PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For WINR Protocol (GensisWlpStaking)

26 April 2023

paladinsec.co info@paladinsec.co

Table of Contents

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 advocation 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 WINR Protocol's Genesis WLP Staking contract on the Arbitrum 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	WINR Protocol
URL	https://winr.games/
Platform	Arbitrum
Language	Solidity
Preliminary Contracts	https://github.com/WINRLabs/winr-protocol/blob/45effb06c8e92e579b4a1f9ea5f6466d0863b263/contracts/stakings/gWLPStaking/GenesisWlpStaking.sol
Resolution 1	https://github.com/WINRLabs/winr-protocol/blob/0be8f291a76deb9077492faa6f540d1c6a563b04/contracts/stakings/gWLPStaking/GenesisWlpStaking.sol
Resolution 2	https://github.com/WINRLabs/winr-protocol/blob/1a7294f5d91181213eee5bbf0510034b08c0fcb1/contracts/stakings/gWLPStaking/GenesisWlpStaking.sol
Resolution 3	https://github.com/WINRLabs/winr-protocol/blob/03acc0efc5092806442ab2044a0f0503f8ba2b69/contracts/stakings/gWLPStaking/GenesisWlpStaking.sol
Resolution 4	https://github.com/WINRLabs/winr-protocol/blob/c69b7ba8f6d46760c674b7ca793f16027a293ca8/contracts/stakings/gWLPStaking/GenesisWlpStaking.sol
Resolution 5	https://github.com/WINRLabs/winr-protocol/blob/d47ded79df8900bdd9227ffb08669a21262e8f25/contracts/stakings/gWLPStaking/GenesisWlpStaking.sol

1.2 Contracts Assessed

Name	Contract	Live Code Match
`GensisWlpStaking`		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● **High**	5	5	-	-
● **Medium**	0	-	-	-
● **Low**	4	3	-	1
● **Informational**	3	1	1	1
Total	**12**	**9**	**1**	**2**

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 with 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 GenesisWlpStaking

ID	Severity	Summary	Status
01	HIGH	Governance risk: Governance can break the contract in multiple ways and take out the underlying tokens	✓ RESOLVED
02	HIGH	Users can lose deposits if they deposit a second time	✓ RESOLVED
03	HIGH	Pool reward logic is fundamentally flawed resulting in wrong and excessive rewards	✓ RESOLVED
04	HIGH	It is possible that the `firstBlock` value resets after everyone withdraws, restarting reward emissions	✓ RESOLVED
05	HIGH	`rewardPerBlock` is incorrectly calculated in `fundReward` and `updateMaxBlock`	✓ RESOLVED
06	LOW	`withdraw` does not adhere to checks-effects-interactions which means if `harvest` were to allow for reentrancy, the WLP balance could be drained	✓ RESOLVED
07	LOW	Contract business logic can break significantly if deposits are re-enabled on the `gWLP`	ACKNOWLEDGED
08	LOW	withdraw does not reset `wlpPerToken` which prevents a secondary deposit after withdrawal from accumulating WLP rewards	✓ RESOLVED
09	LOW	Arbitrum L2 block frequency might be quite irregular	✓ RESOLVED
10	INFO	Typographical errors	✓ RESOLVED
11	INFO	Lack of validation	ACKNOWLEDGED
12	INFO	Gas optimizations	PARTIAL

2 Findings

2.1 GenesisWlpStaking

`GenesisWlpStaking` is a staking contract which implements a staking pool for the Genesis WINR LP (`gWLP`) token. This token represents the initial contributions to the genesis WINR LP pool. These contributions by early backers will eventually be redeemable to WINR LP through this contract, when the contributors withdraw.

Users can deposit `gWLP` tokens into the pool, and receive rewards in return. The rewards are given in `vWINR` and `WLP` tokens, which can be claimed by calling the `harvest()` function. The `vWINR` token rewards are based on time passed and emitted gradually to all active stakers in the pool. The `WLP` reward amount is however not based on time, but instead based on collected fees. These collected fees can be pulled into the genesis staking contract through calling `claimWLP`. Whenever a user harvests, it grants any unclaimed rewards.

The contract also has a few other functions for setting addresses and variables. The team can also change the duration of the genesis emissions freely, and adjust the `vWINR` amount and therefore the reward rate.

Although users stake `gWLP`, they will not be able to withdraw it. Instead, they can eventually withdraw real WLP tokens once the team activates this.

