CYBER UNIT

Cyber Security
Strategic Partner

Dusk Smart Contract Audit

Date: August 5, 2020

[redacted]

This document may contain confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the customer, or it can disclose publicly after all vulnerabilities are fixed – upon the decision of the customer.

Scope and Code Revision Date

Link	https://github.com/dusk-network/prestaking-contract.git
Date	05.08.2020

Table of contents

[redacted]

Introduction

This report presents the findings of the security assessment of Customer` s smart contract and its code review conducted between 30th of July 2020 – 5th of August 2020.

Scope

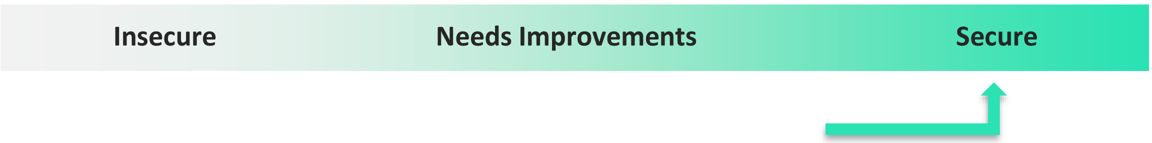
The scope of the project is DUSK smart contract, which can be found here:
https://github.com/dusk-network/prestaking-contract.git

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the widely known vulnerabilities that considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Dusk smart contracts are secure. In the initial audit, three risks were found which cannot lead to token loss. These risks were fixed by the Dusk team. The overall code quality is good.



Our team performed an analysis of code functionality, manual audit and automated checks with Slither and remixed IDE (see Appendix B pic 1-2). All issues found during automated investigation manually reviewed and application vulnerabilities presented in the Audit overview section. A general overview presented in the AS-IS section and all encountered matters can found in the Audit overview section.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss.
High	High-level vulnerabilities are difficult to exploit. However, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium-level vulnerabilities are essential to fix; however, they can't lead to tokens loss.
Low	Low-level vulnerabilities are mostly related to outdated or unused code snippets.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can generally be ignored.

AS-IS overview

Prestaking contract consists of the next smart contracts:

1. SafeMath, IERC20, SafeERC20, Ownable, contracts - standard OpenZeppelin smart contracts for tokens known as etoken 2.
2. Prestaking contract - implementation of OpenZeppelin Ownable, SafeERC20 smart contract

Contracts from point 1 compared to original OpenZeppelin templates and no logic differences found. It's considered secure.

Prestaking contract functional implementation:

1. Prestaking contract inherits Ownable
2. startWithdrawReward,startWithdrawStake,withdrawStake, withdrawReward, withdrawStake

Prestaking contract init function called with the following parameters:

- token – DUSK token address
- min – MIN AMOUNT
- max – MAX AMOUNT
- reward – REWARD

latestVersion set at the moment of review.

Note: Contract testing in production is out-of-scope of the current security review.

Audit overview

Critical

No critical vulnerabilities found.

High (fixed)

1. ~~Use of block.timestamp in a constructor without validation.~~

   ~~In Ethereum, the current timestamp must always be higher than the previous timestamp.~~

Medium

No medium severity vulnerabilities found.

Low (fixed)

1. ~~Performing a multiplication on the result of a division, distributeRewards functions (see Appendix A pic. 2 for evidence).~~
2. ~~For loop over dynamic array~~

   ~~withdrawStake,~~

   ~~distributeRewards,~~

   ~~updateStakingPool~~

~~functions (see Appendix A pic. 3,4,5 for evidence).~~

~~Gas Limit and Loops that do not have a fixed number of iterations, such as loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. Either explicitly or just due to regular operation, the number of iterations in a loop can grow beyond the block gas limit, which can cause the complete contract to stalled at a certain point.~~

~~Solidity integer division will truncate. As a result, we are performing a multiplication before a division, which might lead to loss of precision.~~

Lowest / Code style / Best Practice (Fixed)

1. ~~DUSK contract has different functions with similar names (however, they have different arguments). These functions are transferToICAP. Consider renaming the features, so all services have different names.~~

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality presented in As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found one high and two low vulnerabilities, which were fixed. Other code/features were added upon the completion of the review which was not part of the scope of the audited code.

Disclaimer

The smart contracts given for audit have analyzed following the best industry practices at the date of this report, concerning: cybersecurity vulnerabilities and issues in smart contract source code, the details of which disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

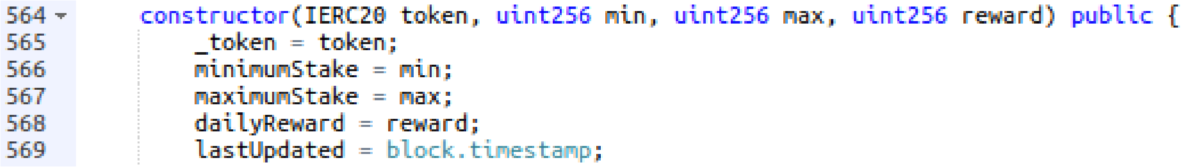
The audit doesn't make warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the system, bug free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is essential to note that you should not rely on this report only. We recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee specific security of the audited smart contracts.

