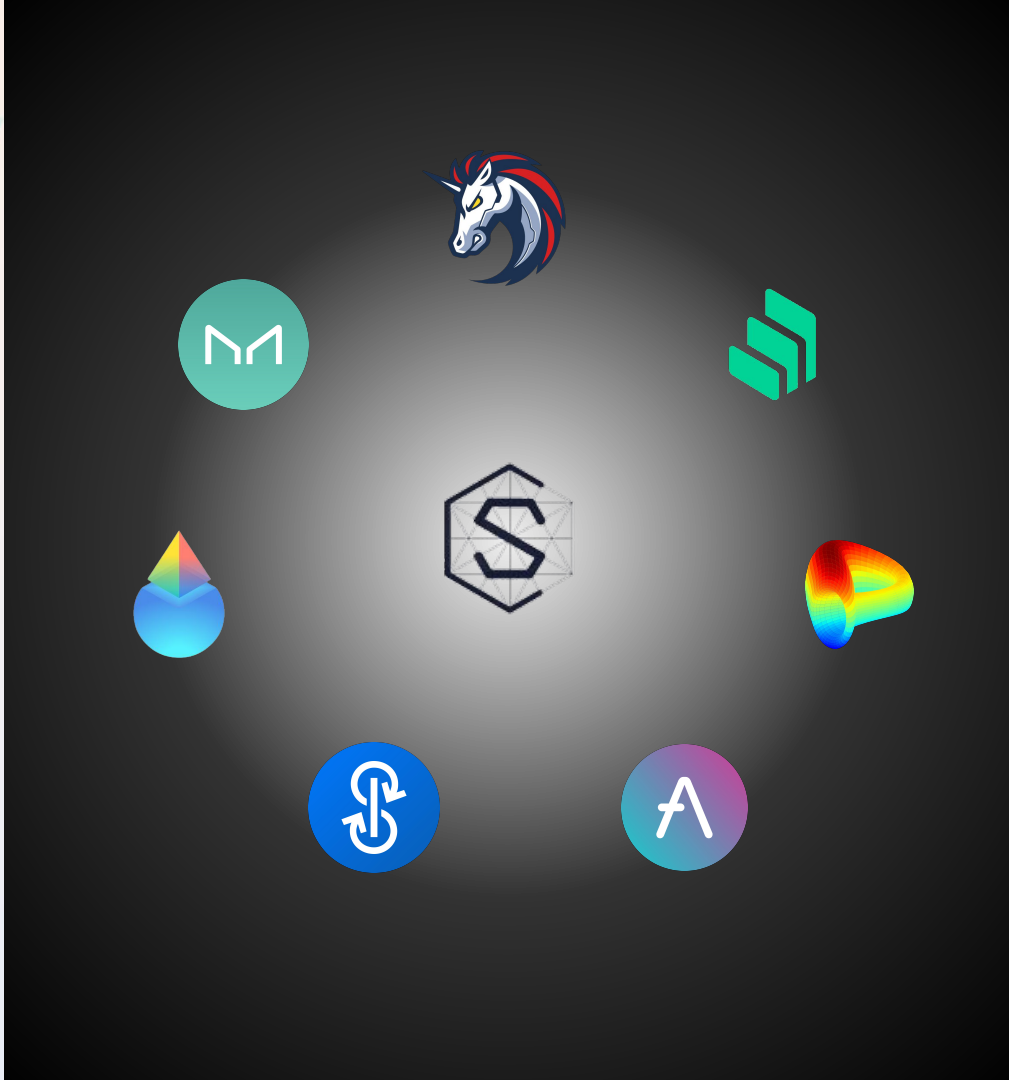
Read-only Reentrancy

Ioannis Sachinoglou

ChainSecurity

About ChainSecurity

- We are focused on blockchain security
- Smart contract audits
- Some of our clients:
  - Maker
  - Curve.fi
  - Compound
  - Aave
  - Yearn
  - 1inch
  - Lido



Why we should care

- It's a novel attack often neglected by developers and auditors
- More and more protocols interact with one another
- It has affected DeFi protocols integrating with Curve.fi
- Total of over $100 million dollars at risk

Affected Protocol	Funds ($) At Risk
MakerDAO	~5M
Enzyme	~1M
Abracadabra	~100M
TribeDAO	~20M
Opyn	~6M

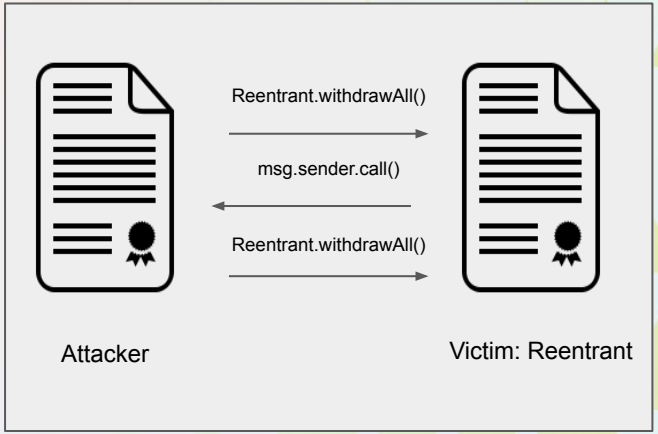
What is Reentrancy

- Execution is interrupted e.g. ETH or ERC777 transfers
- The state has not been fully updated
- The control flow is passed to another contract
- DAO hack: One of the most famous attacks!
- We are usually concerned with entry points that modify the state!

