PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For VersaGames

20 April 2022

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 full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for VersaGames on the Cronos 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	VersaGames
URL	https://versagames.io/
Platform	Cronos
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`VersaCROBar`	`0x8216E362d07741b562eBB02C61b1659B6B1258aD`	✓ MATCH
`SmartCraftInitializable`	`0x0c297000118aadD466da1D3d7800af7a8fB41A6b`	✓ MATCH
`SmartCraftInitializable (Dual Yield)`	`0x7AdeC517739FCb7451c43CABC207ABE1F5fFfAd6`	✓ MATCH
`VersaCROIGO`	`Not yet deployed`	PENDING
`VersaToken`	`0x00D7699b71290094CcB1a5884cD835bD65a78c17`	✓ MATCH
`Timelock`	`0x24734eac8901743f897702663e3d356d22306a7a`	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	1	-	1	-
🟠 Medium	2	1	-	1
🟡 Low	11	8	1	2
🟣 Informational	17	12	-	5
Total	**31**	**21**	**2**	**8**

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 VersaCROBar

ID	Severity	Summary	Status
01	MEDIUM	The first user can steal the tokens deposited by the next ones	✓ RESOLVED
02	LOW	xVERSA price can be manipulated	PARTIAL
03	INFO	Lack of `safeTransfer` usage within `enter` and `leave`	✓ RESOLVED
04	INFO	`versa` can be made `immutable`	✓ RESOLVED

1.3.2 SmartCraftInitializable

ID	Severity	Summary	Status
05	HIGH	Deposits do not support tokens with a fee on transfer	PARTIAL
06	MEDIUM	Pool uses the contract balance to figure out the total deposits	ACKNOWLEDGED
07	LOW	Contracts needs sufficient tokens	ACKNOWLEDGED
08	LOW	Denial of service: Governance `emergencyRewardWithdraw` takes out all reward tokens but does not stop reward emission	ACKNOWLEDGED
09	LOW	Contract malfunctions if the staking and reward tokens are the same	✓ RESOLVED
10	LOW	`stopReward` could be used to add rewards	✓ RESOLVED
11	INFO	Reward per block cannot be updated once rewards have started	✓ RESOLVED
12	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	✓ RESOLVED
13	INFO	`SMART_CRAFT_FACTORY` can be made `immutable`	✓ RESOLVED
14	INFO	Lack of validation	✓ RESOLVED
15	INFO	Typographical errors	✓ RESOLVED
16	INFO	`poolLimitPerUser` is vulnerable to Sybil attacks	ACKNOWLEDGED
17	INFO	Lack of events for `stopReward`	✓ RESOLVED

1.3.3 SmartCraftInitializable (Dual Yield)

ID	Severity	Summary	Status
18	INFO	`rewardToken` and `reward2Token` could be the same token	✓ RESOLVED

1.3.4 VersaCROIGO

ID	Severity	Summary	Status
19	LOW	`offeringToken` lacks validation	✓ RESOLVED
20	LOW	Deposits do not support tokens with a fee on transfer	✓ RESOLVED
21	LOW	Unnecessary precision for user allocations	✓ RESOLVED
22	LOW	`startBlock` and `endBlock` lack validation	✓ RESOLVED
23	LOW	Governance privileges: Admin can withdraw all `lpTokens` and `offeringToken` at any time	✓ RESOLVED
24	INFO	`limitPerUserInLP` is vulnerable to Sybil attacks	ACKNOWLEDGED
25	INFO	Typographical errors	✓ RESOLVED
26	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	✓ RESOLVED
27	INFO	`lpToken` and `offeringToken` can be made `immutable`	✓ RESOLVED

1.3.5 VersaToken

ID	Severity	Summary	Status
28	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	✓ RESOLVED
29	INFO	Governance functionality is broken	ACKNOWLEDGED
30	INFO	`delegateBySig` can be frontrun and cause denial of service	ACKNOWLEDGED
31	INFO	`mint` can be made external	ACKNOWLEDGED

1.3.6 Timelock

No issues found.

2 Findings

2.1 VersaCROBar

The VersaCROBar contract is a fork of SushiSwap's SushiBar. Users deposit VERSA tokens in this contract and receive xVERSA, the staked token of Versa. Upon creation, the VERSA:xVERSA ratio is 1:1, which means that each depositor receives an xVERSA amount equal to the VERSA amount they deposited. Every time VERSA tokens are sent directly to the contract (mainly by the protocol), the VERSA:xVERSA ratio increases. When users withdraw, they receive some bonus VERSA token proportional to the VERSA:xVERSA ratio.