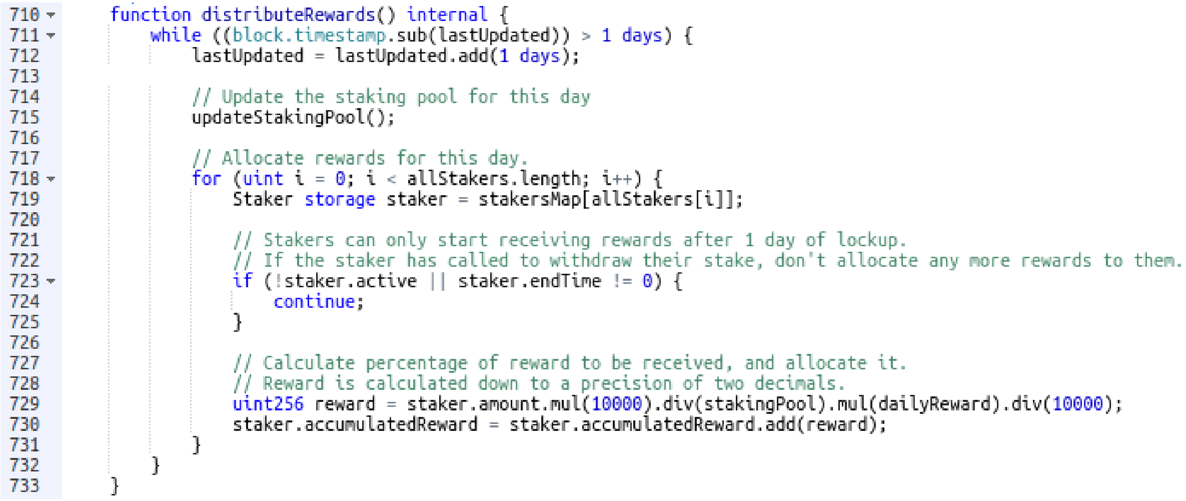
Appendix A. Evidences

Pic 1. Use of block.timestamp in a constructor without validation:



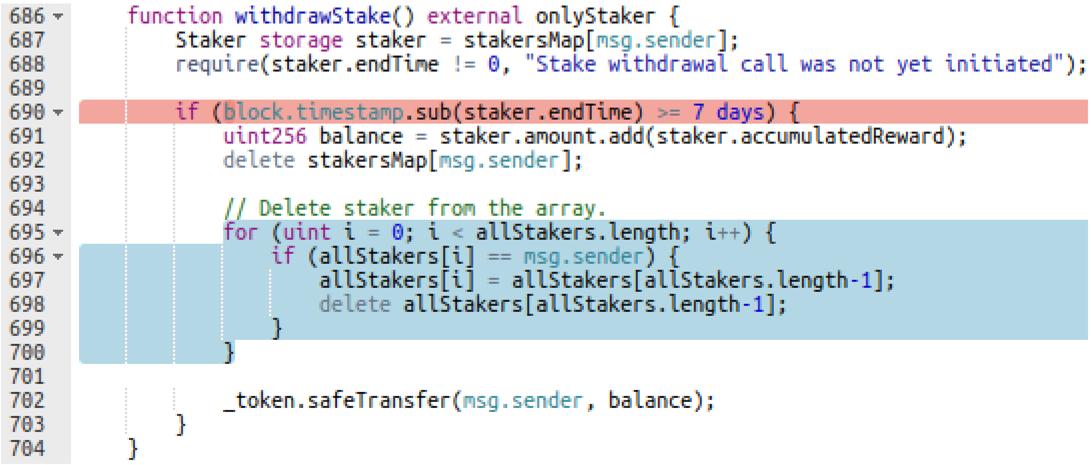
```
constructor(IERC20 token, uint256 min, uint256 max, uint256 reward) public {
    _token = token;
    minimumStake = min;
    maximumStake = max;
    dailyReward = reward;
    lastUpdated = block.timestamp;
```

Pic 2. Performs a multiplication on the result of a division:



```
function distributeRewards() internal {
    while ((block.timestamp.sub(lastUpdated)) > 1 days) {
        lastUpdated = lastUpdated.add(1 days);

        // Update the staking pool for this day
        updateStakingPool();

        // Allocate rewards for this day.
        for (uint i = 0; i < allStakers.length; i++) {
            Staker storage staker = stakersMap[allStakers[i]];

            // Stakers can only start receiving rewards after 1 day of lockup.
            // If the staker has called to withdraw their stake, don't allocate any more rewards to them.
            if (!staker.active || staker.endTime != 0) {
                continue;
            }

            // Calculate percentage of reward to be received, and allocate it.
            // Reward is calculated down to a precision of two decimals.
            uint256 reward = staker.amount.mul(10000).div(stakingPool).mul(dailyReward).div(10000);
            staker.accumulatedReward = staker.accumulatedReward.add(reward);
        }
    }
}
```