It should also be noted that the WINR team is capable of removing any token from this contract except for the `gWLP` token. This might be a risk for investors, especially when the team turns malicious or the keys become compromised.

2.1.1 Privileged Functions

- `setAddresses [ DEFAULT_ADMIN_ROLE ]`
- `setWithdrawable [ DEFAULT_ADMIN_ROLE ]`
- `fundReward [ DEFAULT_ADMIN_ROLE ]`
- `updateMaxBlock [ DEFAULT_ADMIN_ROLE ]`
- `fundWLP [ DEFAULT_ADMIN_ROLE ]`
- `recoverToken [ DEFAULT_ADMIN_ROLE ]`
- `grantRole [ DEFAULT_ADMIN_ROLE ]`
- `revokeRole [ DEFAULT_ADMIN_ROLE ]`
- `renounceRole [ role bearer ]`

2.1.1 Issues & Recommendations

Issue #01	Governance risk: Governance can break the contract in multiple ways and take out the underlying tokens
Severity	HIGH SEVERITY

Description	It is important for users to trust the governance wallets. This is because the Winr team has given themselves significant privileges within this contract. Such privileges are often acceptable if the team is trusted, but they do come with risks for the users. Even if the team is trusted, their keys might get compromised (stolen) and user funds could still get lost.
	
	Users should assume that their deposits are fully in the hands of the Winr team, and trusting said team is critical.
	
	Line 82
	`function setAddresses(IERC20 _WLP, IClaimWlp _claimWlp) external onlyGovernance {`
	
	The team can change the withdrawable WLP token and claim-address at any time. Changing these addresses to bad ones has a significant impact: It would break all withdrawals. Furthermore, not setting them would also prevent withdrawals. Setting WLP to vWINR or gWLP would furthermore break critical assumptions within the contract.
	
	We recommend at the very least that the team prevents setting this address to gWLP and vWINR, and recommend that it can only be set once.
	
	Line 92
	`function setWithdrawable(bool _isWithdrawable) external onlyGovernance {`
	
	We do not understand why withdrawals should be able to be disabled again. It might make sense to only ever allow this boolean to be set (eg. remove the input boolean and just set it to true).

Line 274

```
function recoverToken(IERC20 _token, uint256 _amount)
external onlyGovernance {
```

Every token except for `gWLP` can be withdrawn using `recoverToken`. However, even `gWLP` can be stolen through advanced governance exploits by setting `WLP = gWLP`. This might be an acceptable privilege however given that the governance is needed to be trusted anyway within this contract.

—

It is often assumed that there are enough balances of all tokens within the contract. This is presently not strictly enforced due to the various governance functions. This means that `withdraw` might start reverting even if there is just insufficient reward tokens.

—

It should be noted that even if all governance risks are mitigated, users would still need to place some trust in the team given that the `gWLP` token represents a promise from the team to make it exchangeable to `WLP`. This means that it might be acceptable to leave in some flexibility as there is no way to make this contract "trustless".

Recommendation Consider patching up any unnecessary governance risks. We understand that being able to pull contract value is a desired feature in case of emergencies. Consider placing all roles that are able to do this behind a carefully chosen multi-signature wallet.

Resolution	✓ RESOLVED The recommended steps to improve input validation for governance functions were implemented. However, it should be noted that a `burnAddress` was introduced and the gWLP tokens are no longer locked in the contract, but instead are directly sent to the `burnAddress` upon deposit. If the `burnAddress` is set to an active account address (i.e., someone has access to the private key of this account), then the `burnAddress` will be able to drain the entire staking contract WLP balance because of the standard double-spending problem. We recommend making the `burnAddress` a constant instead of immutable and set it to a value like `address(0)`. Finally, we would like to remind the users that this contract will always have inherent governance risks due to the reliance on the team for supplying WLP tokens. It should be noted that the client made a few more fixes to the contract, in which some logic was redesigned and a new governance risk was introduced. The governance can drain the `vWinr` balance by calling `fundReward` with an incorrect emission rate. Users should would therefore still need to place their trust in the team, especially if the `vWinr` contract balance is a concern to them.