By design, the ratio is always increasing or constant.

2.1.1 Issues & Recommendations

Issue #01	The first user can steal the tokens deposited by the next ones
Severity	● MEDIUM SEVERITY
Location	Line 754 `uint256 what = _amount.mul(totalShares).div(totalVersa);`
Description	When the `totalShares` is really low, especially upon creation, an user can mint an infinitesimal amount of xVERSA at a 1:1 ratio and send a big amount of VERSA to the contract to make the `totalVersa` amount really big. When other users try to enter and stake their tokens, they may receive 0 xVERSA, or in the best case, a rounded down number. The first users that were able to get some xVERSA will then be able to steal users' token deposited to the contract.
Recommendation	Consider permanently locking 1 VERSA and 1 xVERSA by minting it to this contract in the constructor, so `totalVersa` will always be at least equal to 1e18, ensuring no rounding down.
Resolution	✓ RESOLVED The client entered with 1 VERSA and locked it to ensure this exploit will not happen.

Issue #02	xVERSA price can be manipulated
Severity	● LOW SEVERITY

Location	Line 745
	`uint256 totalVersa = versa.balanceOf(address(this));`
	
	Line 765
	`uint256 what =`
	`_share.mul(versa.balanceOf(address(this))).div(totalShares);`

Description	xVERSA price can be manipulated by sending tokens manually. The xVERSA price can also be manipulated to flash enter/leave calls within a single transaction. This is not a problem to the protocol itself but might be something to consider in derivative protocols.
Recommendation	Consider using a local variable that will store the amount of VERSA sent to this contract. Make sure to add a `deposit` function to be able to add Versa reward tokens to that contract to increase the VERSA:xVERSA ratio, while allowing only a set of addresses to be able to use that function.
Resolution	● PARTIALLY RESOLVED The client has indicated that this is not a problem for their smart contract protocol as they do not use this ratio for any critical functionality outside of the `VersaCROBar`.

Issue #03	Lack of `safeTransfer` usage within `enter` and `leave`
Severity	● INFORMATIONAL

Location	Line 758
	`versa.transferFrom(msg.sender, address(this), _amount);`
	
	Line 769
	`versa.transfer(msg.sender, what);`

Description	In the `enter` and `leave` functions the transfer method is used to transfer tokens. This will not work for tokens that returns `false` on transfer (or malformed tokens that do not have a return value). This is not an issue for VERSA tokens, but if the contract is forked, it may become an issue for these forks.
Recommendation	Consider using `safeTransfer` instead of `transfer` as is done throughout most of this codebase.
Resolution	✓ RESOLVED

Issue #04	`versa` can be made `immutable`
Severity	● INFORMATIONAL
Location	Line 734 `IERC20 public versa;`
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	✓ RESOLVED

2.2 SmartCraftInitializable

The SmartCraftInitializable contract allows users to deposit `stakedToken` and receive `rewardToken`. The contract looks a lot like a Masterchef but with a single pool. The `rewardToken` is sent by the Admins, and not minted by it, so the rewards balance needs to be constantly monitored to make sure that everyone can claim their shares.

The owner can add a maximum of `stakedToken` deposited per user, and once this limit is reached, users will not be able to deposit anymore.

2.2.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `initialize`
- `emergencyRewardWithdraw`
- `recoverWrongTokens`
- `stopReward`
- `updatePoolLimitPerUser`
- `updateRewardPerBlock`
- `updateStartAndEndBlocks`
- `updateRewardPerBlockAfterStart`
- `renounceOwnership`
- `transferOwnership`

2.2.2 Issues & Recommendations

Issue #05	Deposits do not support tokens with a fee on transfer
Severity	● HIGH SEVERITY
Description	Within the `deposit` function, there is no logic that supports tokens with a fee on transfer. Therefore if such tokens are deposited, the contract will receive less tokens than the user will get credited. This could be exploited where a malicious user can drain the whole pool, which results in absurd reward minting.
Recommendation	Consider adding before-after logic for fee-on-transfer tokens for the `deposit` function.
Resolution	● PARTIALLY RESOLVED The client has indicated that they will never support such tokens. This issue is marked as partially resolved as users might still be severely impacted if such a token is ever deposited.

Issue #06	Pool uses the contract balance to figure out the total deposits
Severity	● MEDIUM SEVERITY
Description	As with pretty much all Masterchefs and staking contracts, the total number of tokens in the contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the masterchef. This issue is rated as Medium because `stakedToken` can be the same token as the `rewardToken` and cause even more dilution. This is again amplified because the contract does not mint its token, they need to be transferred to the contract beforehand.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits.
Resolution	● ACKNOWLEDGED The client has indicated that they however will not support fee-on-transfer tokens which should remove most if not all of the user impact as long as this is respected.