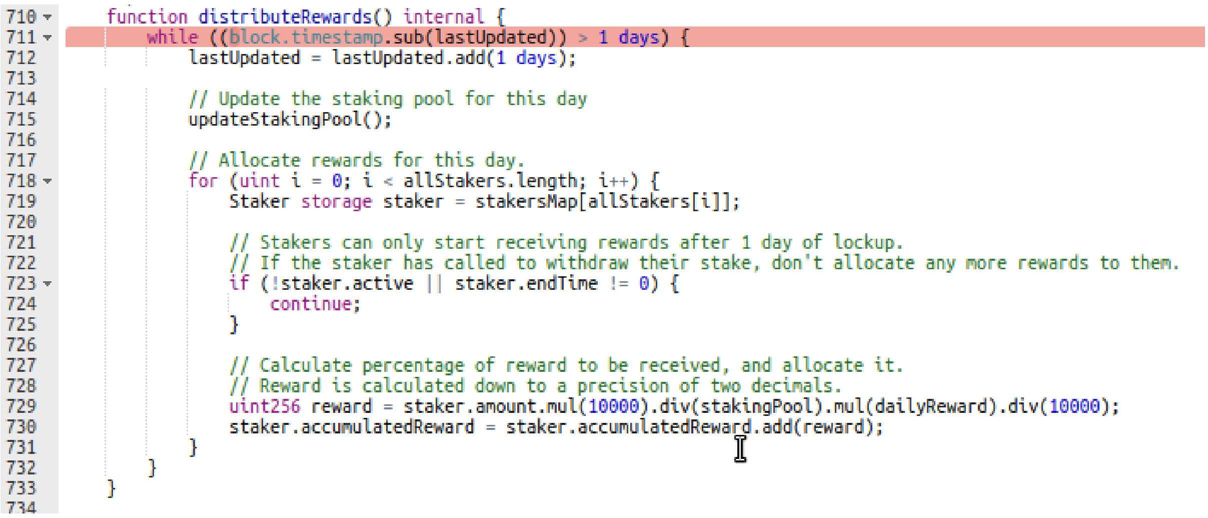
Pic 3. Performs a multiplication on the result of a division:



```
function withdrawStake() external onlyStaker {
    Staker storage staker = stakersMap[msg.sender];
    require(staker.endTime != 0, "Stake withdrawal call was not yet initiated");

    if (block.timestamp.sub(staker.endTime) >= 7 days) {
        uint256 balance = staker.amount.add(staker.accumulatedReward);
        delete stakersMap[msg.sender];

        // Delete staker from the array.
        for (uint i = 0; i < allStakers.length; i++) {
            if (allStakers[i] == msg.sender) {
                allStakers[i] = allStakers[allStakers.length-1];
                delete allStakers[allStakers.length-1];
            }
        }

        _token.safeTransfer(msg.sender, balance);
    }
}
```

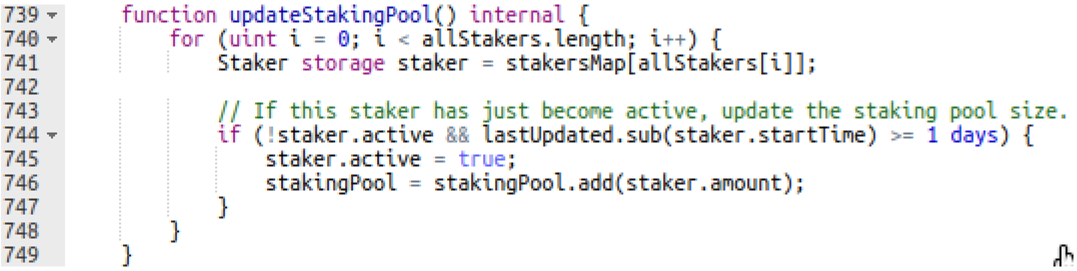
Pic 4. Performs a multiplication on the result of a division:

[redacted]



```
function distributeRewards() internal {
    while ((block.timestamp.sub(lastUpdated)) > 1 days) {
        lastUpdated = lastUpdated.add(1 days);

        // Update the staking pool for this day
        updateStakingPool();

        // Allocate rewards for this day.
        for (uint i = 0; i < allStakers.length; i++) {
            Staker storage staker = stakersMap[allStakers[i]];

            // Stakers can only start receiving rewards after 1 day of lockup.
            // If the staker has called to withdraw their stake, don't allocate any more rewards to them.
            if (!staker.active || staker.endTime != 0) {
                continue;
            }

            // Calculate percentage of reward to be received, and allocate it.
            // Reward is calculated down to a precision of two decimals.
            uint256 reward = staker.amount.mul(10000).div(stakingPool).mul(dailyReward).div(10000);
            staker.accumulatedReward = staker.accumulatedReward.add(reward);
        }
    }
}
```