What is Reentrancy

```
contract Reentrant {
    …
    mapping (address => uint256) private userBalances;
    uint256 totalSupply;

    function withdrawAll() external {
        uint256 balance = userBalances[msg.sender];
        require(balance > 0, "Insufficient balance");
        totalSupply -= balance;
        (bool success, ) = msg.sender.call{value: balance}("");
        require(success, "Failed to send Ether");
        userBalances[msg.sender] = 0;
    }
    …
}
```

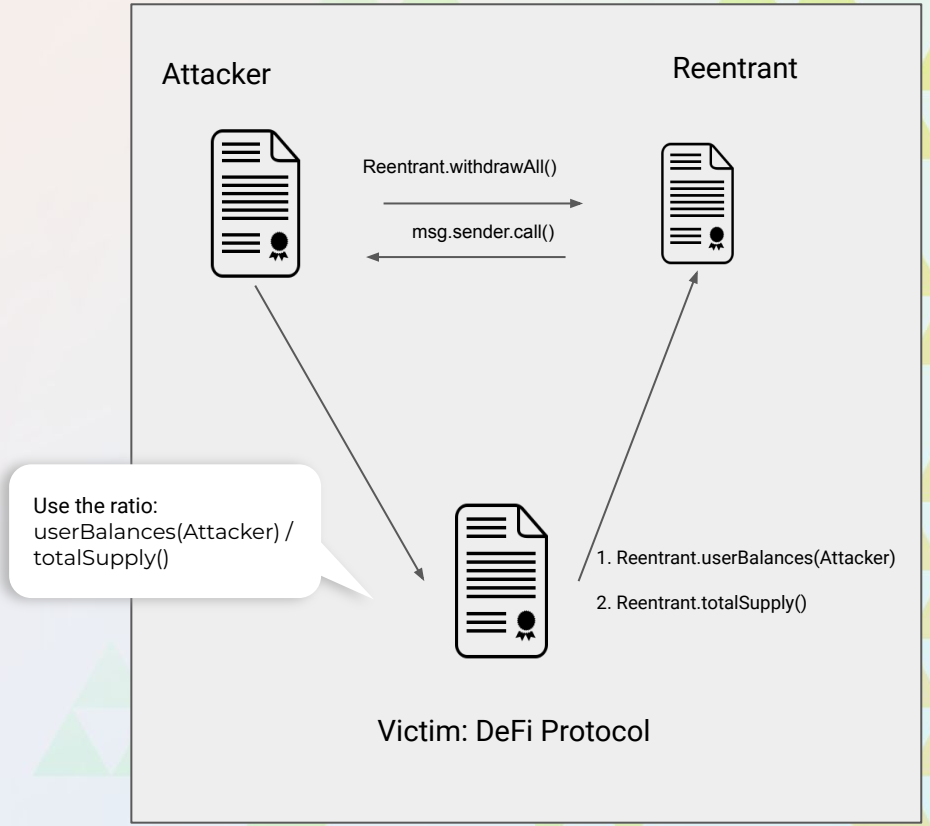


What is read-only Reentrancy

```
contract Reentrant {
    …
    bool private lock;
    mapping (address => uint256) public userBalances;
    uint256 public totalSupply;

    modifier nonReentrant() {
           require(!lock);
           lock = true;
           _;
           lock = false;
    }

    function withdrawAll() external nonReentrant {
           uint256 balance = userBalances[msg.sender];
           require(balance > 0, "Insufficient balance");
           totalSupply -= balance;
           (bool success, ) = msg.sender.call{value: balance}("");
           require(success, "Failed to send Ether");
           userBalances[msg.sender] = 0;
    }
    …
}
```



Curve.fi: StableSwapSTETH

The pool holds ETH (native) and stETH (ERC20)

```
@nonreentrant(lock)
def remove_liquidity(_amount: uint256,_min_amounts: uint256[N_COINS]) ->
uint256[N_COINS]:
    …
    CurveToken(lp_token).burnFrom(msg.sender, _amount)
    …
    for i in range(N_COINS):
        …
     …
        if i == 0:
            raw_call(msg.sender, b"", value=value)
        else:
            …
    …
```

The token_supply of the lp_token is modified but not all the balances have been updated.

```
def get_virtual_price() -> uint256:
    …
    D: uint256 = self.get_D(self._balances(), self._A())
    …
    return D * PRECISION / token_supply
```

get_virtual_price() depends on the balances and the token_supply

Final thoughts

- The storage update is not yet finalized
- We just READ the state and make a decision based on it!
- Reentrancy locks for state changing functions is NOT enough!
- For new protocols: The view functions should revert if the lock is taken or make the lock public
- For the rest: try to call a function with non-reentrant modifier

Thank you!

Ioannis Sachinoglou
ChainSecurity
ioannis.sachinoglou@chainsecurity.com

Non-technical read

Technical read