Issue #02	Users can lose deposits if they deposit a second time
Severity	● HIGH SEVERITY
Location	Line 106 `staker.amount = _amount;`
Description	Within the `GenesisWlpStaking` contract, users can stake their whole `gWLP` token balance at once by calling the `deposit()` function. The `Winr` team clearly envisions that everyone will only ever call this function once. However, it is possible that a user purchased `gWLP` with multiple addresses and decides to stake the balance of their first address, then transfer in the secondary balance and stake it as well under the first address. In this scenario, they call `deposit` twice. The issue is that as the developer did not envision this scenario, the second deposit will in fact override the first one. Instead, it should add it to the first one. This means that in our example the user completely loses their first deposit and any accumulated rewards.
Recommendation	Consider incrementing the amount: `staker.amount += _amount;` It might also make sense to do a `harvest` before the second deposit to avoid losing accumulated rewards. However, we understand if this is not done because this secondary deposit is an unlikely scenario and it might be acceptable to void rewards in favor of keeping the code simple.
Resolution	✓ RESOLVED An `exist` boolean flag was added to the `StakerInfo` struct to check if the same user tries to deposit more than once. The `deposit` function will revert if this is the case.

Issue #03	Pool reward logic is fundamentally flawed resulting in wrong and excessive rewards
Severity	🔴 HIGH SEVERITY
Description	updatePoolRewards and _computevWINRAmount are fundamentally flawed. They make various assumptions which do not hold: - rewardPerBlock does not change - maxBlock does not change - It is desired to do a subtracting logic as is currently implemented Specifically, users may suddenly receive surges of disproportionate rewards if the governance reconfigures rewardPerBlock for example. It is much cleaner to simply stop accruing rewards within the updatePoolRewards function.
Recommendation	Consider adding the cap logic to updatePoolRewards instead:

```
function updatePoolRewards() private {
  if (pool.tokensStaked == 0) {
      pool.lastRewardedBlock = arbSys.arbBlockNumber();
      firstBlock = arbSys.arbBlockNumber();
      return;
  }

  uint256 nextBlock = arbSys.arbBlockNumber();
  uint256 maxNextBlock = firstBlock + maxBlock;
  if (nextBlock > maxNextBlock) nextBlock = maxNextBlock;

  uint256 blocksSinceLastReward = nextBlock -
pool.lastRewardedBlock;
  uint256 rewards = blocksSinceLastReward * rewardPerBlock;
  pool.accumulatedRewardsPerShare += ((rewards *
ACC_REWARD_PRECISION) / pool.tokensStaked);
  pool.lastRewardedBlock = nextBlock;
}
```

and adjust _computevWINRAmount to:

```
function _computevWINRAmount(
  StakerInfo memory staker,
  uint256 blockNumber
) internal view returns (uint256 vWINRProfit) {
  vWINRProfit = (staker.amount *
pool.accumulatedRewardsPerShare) / ACC_REWARD_PRECISION;

  uint256 nextBlock = arbSys.arbBlockNumber();
  uint256 maxNextBlock = firstBlock + maxBlock;
  if (nextBlock > maxNextBlock) nextBlock = maxNextBlock;

  uint256 blocksSinceLastReward = nextBlock -
pool.lastRewardedBlock;

  if (blocksSinceLastReward > 0) {
      uint256 rewards = ((blocksSinceLastReward *
ACC_REWARD_PRECISION) * rewardPerBlock) /
      pool.tokensStaked;
      vWINRProfit += (staker.amount * rewards) /
ACC_REWARD_PRECISION;
  }

if (vWINRProfit < staker.rewardDebt) {
      return 0;
  } else {
      vWINRProfit -= staker.rewardDebt;
  }
}
```

This code section should be extremely carefully tested and vetted. The above is just example code meant to help get the team started and has not been tested.

Note that if the maxBlock is adjusted, rewards might still occur.

Resolution

RESOLVED

_getNextBlock() was implemented as recommended above. The recommended fix was implemented as well, at first with an error but this was later resolved.

Issue #04	It is possible that the `firstBlock` value resets after everyone withdraws, restarting reward emissions
Severity	● HIGH SEVERITY

Location	Lines 203-207
	`if (pool.tokensStaked == 0) {`
	`    pool.lastRewardedBlock = arbSys.arbBlockNumber();`
	`    firstBlock = arbSys.arbBlockNumber();`
	`    return;`
	`}`

Description	The `gWLP` contract stops emissions after `firstBlock + maxBlock` has been reached. The first block is essentially denoted as the block of the first deposit, as can be seen in the logic mentioned above. However, this logic is flawed since it resets the `firstBlock` whenever there are no tokens inside the pool. This has multiple flaws: - This logic can be called multiple times before any deposits come in through harvest. - More severely, once everyone withdraws, a subsequent deposit will reset the first block! As the first block can be reset once everyone withdraws, this might have severe implications on the reward logic as rewards would potentially re-enable. If there are still any reward tokens in the contract, these might be claimable by a malicious actor, even though this actor should not be able to claim them.
Recommendation	Consider whether it makes sense to add this logic to `deposit` instead of `harvest`. Consider whether it makes sense to instead check if the `firstBlock` is zero in the if-statement.
Resolution	✓ RESOLVED `firstBlock` is now only updated if the previous `firstBlock` value is 0.