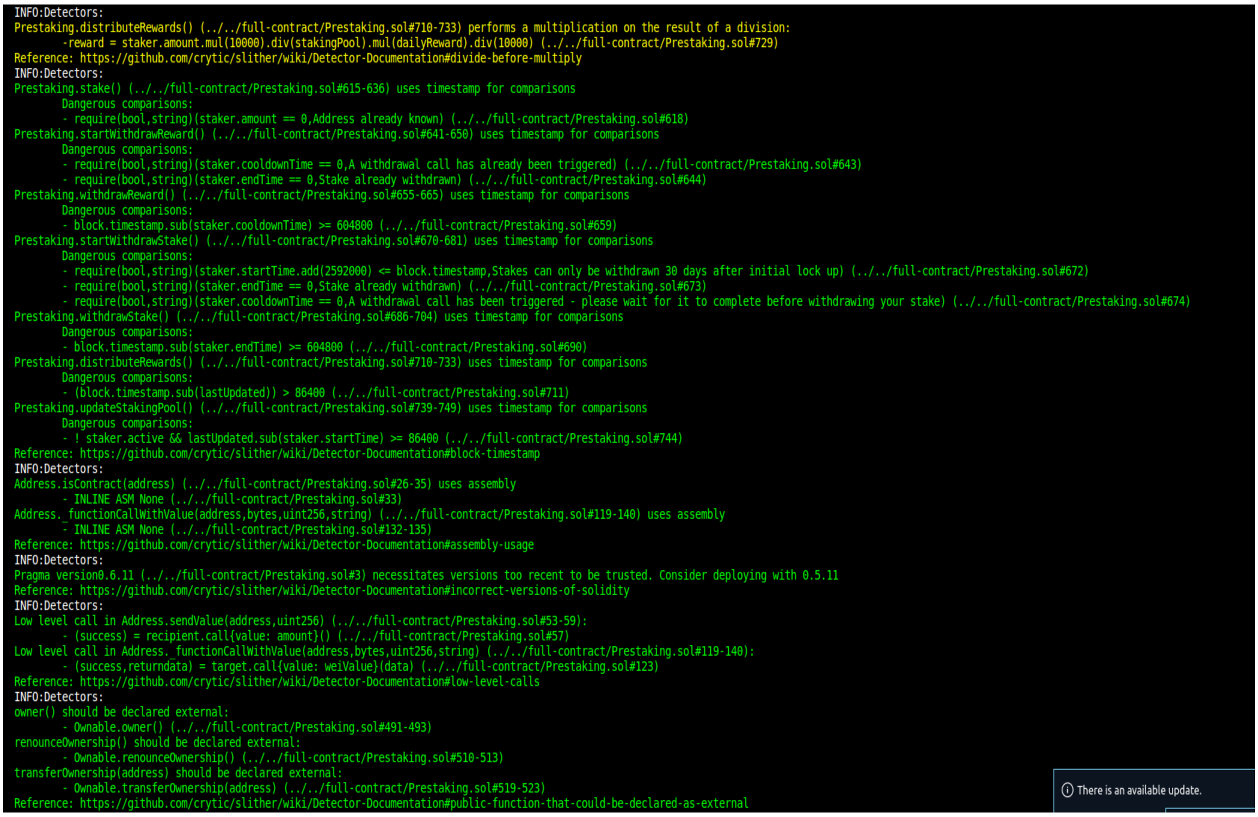
Pic 5. Performs a multiplication on the result of a division:



```
function updateStakingPool() internal {
    for (uint i = 0; i < allStakers.length; i++) {
        Staker storage staker = stakersMap[allStakers[i]];

        // If this staker has just become active, update the staking pool size.
        if (!staker.active && lastUpdated.sub(staker.startTime) >= 1 days) {
            staker.active = true;
            stakingPool = stakingPool.add(staker.amount);
        }
    }
}
```

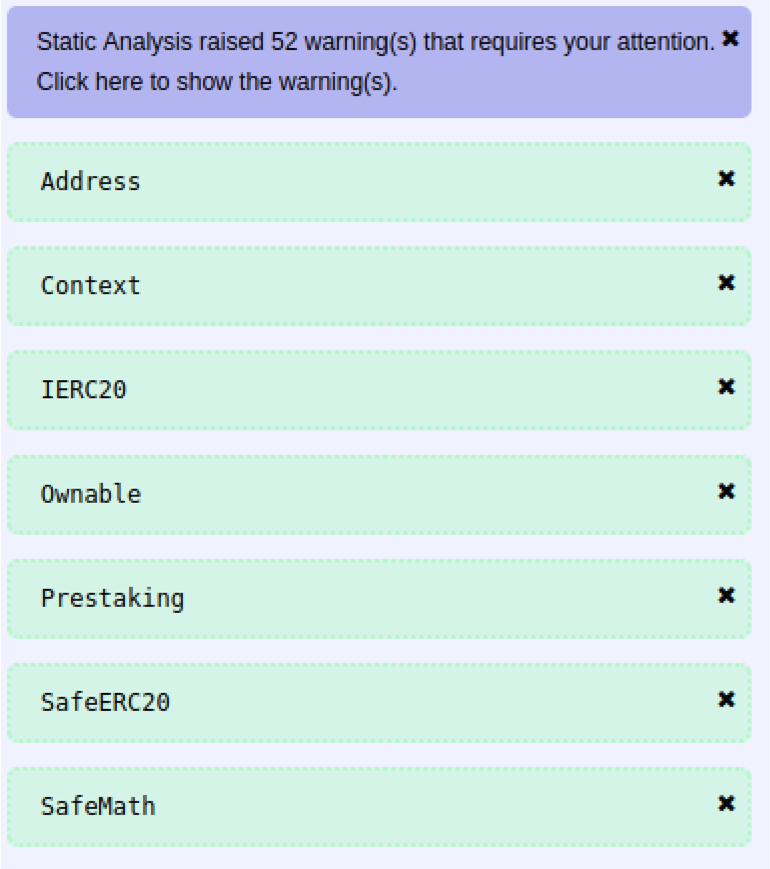
Appendix B. Automated tools report

Pic 1. Slither automated report:



