



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For DragonSwap (Airdrop & Staker)

02 July 2025



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 General	6
1.3.2 AirdropFactory	6
1.3.3 Airdrop	6
1.3.4 Staker	7
2 Findings	8
2.1 General	8
2.1.1 Issues & Recommendations	9
2.2 AirdropFactory	10
2.2.1 Privileged Functions	10
2.2.1 Issues & Recommendations	11
2.3 Airdrop	12
2.3.1 Privileged Functions	12
2.3.2 Issues & Recommendations	13
2.4 Staker	17
2.4.1 Privileged Functions	17
2.4.2 Issues & Recommendations	18

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for DragonSwap's airdrop and staked contracts on the SEI network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	DragonSwap
URL	https://dragonswap.app
Platform	SEI
Language	Solidity
Preliminary Contracts	https://github.com/dragonswap-app/dragonswap-airdrop-and-staker/tree/99cc3a475a2dd095c099eabb1e7db480b7be425c
Resolution 1	https://github.com/dragonswap-app/dragonswap-airdrop-and-staker/commit/9bdd10640cdfa4311f180959255f52711b211e4a

1.2 Contracts Assessed

Name	Contract	Live Code Match
AirdropFactory		
Airdrop		
Staker		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	0	-	-	-
● Medium	2	1	-	1
● Low	7	3	-	4
● Informational	11	8	1	2
Total	20	12	1	7

Classification of Issues

Severity	Description
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 General

ID	Severity	Summary	Status
01	LOW	Both Airdrop and Staker contracts do not work with tokens with a fee on transfer or tokens that rebase	ACKNOWLEDGED

1.3.2 AirdropFactory

ID	Severity	Summary	Status
02	INFO	Typographical issues	PARTIAL

1.3.3 Airdrop

ID	Severity	Summary	Status
03	LOW	Double-spend edge case is possible	ACKNOWLEDGED
04	INFO	Owner can assign more portions than the available deposits for distribution	ACKNOWLEDGED
05	INFO	Withdraw signature is missing expiry	✓ RESOLVED
06	INFO	toWallet parameter might cause user portion to get stuck and seek admin action or new signature	✓ RESOLVED
07	INFO	Fee precision in Airdrop could differ from fee precision in the configured Staker contract	✓ RESOLVED
08	INFO	Typographical issues	✓ RESOLVED

1.3.4 Staker

ID	Severity	Summary	Status
09	MEDIUM	Reward sniping when <code>totalDeposits == 0</code> lets the first new staker seize the entire idle rewards	ACKNOWLEDGED
10	MEDIUM	Reentrancy reward tokens allow attacker to drain all Staker contract funds	✓ RESOLVED
11	LOW	Flash-loan "stake → reward → withdraw" siphons windfall rewards when <code>fee < reward share</code>	ACKNOWLEDGED
12	LOW	Owner can confiscate pending rewards via <code>removeRewardToken()</code> + <code>sweep()</code>	ACKNOWLEDGED
13	LOW	<code>pendingRewards()</code> will show incorrect overly inflated price if stake is claimed	✓ RESOLVED
14	LOW	User's stake and rewards may remain locked if they are blacklisted by one reward token	✓ RESOLVED
15	LOW	Sweep function DoS in current implementation	✓ RESOLVED
16	INFO	<code>claimEarnings()</code> and <code>withdraw()</code> allow for potential loss of rewards	✓ RESOLVED
17	INFO	The minimum deposit should be set during constructor	✓ RESOLVED
18	INFO	<code>claimEarnings()</code> loads all user stakes in memory resulting in increasing gas consumption and possible block gas limit DOS	✓ RESOLVED
19	INFO	Fee changes get applied to present unlocked stakes	ACKNOWLEDGED
20	INFO	Typographical issues	✓ RESOLVED



2 Findings

2.1 General

The issues in this section apply to several contracts across the protocol. Please go through them carefully and implement the proper resolutions in the relevant contracts.



2.1.1 Issues & Recommendations

Issue #01	Both Airdrop and Staker contracts do not work with tokens with a fee on transfer or tokens that rebase
Severity	 LOW SEVERITY
Description	Both Airdrop and Staker contracts do not use a before/after balance when moving funds in/out of the contract which will cause issues if a token is a rebase token or a token with fee on transfer.
Recommendation	Consider never accepting such tokens by documenting such behavior.
Resolution	 ACKNOWLEDGED

2.2 AirdropFactory

AirdropFactory creates minimal proxy clones of Airdrop implementations using the EIP-1167 standard. It allows the owner to deploy multiple airdrop instances with different configurations (token, staker, treasury, signer, timestamps) while sharing the same implementation logic to save gas.

