TECHFUND Audit Score

Smart contract Audit Report

OASYS Audit

Completed on 2022-12-02

Score **POSITIVE**

Risk level		
	Critical	0
	High	0
	Medium	1
	Low	2
	Note	2

Risk level detail

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

The tester arrives at the likelihood and impact estimates, they can now combine them to get a final severity rating for this risk. Note that if they have good business impact information, they should use that instead of the technical impact information.

https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

Vulnerability Review

	Number of warnings
Compiler Version	0
State Variable Default Visibility	0
In-Correct Estimation of Rewards in Validator	1
Costly Loops for external calls	0
Integer Overflow / Underflow	0
Parity Multisig Bug	0
Callstack Depth Attack	0
Error in setting up Dynamic Arrays	0
In-sufficient return check while staking / unstacking	2
Re-Entrancy (out of order events)	2
Double Withdrawal	0

In-Correct Estimation of Rewards in Validator

1

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
In the case of JavaScript, following issue happens due to floating point errors. In Solidity we don't have floating points, but we get a similar issue and we can get "rounding errors".
For example : In Solidity the second operation would actually yield 2999. By doing all our multiplications first, we mitigate rounding related issues as much as possible. Hence it is suggested to do all multiplication before division otherwise the output can be in-correct.

```
» console.log((30 * 100 * 13) / 13)
  3000
← undefined
» console.log((30 / 13) * 100 * 13)
  2999.9999999999995
← undefined
»
```

```
function getRewards(
    IStakeManager.Validator storage validator,
    IEnvironment.EnvironmentValue memory env,
    uint256 epoch
) internal view returns (uint256 rewards) {
    if (isInactive(validator, epoch) || isJailed(validator, epoch)) return 0;

    uint256 _stake = getTotalStake(validator, epoch);

    if (epoch >= 73) {
        if (_stake < env.validatorThreshold) return 0;
        rewards = (_stake * 2612) / 1e7;
    } else {
        if (_stake == 0) return 0;

        rewards =
            (_stake * Math.percent(env.rewardRate, Constants.MAX_REWARD_RATE, Constants.REWARD_PRECISION)) /
            10**Constants.REWARD_PRECISION;
        if (rewards == 0) return 0;

        rewards *= Math.percent(
            env.blockPeriod * env.epochPeriod,
            Constants.SECONDS_PER_YEAR,
            Constants.REWARD_PRECISION
        );
        rewards /= 10**Constants.REWARD_PRECISION;
    }

    uint256 slashes = validator.slashes[epoch];
    if (slashes > 0) {
        uint256 blocks = validator.blocks[epoch];
        rewards = Math.share(rewards, blocks - slashes, blocks, Constants.REWARD_PRECISION);
    }
    return rewards;
}
```

Because of the above issue in Validator, the reward count will not be correctly calculated. It is highly suggested to do the multiplications first and then proceed to do the division.

In-sufficient return check while staking / unstacking

2

In validator, the return value of **“stake”** and **“unstakeV2”** call is not checked. Sometimes due to unforeseen circumstances, call may fail and return false. Currently such scenario is not handled in the stake. It is suggested to ensure that the return value is checked.

```
function stake(
    IStakeManager.Validator storage validator,
    uint256 epoch,
    address staker,
    uint256 amount
) internal {
    if (!validator.stakerExists[staker]) {
        validator.stakerExists[staker] = true;
        validator.stakers.push(staker);
    }
    validator.stakeUpdates.add(validator.stakeAmounts, epoch, amount);
}
```