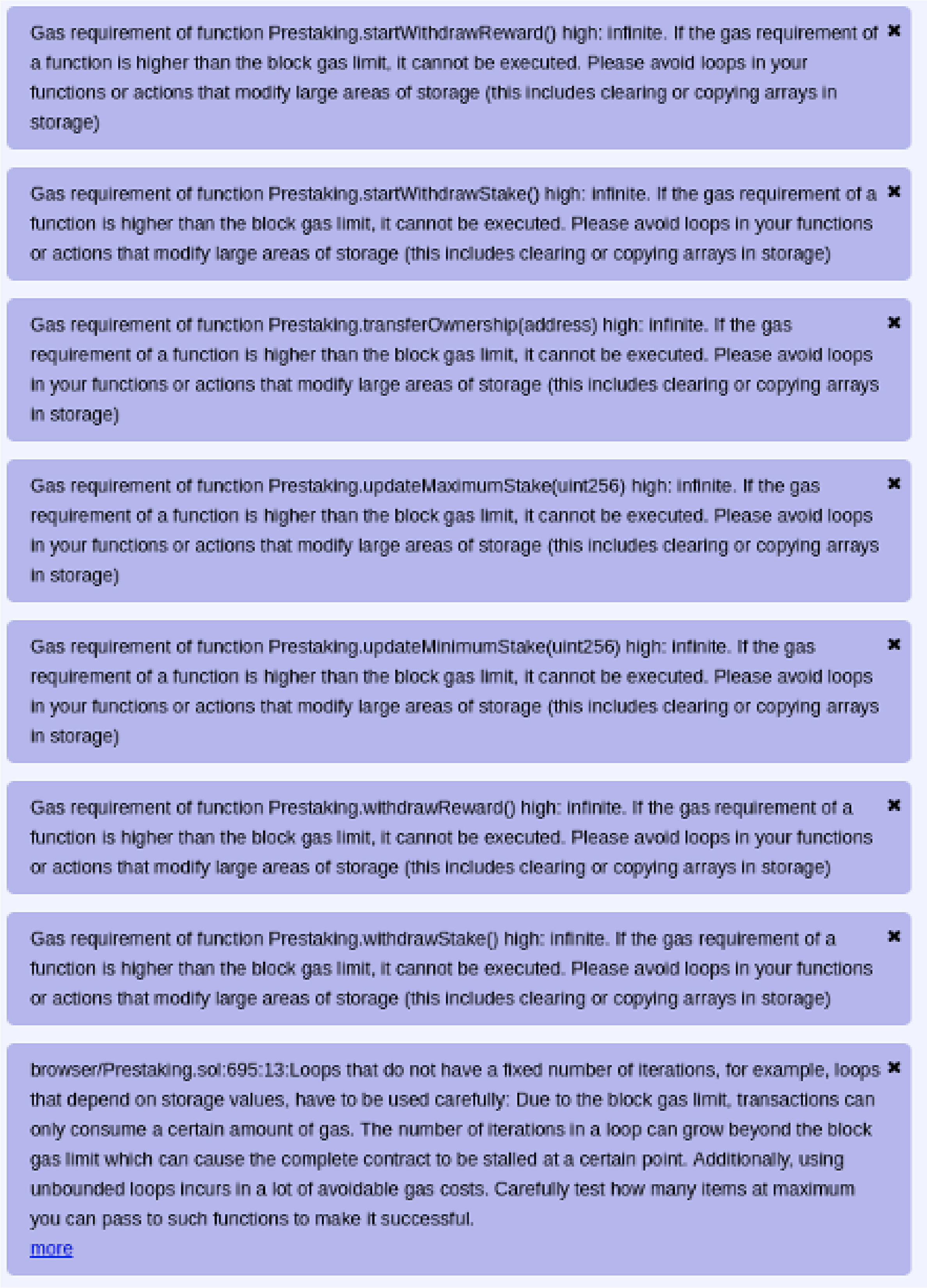
```
INFO:Detectors:
Prestaking.distributeRewards() (../../full-contract/Prestaking.sol#710-733) performs a multiplication on the result of a division:
        -reward = staker.amount.mul(10000).div(stakingPool).mul(dailyReward).div(10000) (../../full-contract/Prestaking.sol#729)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Prestaking.stake() (../../full-contract/Prestaking.sol#615-636) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(staker.amount == 0,Address already known) (../../full-contract/Prestaking.sol#618)
Prestaking.startWithdrawReward() (../../full-contract/Prestaking.sol#641-650) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(staker.cooldownTime == 0,A withdrawal call has already been triggered) (../../full-contract/Prestaking.sol#643)
        - require(bool,string)(staker.endTime == 0,Stake already withdrawn) (../../full-contract/Prestaking.sol#644)
Prestaking.withdrawReward() (../../full-contract/Prestaking.sol#655-665) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.sub(staker.cooldownTime) >= 604800 (../../full-contract/Prestaking.sol#659)
Prestaking.startWithdrawStake() (../../full-contract/Prestaking.sol#670-681) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(staker.startTime.add(2592000) <= block.timestamp,Stakes can only be withdrawn 30 days after initial lock up) (../../full-contract/Prestaking.sol#672)
        - require(bool,string)(staker.endTime == 0,Stake already withdrawn) (../../full-contract/Prestaking.sol#673)
        - require(bool,string)(staker.cooldownTime == 0,A withdrawal call has been triggered - please wait for it to complete before withdrawing your stake) (../../full-contract/Prestaking.sol#674)
Prestaking.withdrawStake() (../../full-contract/Prestaking.sol#686-704) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.sub(staker.endTime) >= 604800 (../../full-contract/Prestaking.sol#690)
Prestaking.distributeRewards() (../../full-contract/Prestaking.sol#710-733) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp.sub(lastUpdated)) > 86400 (../../full-contract/Prestaking.sol#711)
Prestaking.updateStakingPool() (../../full-contract/Prestaking.sol#739-749) uses timestamp for comparisons
        Dangerous comparisons:
        - ! staker.active && lastUpdated.sub(staker.startTime) >= 86400 (../../full-contract/Prestaking.sol#744)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (../../full-contract/Prestaking.sol#26-35) uses assembly
        - INLINE ASM None (../../full-contract/Prestaking.sol#33)
Address._functionCallWithValue(address,bytes,uint256,string) (../../full-contract/Prestaking.sol#119-140) uses assembly
        - INLINE ASM None (../../full-contract/Prestaking.sol#132-135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version0.6.11 (../../full-contract/Prestaking.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (../../full-contract/Prestaking.sol#53-59):
        - (success) = recipient.call{value: amount}() (../../full-contract/Prestaking.sol#57)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (../../full-contract/Prestaking.sol#119-140):
        - (success,returndata) = target.call{value: weiValue}(data) (../../full-contract/Prestaking.sol#123)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner() (../../full-contract/Prestaking.sol#491-493)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (../../full-contract/Prestaking.sol#510-513)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (../../full-contract/Prestaking.sol#519-523)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
```