The factory maintains a registry of all deployed instances, tracks which implementation each deployment uses, and provides utility functions to query deployment history and verify if an airdrop was created through this factory.

2.2.1 Privileged Functions

- `setImplementation()`
- `deploy()`
- `transferOwnership()`
- `renounceOwnership()`



2.2.1 Issues & Recommendations

Issue #02	Typographical issues
Severity	INFORMATIONAL
Description	<p><u>L74</u></p> <pre>if (!success) revert();</pre> <p>The revert in the deploy function should revert with a custom error for easy debugging in case this fails.</p> <p>—</p> <p>Inconsistent return variable — sometimes the return variables are named (e.g. deploy) and sometimes they are not (e.g. getLatestDeployment)</p> <pre>function deploy(address token, address staker, address treasury, address signer, address _owner, uint256[] calldata timestamps) external onlyOwner returns (address instance) { ... function getLatestDeployment() external view returns (address) { —</pre> <p>Maximum fee is calculated every time instead of just being introduced as a constant.</p> <p>Introduce a MAX_FEE variable:</p> <pre>uint256 private constant MAX_FEE = feePrecision * 9 / 10;</pre> <p>And use it like this:</p> <pre>if (_fee > MAX_FEE) revert InvalidValue();</pre>
Recommendation	Consider fixing the issues.
Resolution	PARTIALLY RESOLVED



2.3 Airdrop



Airdrop is an upgradeable contract of a token distribution system that manages vested airdrops with time-locked unlocking periods. It allows users to withdraw their allocated tokens either directly to their wallet (with a penalty fee) or to a connected staker contract (penalty-free), using signature-based authorization for withdrawals. The contract supports multiple unlock timestamps, owner-managed portion assignments, and includes a cleanup mechanism for unclaimed tokens after a 60-day buffer period following the final unlock.



2.3.1 Privileged Functions



- `lockUp()`
- `addTimestamp()`
- `changeTimestamp()`
- `assignPortions()`
- `deposit()`
- `cleanUp()`
- `transferOwnership()`
- `renounceOwnership()`



2.3.2 Issues & Recommendations

Issue #03	Double-spend edge case is possible
Severity	 LOW SEVERITY
Description	<p>If a user is assigned a portion and the owner wants to change the portion to a different value (let's say they want to increase the airdrop), then the user can frontrun the owner's call, withdraw the initially assigned portion (if he has a signature) and then be assigned the new portion. After that, they can withdraw the new full portion too when the intention of the owner was to just increase the portion.</p> <p>This has a very low chance of occurring since the user has to have a signature to withdraw (keep in mind that signatures can be reused currently) and the owner has to want to change the portion of the user (or call <code>assignPortions()</code> for that account twice).</p>
Recommendation	Consider increasing the portion instead of assigning a value.
Resolution	 ACKNOWLEDGED
	The client stated: "If portions are finalized/locked and then signatures get enabled the issue will not be exploitable."

Issue #04 Owner can assign more portions than the available deposits for distribution	
Severity	 INFORMATIONAL
Description	<p>When the owner deposits funds for distribution, <code>totalDepositedForDistribution</code> tracks the total deposit amount. This variable however is not updated upon a withdrawal.</p> <p>When assigning portions, the owner is not limited by the deposited assets for distribution which could lead to a situation where users could compete to withdraw before the funds dry up.</p>
Recommendation	Consider reducing <code>totalDepositedForDistribution</code> in the withdraw and cleanup function and consider introducing a variable that tracks the assigned total amount of portions to users. This new variable could be used in <code>assignPortions()</code> to make the function revert if more portions are assigned than available.
Resolution	 ACKNOWLEDGED

Issue #05 Withdraw signature is missing expiry	
Severity	 INFORMATIONAL
Description	<p>It is considered best practice to have a signature expiry parameter for a signature to prevent unexpected signature executions after a long time.</p> <p>Here, it might be desirable to introduce one in order to make the double spend edge case (Issue #03) more difficult since the signature will be usable multiple times only until it expires (on a separate note signatures could be made executable only once).</p>
Recommendation	Consider introducing a signature expiry.
Resolution	 RESOLVED