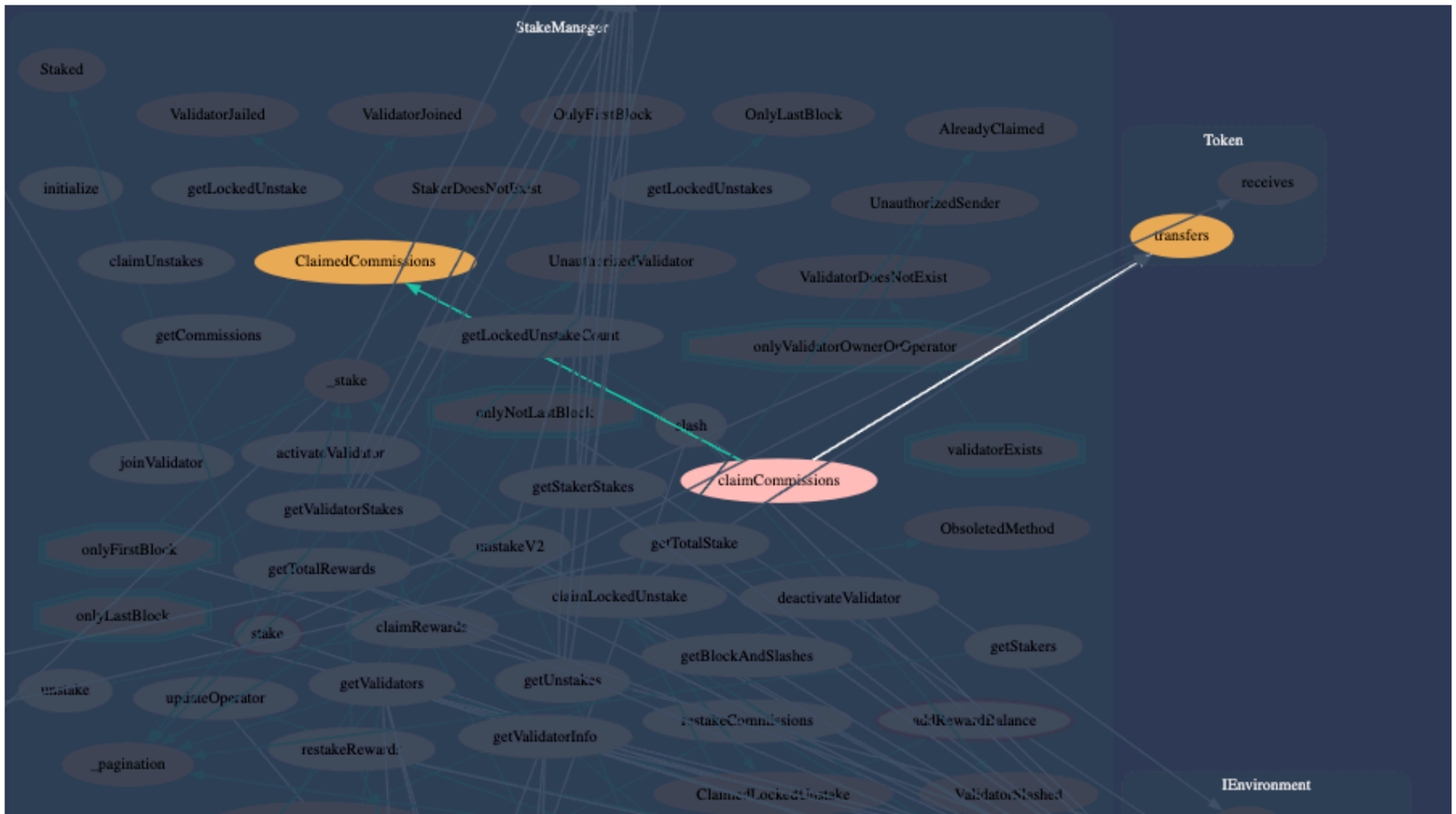
```
function unstakeV2(
    address validator,
    Token.Type token,
    uint256 amount
) external validatorExists(validator) stakerExists(msg.sender) onlyNotLastBlock {
    Staker storage _staker = stakers[msg.sender];

    amount = _staker.unstake(environment, validators[validator], token, amount);
    if (amount == 0) revert NoAmount();

    _staker.lockedUnstakes.push(LockedUnstake(token, amount, block.timestamp + 10 days));
    stakeUpdates.sub(stakeAmounts, environment.epoch() + 1, amount);

    emit UnstakedV2(msg.sender, validator, _staker.lockedUnstakes.length - 1);
}
```

Re-Entrancy (out of order events) 2

A state variable is changed or an event is emitted after a contract uses external value. The attacker may use a fallback function to execute the vulnerable function again, before the state variable is changed or event is emitted.

That being said, not every reentrant behaviour can be called a bug. It can turn into a critical security issue if two conditions are met:
1) One contract calls another one while the former hasn't updated its state yet. This can be accidentally enabled by adding a function that makes an external call to another untrusted contract before the state changes get executed.
2) Once the malicious actor gains control over the untrusted contract, they can recursively call the original function over and over again, draining crypto wallets or sneaking into the code to make unsafe changes.