Pic 2. RemixIDE automated report:



Static Analysis raised 52 warning(s) that requires your attention. ✖
Click here to show the warning(s).

```
Address ✖
Context ✖
IERC20 ✖
Ownable ✖
Prestaking ✖
SafeERC20 ✖
SafeMath ✖
```



Gas requirement of function Prestaking.startWithdrawReward() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function Prestaking.startWithdrawStake() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function Prestaking.transferOwnership(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function Prestaking.updateMaximumStake(uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

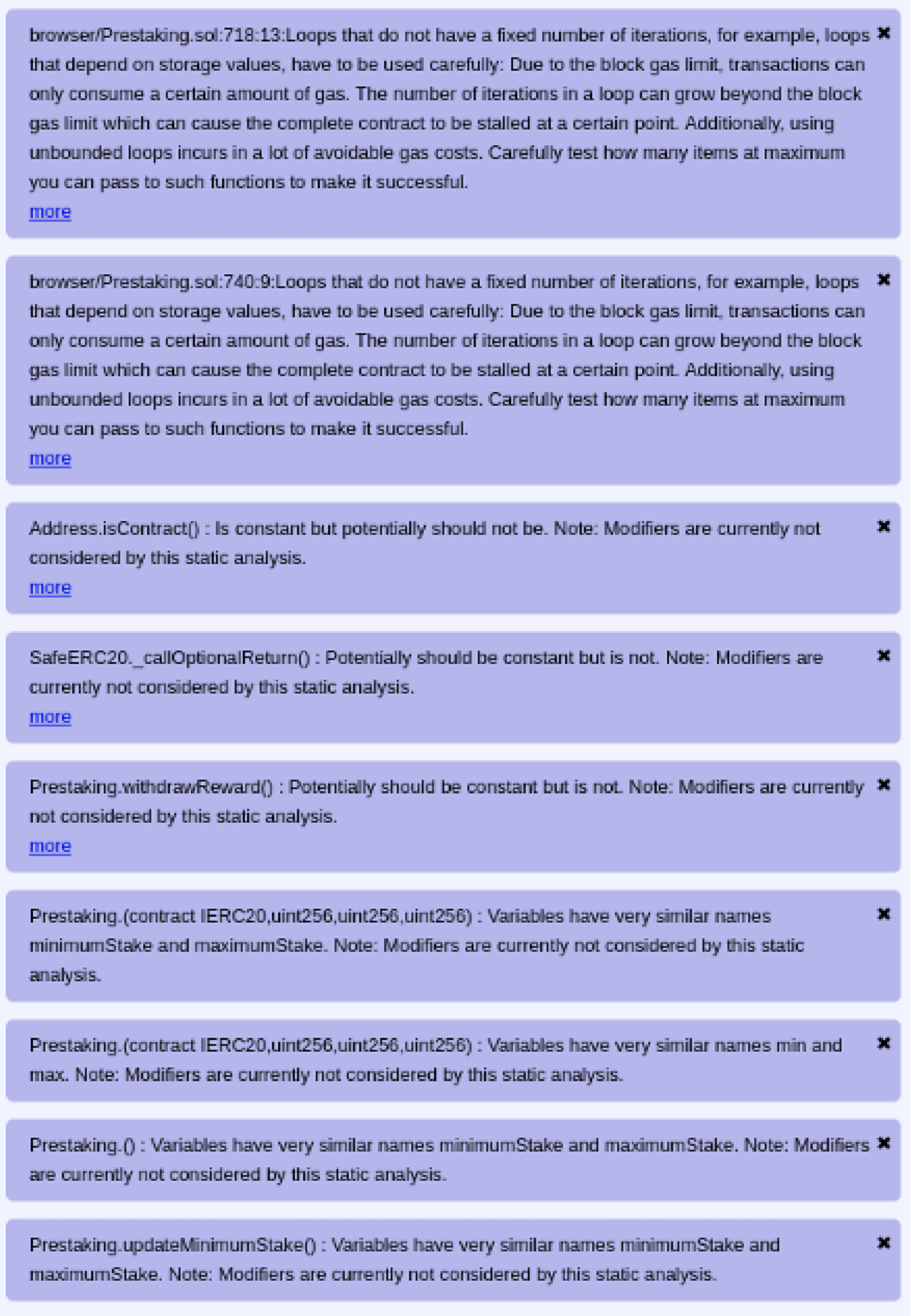
Gas requirement of function Prestaking.updateMinimumStake(uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function Prestaking.withdrawReward() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function Prestaking.withdrawStake() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