Severity	 INFORMATIONAL
Description	<p>The toWallet parameter is included in the signature which can cause airdrop withdrawal to revert if toWallet is set to false and the portion total amount is below the staker minimumDeposit value which is hardcoded to 100e18.</p> <p>If the param is not included in the signature, the user can change it to true and withdraw the airdrop.</p>
Recommendation	Consider removing toWallet if you think such cases could occur.
Resolution	 RESOLVED

Issue #07	Fee precision in Airdrop could differ from fee precision in the configured Staker contract
Severity	 INFORMATIONAL
Description	<p>This is not an issue since the Staker contract in scope of this audit has the same precision as the Airdrop contract.</p> <p>However, it would be considered good practice if we use the fee value from the Staker contract to use its precision as well because after some time the Airdrop contract could be used with a different Staker contract and this could become problematic.</p>
Recommendation	Consider introducing a new view function in the Staker contract that will return the penaltyAmount. The same function could then be reused in Staker to calculate the fee amount there.
Resolution	 RESOLVED

Issue #08	Typographical issues
Severity	<div data-bbox="454 165 486 197" data-label="Image"></div> INFORMATIONAL
Description	<p>penaltyWallet and penaltyStaker are not used and can be removed.</p> <p>—</p> <p>Unclear treasury handling in withdraw and cleanup. The contract has a parameter called treasury but in withdraw, treasury is fetched from the Staker contract.</p>
Recommendation	Consider fixing the issues.
Resolution	<div data-bbox="454 685 486 716" data-label="Image"></div> RESOLVED



2.4 Staker



Staker enables users to deposit staking tokens and earn multiple reward tokens proportionally to their stake. Users can choose to lock their stakes for 30 days to avoid withdrawal fees, or stake unlocked but pay a configurable fee to the treasury upon withdrawal.

The contract supports multiple individual stakes per user, uses a reward-per-share mechanism to distribute accumulated rewards from various reward tokens, and allows the owner to dynamically add or remove reward tokens. It includes emergency withdrawal functionality and sweep capabilities for non-reward tokens.

2.4.1 Privileged Functions

- `setTreasury()`
- `setFee()`
- `addRewardToken()`
- `removeRewardToken()`
- `sweep()`
- `transferOwnership()`
- `renounceOwnership()`

2.4.2 Issues & Recommendations

Issue #09	Reward sniping when <code>totalDeposits == 0</code> lets the first new staker seize the entire idle rewards
Severity	 MEDIUM SEVERITY
Description	<p>When all users withdraw and <code>totalDeposits</code> drops to 0, incoming reward tokens are ignored because <code>_updateAccumulated()</code> short-circuits:</p> <pre>if (rewardBalance == _lastRewardBalance _totalDeposits == 0) return;</pre> <p>Every reward that is added during the idle period can be snipped by the next depositor.</p>
Recommendation	As there is no easy solution for this, consider always having a stake with a controlled address that will redirect the idle reward to the treasury to be redistributed or redirect rewards to the treasury if deposits are 0.
Resolution	 ACKNOWLEDGED <p>The client stated: "The sweeping flow will not work here since it is working only for non-reward tokens. Always having a stake will mitigate the issue."</p>

Description

There are a couple of sensitive sections of `withdraw()` that allow for this exploit to happen.

The first is deleting the `rewardDebt` (<https://github.com/dragonswap-app/dragonswap-airdrop-and-staker/blob/99cc3a475a2dd095c099eabb1e7db480b7be425c/src/Staker.sol#L257>), not executing `_stake.claimed = true;` before the second loops start sending out the reward tokens and one reward token being an ERC777 or a reentrancy on transfer one.

Imagine we have as reward tokens [token 1, token 2, token 3, token 4 that is a reentrancyToken].

When an attacker calls `withdraw`, the rewards for token 1, 2 and 3 are sent while also setting their `rewardDebt` to 0.

Then it is now time for sending the rewards of the reentrancy token to the attacker. A call in the `transfer` function calls the attacker (assuming it is a smart contract or a smart account via EIP-7702) and he reenters into `claimEarnings()`.

Then the interesting part is that now the attacker will not just claim his rewards but all the accumulated rewards per share over time because `rewardDebt` for each token is now 0:

```
uint256 pending = accumulated -  
rewardDebt[rewardDebtHash];
```

If the staking has been going for some time, then accumulated rewards per share will be a large value and there will be an attempt to send the whole token balance to the attacker.