Issue #05	**rewardPerBlock is incorrectly calculated in fundReward and updateMaxBlock**
Severity	● HIGH SEVERITY
Location	Line 129 `rewardPerBlock = _balance / _maxBlock;`
Description	`fundReward` and `updateMaxBlock` are called by the governance to set the `maxBlock` variable which determines over how many blocks the vWinr token rewards will be distributed. The `rewardPerBlock` is correspondingly updated by distributing the current contract balance of vWinr tokens among all users that have some stake. The problem is that this calculation assumes that all the previously accounted vWinr rewards have already been claimed by stakers and therefore excluded from `_balance`. This leads to an incorrect calculation of `rewardPerBlock`, which leads to the wrong amount of vWinr tokens being sent as reward to the users, and the function will start reverting at some point when the contract's balance is insufficient.
Recommendation	Consider tracking the received vWinr rewards using a state variable e.g. `allocatedRewards` which should be incremented in `updatePoolRewards` and decremented in `harvest`. The formula should therefore change as follows: `rewardPerBlock = (_balance - allocatedRewards) / (_firstBlock + _maxBlock - arbSys.arbBlockNumber());` Note that a requirement should be added to ensure that the `arbBlockNumber` is still before the last emission block. Secondly, the new reward rate is granted retroactively when the `rewardPerBlock` is updated. Consider adding a `updatePoolRewards` call before any `rewardPerBlock` adjustment to avoid retroactive updates. Consider re-using `updateMaxBlock` within `fundReward` to follow the DRY principle. Ideally an `internal` function is re-used.
Resolution	✓ RESOLVED The `updateMaxBlock` logic has been fully resolved in a secondary iteration. The reward rate logic now needs to be manually governed and retroactive checks are still not present.

Issue #06	`withdraw` **does not adhere to checks-effects-interactions which means if** `harvest` **were to allow for reentrancy, the WLP balance could be drained**
Severity	● LOW SEVERITY

Location	`Line 129`
	`harvest();`

Description	The `harvest` function is called early on within `withdraw`, however, the withdraw amount (the full user balance) is cached at that time. This means that if an exploiter were to be able to reenter into `withdraw` at this point, they would be able to potentially withdraw their balance twice. This issue is marked as low given that the interactions within `harvest` are unlikely to allow for reentrancy. Additionally, the body of `harvest` body does not adhere to checks-effects-interactions internally as well.
Recommendation	Consider adding reentrancy guards to `deposit`, `withdraw`, `claimWLP` and `harvest`. Note that by adding a guard to `harvest`, the contract will start reverting as deposit calls it. It is therefore necessary to create an internal `_harvest` function without a guard.
Resolution	✓ RESOLVED The recommendation was implemented.

Issue #07	Contract business logic can break significantly if deposits are re-enabled on the gWLP
Severity	● LOW SEVERITY
Description	The contract assumes that no further deposits can be made on `gWLP`. However, this is not strictly the case. It is possible for the `gWLP` governance to re-enable deposits by calling the following function: https://arbiscan.io/address/0x2798419F2Db8ea5F0f3A9b405313801e052B9cA7#code#F14#L56 If this function is called, the `GenesisWlpStaking` contract will start emitting excessive rewards to gWLP stakers and will likely fully break.
Recommendation	Consider locking out the possibility of calling this function, eg. by renouncing the role after the genesis USDC has been taken out.
Resolution	● ACKNOWLEDGED

Issue #08	**`withdraw` does not reset `wlpPerToken` which prevents a secondary deposit after withdrawal from accumulating wlp rewards**
Severity	● LOW SEVERITY
Description	The `withdraw` function does not revert the `wlpPerToken` variable. This is not critical but it does mean that a secondary deposit after a withdrawal will not accumulate any of the previously accumulated wlp tokens. However, this secondary deposit should have been eligible for these rewards. This last statement can be intuitively verified by having the user transfer their secondary deposit to a second wallet and stake via that second wallet.
Recommendation	Consider resetting `wlpPerToken` within `withdraw`. We are comfortable with the other reward debt not being reset since there does not appear to be any impact from that. Consider extremely carefully validating these changes, look for any potential impact and add test coverage since they have significant impact.
Resolution	✓ RESOLVED Secondary deposits are no longer possible so this case is also caught.