browser/Prestaking.sol:695:13:Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully: Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more

[redacted]



browser/Prestaking.sol:718:13:Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully: Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more

browser/Prestaking.sol:740:9:Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully: Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more

Address.isContract() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more

SafeERC20._callOptionalReturn() : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis.
more

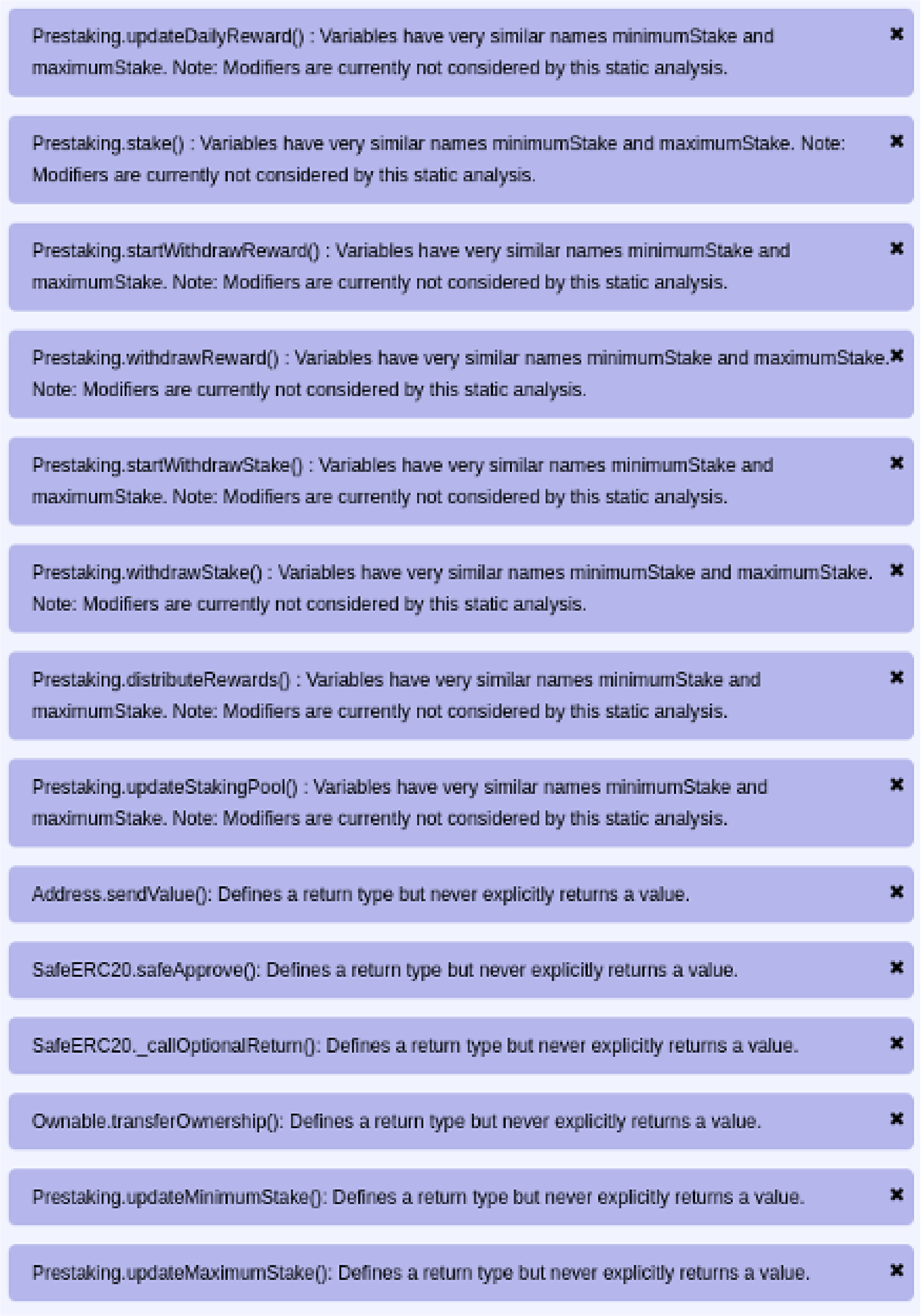
Prestaking.withdrawReward() : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis.
more