Even if it attempts to send a bigger amount than the contract holds, the call will not revert since `_payout` will reduce the amount to the available funds:

```
function _payout(IERC20 token, uint256 pending) private {
    uint256 currRewardBalance =
token.balanceOf(address(this));
    uint256 rewardBalance = token == stakingToken ?
currRewardBalance - totalDeposits : currRewardBalance;
    uint256 amount = pending > rewardBalance ? rewardBalance
: pending;


    lastRewardBalance[address(token)] -= amount;
    token.safeTransfer(msg.sender, amount);
    emit Payout(msg.sender, token, pending);
}
```

After the funds of all reward tokens are now with the attacker, they can send as many tokens as needed to the Staker contract in order for the `withdraw()` call to finish execution without reverting.

Recommendation	Move <code>_stake.claimed = true;</code> above the rewards loop distribution and put reentrancy guard modifiers on all Staker contract related functions.
-----------------------	---

Resolution



Issue #11**Flash-loan "stake → reward → withdraw" siphons windfall rewards when fee < reward share****Severity** LOW SEVERITY**Description**


As `stake()` allows unlock-on-arrival (`locking == false`) and `_updateAccumulated()` only runs when a user function is invoked, an attacker can run the following atomic sequence:

1. Flash-loan & stake a huge amount F
2. Front-run / sandwich a large reward top-up R
3. Immediately withdraw the stake
4. `_updateAccumulated()` attributes $\approx R \times F / \text{totalDeposits}$ to the attacker ($\approx R$ because $F \gg$ existing stake)
5. Withdrawal fee is charged only on principal: $\text{feeAmount} = F \times \text{fee} / 10\,000$
6. Profit in the same transaction: $\text{profit} = \text{pendingRewards} - \text{feeAmount} - \text{flashLoanFee}$.



Whenever $R > F \times \text{fee} / 10\,000$, the attacker exits with a positive, risk-free gain while honest stakers receive a diluted share of R .

Recommendation



It is hard to mitigate this issue without changing a lot of logic. One way to solve it is to charge the withdrawal fee on principal + harvested rewards to eliminate easy profit but that would yield less rewards for honest users.



Resolution ACKNOWLEDGED

The client commented: "By distributing rewards in smaller portions this can reduce the impact. One way to mitigate the issue is by distributing rewards based on the amount of time the user has staked, similar to the Synthetix StakingRewards contract. Here is a simplified example implementation: <https://solidity-by-example.org/defi/staking-rewards/>"

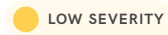
Issue #12	Owner can confiscate pending rewards via <code>removeRewardToken()</code> + <code>sweep()</code>
Severity	 LOW SEVERITY
Description	<p>After users have accrued rewards in a given reward token, the owner can call <code>removeRewardToken(token)</code> to delete the token from <code>rewardTokens[]</code>, preventing any further <code>_payout()</code> calls for that asset. The owner may then invoke <code>sweep(token, to)</code> and transfer the entire residual balance, including users' unclaimed rewards.</p>
Recommendation	<p>Consider reverting if a reward token has accumulated rewards that have not been claimed.</p> <p>An alternative would be to ensure that when the owner removes a reward token, they do not remove one that still has unclaimed rewards.</p> <p>We understand the team wishes to simply acknowledge this issue if they want to have the ability to remove malicious reward tokens even if they have unclaimed rewards.</p>
Resolution	 ACKNOWLEDGED



Issue #13	pendingRewards() will show incorrect overly inflated price if stake is claimed
Severity	 LOW SEVERITY
Description	<p>If stake is claimed, pendingRewards() will act as if it is not and will accrue rewards if necessary, and thus calculate the pending rewards of the stake via this calculation:</p> <pre>stake.amount * accRewardTokenPerShare) / accumulatedPrecision - rewardDebt;</pre> <p>where rewardDebt will be 0 since the rewards would be already distributed for this stake in withdraw().</p> <p>This will cause the pendingRewards value to be very large since the whole accumulated price per share will be used.</p>
Recommendation	Consider reverting or returning 0 if stake is marked as claimed.
Resolution	 RESOLVED

Issue #14	User's stake and rewards may remain locked if they are blacklisted by one reward token
Severity	 LOW SEVERITY
Description	If one reward token blacklists the user, then this user will not be able to call withdraw() successfully since msg.sender is used as the recipient of the funds.
Recommendation	Consider introducing a receiver parameter to the withdraw() function.
Resolution	 RESOLVED