Issue #07	Contracts needs sufficient tokens
Severity	● LOW SEVERITY
Description	As the `rewardTokens` are sent to the contract by the admins and not minted by the contract, the transfer of reward tokens to users might revert if the balance in the contract is too low (i.e., there are not have enough tokens to reward users).
Recommendation	Consider making sure that the contract always has enough tokens. The easiest way would be to send the entire amount needed directly to the contract (as the `rewardPerBlock` cannot be changed once the pool has started. This amount would be equal to `rewardPerBlock * (bonusEndBlock - startBlock)`.
Resolution	● ACKNOWLEDGED The team has indicated that they will make sure to fully fund these contracts during deployment.

Issue #08	Denial of service: Governance `emergencyRewardWithdraw` takes out all reward tokens but does not stop reward emission
Severity	● LOW SEVERITY
Description	The `emergencyRewardWithdraw` does not set the `bonusEndBlock` to the current `block.number`. Because of this, all withdrawals may revert because the contract may no longer have enough tokens to transfer it to the users.
Recommendation	Consider setting the `bonusEndBlock` to `block.number` to prevent this issue. It should be noted that users might and likely will still have pending harvests which would still cause functions to fail. A `safeTokenTransfer` function that transfers up to the contract's balance might be ideal. Such functionality is present in most masterchefs.
Resolution	● ACKNOWLEDGED The team has indicated they do not plan to ever call this function.

Issue #09	Contract malfunctions if the staking and reward tokens are the same
Severity	● LOW SEVERITY

Description	If the two tokens are the same, `updatePool` will be incorrect because the `rewardToken` would be incorporated in `stakedTokenSupply`, causing rewards to be diluted.
	
	Line 1143
	`uint256 stakedTokenSupply = stakedToken.balanceOf(address(this));`
	
	Additionally, the `emergencyRewardWithdraw` function could withdraw a user's deposits.
	
	L1240
	`rewardToken.safeTransfer(address(msg.sender), _amount);`

Recommendation	Consider adding a requirement that the two tokens are different.
Resolution	✓ RESOLVED The recommendation has been implemented as a requirement in the constructor of the contract. This enforces that both tokens must not be equal to each other.

Issue #10	`stopReward` could be used to add rewards
Severity	● LOW SEVERITY

Location	Lines 1263 - 1265
	`function stopReward() external onlyOwner {`
	`    bonusEndBlock = block.number;`
	`}`

Description	The function `stopReward` could be used to add rewards when the rewards are over because it sets the `bonusEndBlock` to the current block, but the rewards may already be finished. This will allow rewards to be re-enabled and be distributed from the previous `bonusEndBlock` to the current block number.
Recommendation	Consider checking that `block.number < bonusEndBlock` to prevent additional rewards from being distributed.
Resolution	✓ RESOLVED The recommended check has been added to the `stopReward` function.

Issue #11	Reward per block cannot be updated once rewards have started
Severity	INFORMATIONAL
Description	The `rewardPerBlock` variable cannot be updated once the rewards have started. We have raised this issue to confirm that this is intended.
Recommendation	Consider adding a function to update the rewards rate if needed while adding a cap to prevent setting it to a huge value. If the `rewardPerBlock` changes, consider updating the pool to reward users accurately. This issue can also be resolved on the note that VersaGames does not need to adjust the emission rate after rewards start.
Resolution	✓ RESOLVED The client has added a `updateRewardPerBlockAfterStart` privileged function. They have added a maximum value to cap the maximum reward rate. This value is set in the constructor and is immutable, therefore users should check that this value was set accordingly and not to an absurdly high number.

Issue #12	`msg.sender` is unnecessarily cast to `address(msg.sender)`
Severity	INFORMATIONAL
Description	`msg.sender` is cast to `address(msg.sender)` throughout the contract when used with `pool.lpToken.safeTransfer()`. This is unnecessary.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`. An even better solution to be consistent would be to replace `address(msg.sender)` by `_msgSender()` as this contract inherits from Ownable that inherits from Context.
Resolution	✓ RESOLVED

Issue #13	SMART_CRAFT_FACTORY can be made immutable
Severity	● INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	✓ RESOLVED

Issue #14	Lack of validation
Severity	● INFORMATIONAL

Description	The contract contains sections of code which lack proper validation. This could cause errors in case unexpected inputs are provided.
	