Issue #09	Arbitrum L2 block frequency might be quite irregular
Severity	● LOW SEVERITY
Description	The time-based rewards are granted based on the L2 block number. Eg. every block minted, some rewards unlock. However, since L2 blocks are so frequent, this might be quite variable with regards to a reward rate in seconds.
Recommendation	Consider moving to `timestamp` based block counts or `block.number` which is supposed to increment every 12 seconds since the PoS update.
Resolution	✓ RESOLVED `block.number` is now used.

Issue #10	Typographical errors
Severity	● INFORMATIONAL

Description	Line 13
	`event LogUpdatePool(uint256 lastRewardBlock, uint256 lpSupply, uint256 accRewardPerShare);`
	
	This event is unused and should be removed. If the client decides to keep it, we recommend renaming it and removing the word "log" since that is implicitly understood.
	
	Line 29
	`uint256 accumulatedRewardsPerShare;`
	
	This is accumulated rewards per share times `REWARDS_PRECISION`.
	
	Line 69
	`require(address(_gov) != address(0), "gov address zero");`
	
	_gov is already of the `address` type. The casting can therefore be removed.
	
	Line 80
	`* @notice funtion to set wlp and claimWlp addresses`
	
	This should say "function".
	
	Line 112
	`gWLP.safeTransferFrom(address(msg.sender), address(this), _amount);`
	
	Consider using `msg.sender` instead since it is already an address.
	
	Line 142
	`WLP.safeTransfer(address(msg.sender), WLPAmount);`
	
	Consider using msg.sender instead since it's already an address.

Line 210

```
pool.accumulatedRewardsPerShare += ((rewards *
ACC_REWARD_PRECISION) / pool.tokensStaked);
```

Consider removing the extra brackets.

Line 196

```
pool.wlpPerToken += (_amount * PRECISION) / totalGWlp;
```

This will revert claims if no one is staking (all supply has been withdrawn). Perhaps there might still be a small claim remaining and the team might still want to claim it. Since this small amount is likely going to be insignificant we just highlight the possibility here and nothing needs to be done if the team agrees with it being insignificant.

Both the `onlyGovernance` and `onlyRole(DEFAULT_ADMIN_ROLE)` modifiers are in use. Consider sticking to `onlyGovernance`.

`pendingRewards` and `claimWLP` can be made external.

`setAddresses`, `setWithdrawable`, `fundReward`, `updateMaxBlock` , `updateRatio`, `fundWLP` and `recoverToken` should have events.

Recommendation Consider fixing the typographical errors.

Resolution ✓ RESOLVED

All typographical issues except for the division by zero concern has been addressed.

Issue #11	Lack of validation
Severity	● INFORMATIONAL

Description	The contract contains functions with parameters which are not properly validated. Having unvalidated parameters could allow the governance or users to provide variable values which are unexpected and incorrect. This could cause side-effects or worse exploits in other parts of the codebase.
	
	Consider validating the following function parameters:
	
	Line 243
	`function updateRatio(uint256 _ratio) external onlyRole(DEFAULT_ADMIN_ROLE) {`
	
	Consider doing a rough check on the `_ratio` to validate that it is within an expected value range as to prevent any typos.
	
	Line 253
	`function fundWLP(uint256 _amount) external onlyRole(DEFAULT_ADMIN_ROLE) {`
	
	If this is called twice, the second call will wrongly update the ratio. Consider only allowing to call this function once.

Recommendation	Consider validating the function parameters mentioned above.
Resolution	● ACKNOWLEDGED

Issue #12	Gas optimizations
Severity	INFORMATIONAL

Description	vWINR, gWLP and arbSys can be made immutable. Note that within the constructor, line 71 needs to be adjusted to use _gWLP to avoid a compiler error.
	
	Line 161-162
	`staker.wlpRewardDebt += wlpAmount;`
	`pool.totalWlpProfit -= wlpAmount;`
	
	This can be moved within the if statement to save some gas in the zero reward scenario.
	
	Line 202
	`function updatePoolRewards() private`
	
	A lot of gas can be saved by caching arbSys.arbBlockNumber().

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	PARTIALLY RESOLVED The block number is now efficient as the client has moved to a native call instead of the sys call. The variables have also been made immutable. However, the gas optimization with regards to the if statement was not implemented.

PALADIN
BLOCKCHAIN SECURITY