Severity



Description

Due to the first check and staking token being set as a reward token, this function will revert every time the admin tries to sweep the staking token.

```
/**
 * @notice Sweep token to the to address
 * @param token The address of the token to sweep
 * @param to The address that will receive token balance
 */
function sweep(IERC20 token, address to) external onlyOwner
{
    if (isRewardToken[address(token)]) revert();

    uint256 balance = token.balanceOf(address(this));
    if (token == stakingToken) {
        unchecked {
            balance -= totalDeposits;
        }
    }
    if (balance == 0) revert();
    //@audit an address zero check here might be a good idea
    token.safeTransfer(to, balance);
    emit Swept(address(token), to, balance);
}
```



Recommendation



Change the function as follows:

```
function sweep(IERC20 token, address to) external onlyOwner
{
    uint256 balance = token.balanceOf(address(this));
    if (token == stakingToken) {
        unchecked {
            balance -= totalDeposits;
        }
    }
    else {
        if (isRewardToken[address(token)]) revert();
    }
    if (balance == 0) revert();
    token.safeTransfer(to, balance);
    emit Swept(address(token), to, balance);
}
```

This way, if the token is the stakingToken, no check would be applied, but if it is, the check will be applied. This means all the tokens that are not reward tokens + the staking tokens itself can be swept, ensuring mistaken transfers or locked stakes can be saved.

Resolution

Issue #16	claimEarnings() and withdraw() allow for potential loss of rewards
Severity	 INFORMATIONAL
Description	<p>_payout() sends out as many funds are available currently in the contract (even if that means sending less than it is supposed to) but claimEarnings() and withdraw() update rewardDebt as if the whole reward was claimed.</p> <p>If somehow the Staker contract does not have enough rewards to distribute (which could happen in case of a vulnerability), users that call either of these functions will lose their rewards from an accounting point of view.</p>
Recommendation	Consider reverting if there are not enough funds.
Resolution	 RESOLVED

Issue #17	The minimum deposit should be set during constructor
Severity	 INFORMATIONAL
Description	<p>The minimum deposit should not be hardcoded, should be immutable and set during constructor as every stake token would have its own value. This value is set to 100e18, and it is hard coded in the contract.</p> <p>Furthermore, if the staking token is with a lower decimal count (6 for example), 100e18 would be a very large minimal amount.</p>
Recommendation	Consider having this variable being set via the constructor.
Resolution	 RESOLVED

Issue #18**claimEarnings() loads all user stakes in memory resulting in increasing gas consumption and possible block gas limit DOS****Severity** INFORMATIONAL**Description**

claimEarnings() load all user stakes via in memory:

```
Stake[] memory _stakes = stakes[msg.sender];
```

This results in increased gas consumption for the user that calls claimEarnings.

A test showed that a call to claimEarnings() costs:

- 58_278 gas with 2 stakes
- 652_877 gas with 1000 stakes
- 2_031_829 gas with 3000 stakes

There is a chance of a DOS due to block gas limit because an attacker could create many stakes on behalf of the user and grief him for this particular function.

This however, requires the deposit token to be very cheap and the minimal deposit amount to be very low so it seems unlikely for somebody to attempt this but still should have it in mind.

Recommendation

Avoid loading all stakes in memory and instead access the stake inside the loop via stakes[msg.sender][stakeIndex].

Resolution RESOLVED

Severity

 INFORMATIONAL

Description

Users might decide not to lock if the fee is set to a low value, but after some time, if the fee changes to a high value then this might be unpleasant for them and the user could decide to lock earlier to avoid paying the fees.

Recommendation

This would be an issue if it was interest rates instead of fees. Consider saving the fee in the stake struct to decide how much penalty fees to charge stakers on withdrawal. However, if admins want to have more direct control this recommendation might not be desirable.

Resolution

 ACKNOWLEDGED

Severity

 INFORMATIONAL

Description

The Stake struct can be optimized by declaring `unlockTimestamp` as `uint64` so it can be packed together with `claimed`.

The Payout event emits `pending` instead of `amount`, which means that if `amount` is actually lower than `pending`, the event logs an incorrect amount.

```
uint256 amount = pending > rewardBalance ? rewardBalance :  
pending;
```

The sweep reverts with a simple `revert()` if it is a reward token; consider reverting with a specific error.

Make sure all `stake()` parameters are documented.

The function param comments above `withdraw()` are incorrect.

Recommendation Consider fixing the issues.

Resolution

 RESOLVED

The client added the receiver address param to a couple of functions. We would like to remind them to update the function documentation comments as well.



PALADIN
BLOCKCHAIN SECURITY