	Line 1142
	`rewardPerBlock = _rewardPerBlock;`
	
	`rewardPerBlock` could be a huge value. Consider adding an upper bound for `rewardPerBlock`.
	
	Lines 1143-1144
	`startBlock = _startBlock;`
	`bonusEndBlock = _bonusEndBlock;`
	
	Consider checking that `startBlock` is less than `bonusEndBlock`.

Recommendation	Consider implementing the above recommendations.
Resolution	✓ RESOLVED The client has introduced validation on the end block and a maximum cap to the `rewardPerBlock`.

Issue #15	Typographical errors
Severity	● INFORMATIONAL

Description	Line 1069
	`// The block number when stakedToken mining ends.`
	
	The comment should mention `rewardToken` instead of `stakedToken`.
	
	Line 1070
	`uint256 public bonusEndBlock;`
	
	The variable name should be `rewardsEndBlock` as this is not the end of any bonus.
	
	Line 1154
	`PRECISION_FACTOR = uint256(10**(uint256(36).sub(decimalsRewardToken)));`
	
	The final casting to `uint256` is unnecessary.
	
	Line 1165
	`@param _amount: amount to withdraw (in rewardToken)`
	
	The comment should mention `deposit (in stakedToken)` instead of `withdraw (in rewardToken)`.
	
	Line 1194
	`@param _amount: amount to withdraw (in rewardToken)`
	
	The comment should mention `(in stakedToken)` instead of `(in rewardToken)`.
	
	Line 1219
	`@notice Withdraw staked tokens without caring about rewards rewards`
	
	The comment should mention rewards only once.

	Line 1250~
	`function recoverWrongTokens(address _tokenAddress, uint256 _tokenAmount) external onlyOwner {`
	`    require(_tokenAddress != address(stakedToken), "Cannot be staked token");`
	`    require(_tokenAddress != address(rewardToken), "Cannot be reward token");`
	
	`    ERC20(_tokenAddress).safeTransfer(address(msg.sender), _tokenAmount);`
	
	`    emit AdminTokenRecovery(_tokenAddress, _tokenAmount);`
	`}`
	
	The `_tokenAddress` could be cast to ERC20 directly and avoid the unnecessary cast to `address` and `ERC20`. It should also be noted that Paladin in general prefers casting parameters to the interface IERC20 compared to ERC20 as one does not care about the implementation of this standard.

Recommendation	Consider fixing the typographical errors.
Resolution	✓ RESOLVED Note that the client however did not replace ERC20 with IERC20.

Issue #16	**`poolLimitPerUser` is vulnerable to Sybil attacks**
Severity	● INFORMATIONAL
Description	The `poolLimitPerUser` value indicates the maximum amount of token deposited per user. There is however nothing that prevents a user from creating several wallets to deposit more than allowed.
Recommendation	As Sybil resistance is an extremely difficult topic to solve, we have no easy recommendation. We have seen well-known actors utilise KYC procedures to do this but expect this to not match with the ethos of Versa.
Resolution	● ACKNOWLEDGED

Issue #17	Lack of events for `stopReward`
Severity	● INFORMATIONAL
Description	Function that affect the status of sensitive variables should emit events as notifications. Additionally, an actual event `RewardStopped` was created but is unused — the client might have forgotten to add it within the `stopReward` function.
Recommendation	Add an event for the function.
Resolution	✓ RESOLVED A `RewardsStop` event has been added.

2.3 SmartCraftInitializable (Dual Yield)

The SmartCraftInitializable (Dual Yield) is an exact copy of the Single Yield SmartCraftInitializable except that it adds a second reward token. Users can deposit their stakedToken to receive rewards in rewardToken and reward2Token.

All the errors previously raised for the single yield version also apply for this contract.

2.3.1 Privileged Functions

The following functions can be called by the `feeToSetter`:

- `initialize`
- `emergencyRewardWithdraw`
- `emergencyReward2Withdraw`
- `recoverWrongTokens`
- `stopReward`
- `updatePoolLimitPerUser`
- `updateRewardPerBlock`
- `updateReward2PerBlock`
- `updateStartAndEndBlocks`
- `renounceOwnership`
- `transferOwnership`

2.3.2 Issues & Recommendations

Issue #18	`rewardToken` and `reward2Token` could be the same token
Severity	INFORMATIONAL
Description	Although not necessarily an issue, the two reward tokens being equal would be gas inefficient. The single yield version would do the exact same thing if the `rewardPerBlock` was set accordingly without using as much gas.
Recommendation	Consider requiring that the two tokens are different.
Resolution	✓ RESOLVED Validation has been added that these two tokens must be different.