Prestaking.(contract IERC20,uint256,uint256,uint256) : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis.

Prestaking.(contract IERC20,uint256,uint256,uint256) : Variables have very similar names min and max. Note: Modifiers are currently not considered by this static analysis.

Prestaking.() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis.

Prestaking.updateMinimumStake() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis.



Prestaking.updateDailyReward() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Prestaking.stake() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Prestaking.startWithdrawReward() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Prestaking.withdrawReward() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Prestaking.startWithdrawStake() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Prestaking.withdrawStake() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Prestaking.distributeRewards() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Prestaking.updateStakingPool() : Variables have very similar names minimumStake and maximumStake. Note: Modifiers are currently not considered by this static analysis. ✖

Address.sendValue(): Defines a return type but never explicitly returns a value. ✖

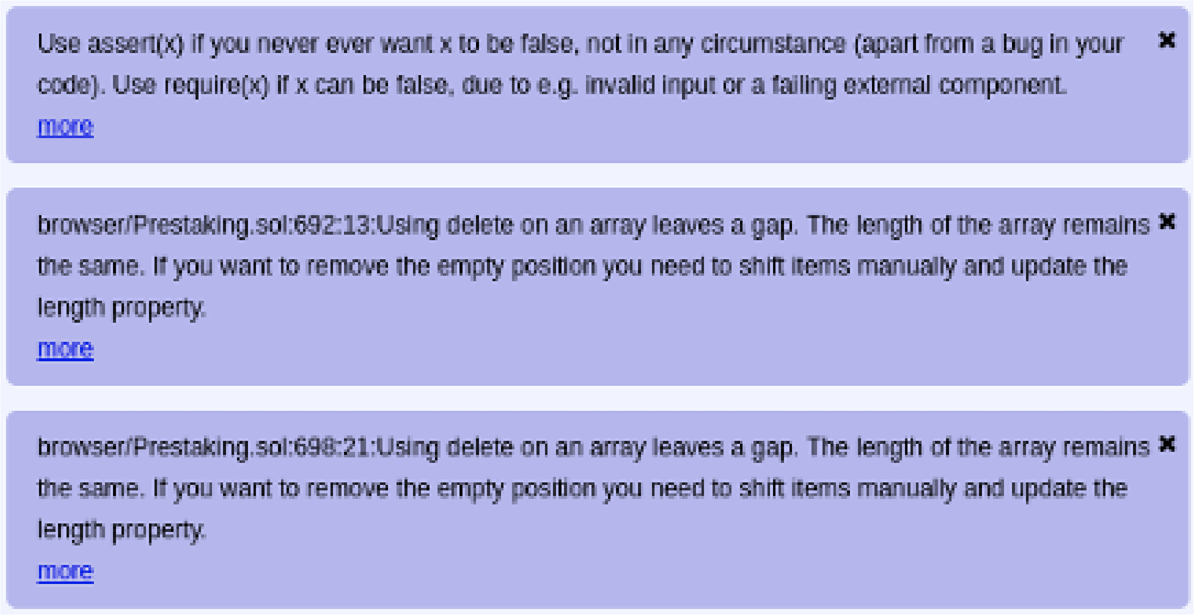
SafeERC20.safeApprove(): Defines a return type but never explicitly returns a value. ✖

SafeERC20._callOptionalReturn(): Defines a return type but never explicitly returns a value. ✖

Ownable.transferOwnership(): Defines a return type but never explicitly returns a value. ✖

Prestaking.updateMinimumStake(): Defines a return type but never explicitly returns a value. ✖

Prestaking.updateMaximumStake(): Defines a return type but never explicitly returns a value. ✖



Use assert(x) if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use require(x) if x can be false, due to e.g. invalid input or a failing external component.
more

browser/Prestaking.sol:692:13:Using delete on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the length property.
more

browser/Prestaking.sol:698:21:Using delete on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the length property.
more