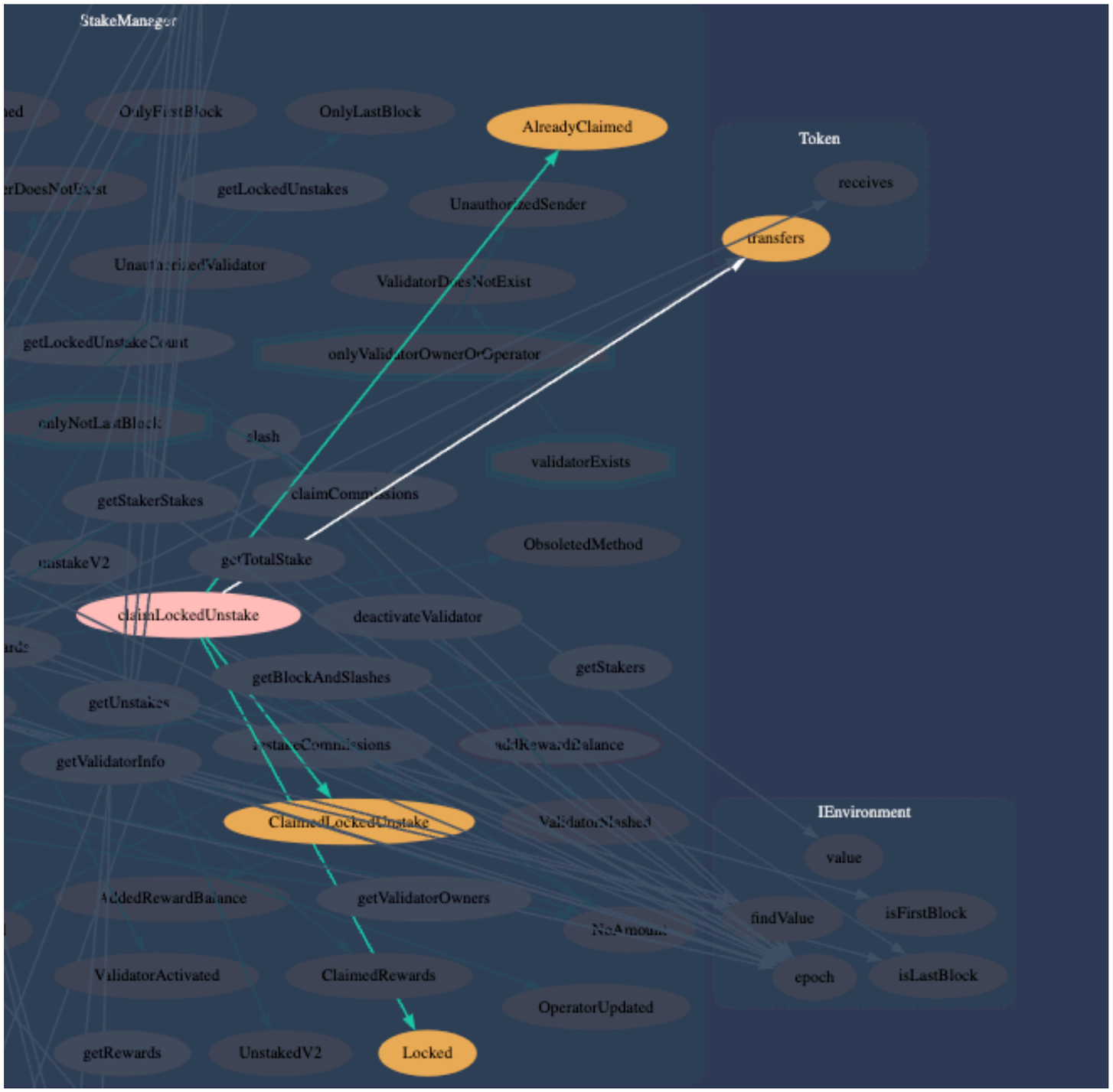
As a part of preventative techniques please ensure that all state changes happen before calling external contracts and use function modifiers that prevent re-entrancy. Although it is not a critical issue hence it has been mentioned as low level issue.



StakeManager.claimCommissions has event update after making external calls. First it calls transfers and after that calls ClaimedCommissions.
If transfers.() re-enters, the claimCommissions events will be shown in an incorrect order, which might lead to issues for third parties.

Re-Entrancy (out of order events)

2



Similarly in **claimLockedUnstake** it calls external transfers function. After the transfers is finished ClaimedLockedUnstake is emitted.
If transfers.() re-enters, the **claimLockedUnstake** events will be shown in an incorrect order, which might lead to issues for third parties.