2.4 VersaCROIGO

The VersaCROIGO allows users to deposit `lpToken` to receive `offeringToken` in proportion to the share of all the `lpToken` deposited in this contract. It is a common method of raising funds for new token launches.

Each pool can have fees that are proportional to the overflow of deposited tokens compared to the raising amount. This means that a small portion of the overflow is not refunded but instead paid to the platform in the form of fees.

2.4.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `finalWithdraw`
- `recoverWrongTokens`
- `setPool`
- `updateCampaignId`
- `updateStartAndEndBlocks`
- `renounceOwnership`
- `transferOwnership`

2.4.2 Issues & Recommendations

Issue #19	`offeringToken` lacks validation
Severity	● LOW SEVERITY
Location	L985 `offeringToken.safeTransfer(address(msg.sender), offeringTokenAmount);`
Description	There is no guarantee for users that the admin has sent offering tokens into the contract. `harvest()` will revert if this is the case, and users will not be able to call it.
Recommendation	Consider sending the tokens within `setPool`. Be careful if it is a token with a fee on transfer as the amount transferred and the amount received may be different.
Resolution	✓ RESOLVED The client has indicated they have no desire to support such tokens. We have reiterated the need for them to be careful and check all that tokens added have no fees on transfer.

Issue #20	Deposits do not support tokens with a fee on transfer
Severity	● LOW SEVERITY
Description	Within the `deposit` function, there is no logic that supports tokens with a fee on transfer. Therefore, during a deposit, the Masterchef will receive fewer tokens than the user will get credited for. This could be exploited where a malicious user can drain the whole pool, which results in absurd reward minting.
Recommendation	Consider adding logic to handle tokens with a fee on transfer: `uint256 balanceBefore = pool.lpToken.balanceOf(address(this)); pool.lpToken.safeTransferFrom(msg.sender, address(this), _amount); _amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
Resolution	✓ RESOLVED

Issue #21	Unnecessary precision for user allocations
Severity	● LOW SEVERITY
Location	Line 1256 `uint256 allocation = _getUserAllocationPool(_user, _pid);`
Description	Rounding down user allocations with 1e12 precision is unnecessary and could lead to severe rounding errors for smaller stakers in a large IGO.
Recommendation	Consider inlining the allocation math to avoid this division before multiplication antipattern.
Resolution	✓ RESOLVED

Issue #22	`startBlock` and `endBlock` lack validation
Severity	● LOW SEVERITY

Location	Lines 898~
	`startBlock = _startBlock;`
	`endBlock = _endBlock;`

Description	The contract contains sections of code which lack proper validation. This could cause errors in case unexpected inputs are provided.
Recommendation	Consider checking that `_startBlock < _endBlock` and that `_startBlock > block.number`.
Resolution	✓ RESOLVED

Issue #23	Governance privileges: Admin can withdraw all `lpTokens` and `offeringToken` at any time
Severity	● LOW SEVERITY
Location	Line 1001 `function finalWithdraw(uint256 _lpAmount, uint256 _offerAmount) external override onlyOwner {`
Description	The admin can withdraw all deposited `lpTokens` and all `offeringToken` at any time before the end block and without giving users enough time to harvest.
Recommendation	Consider adding a requirement that the IGO is over and add a few days' delay for users to harvest before being able to call `finalWithdraw`.
Resolution	✓ RESOLVED The `finalWithdraw` can now only occur 100,800 blocks after the IGO has ended.

Issue #24	`limitPerUserInLP` is vulnerable to Sybil attacks
Severity	● INFORMATIONAL
Description	The `limitPerUserInLP` value indicates the maximum amount of LP deposited per user. There is however nothing stopping a user from creating many wallets to deposit more than allowed.
Recommendation	As Sybil resistance is an extremely difficult topic to solve, we have no easy recommendation. We have seen well-known actors utilize KYC procedures to do this but expect this to not match with the ethos of Versa.
Resolution	● ACKNOWLEDGED

Issue #25	Typographical errors
Severity	● INFORMATIONAL

Description	The contract contains a number of typographic mistakes which we've enumerated below in a single issue in an effort to keep the report size reasonable.
	
	Line 764
	`* @dev 100,000 means 0.1(10%)/ 1 means 0.000001(0.0001%)/ 1,000,000 means 1(100%)`
	
	Lines 1124/1211/1294
	`* @dev 100,000,000,000 means 0.1 (10%) / 1 means 0.0000000000001 (0.0000001%) / 1,000,000,000,000 means 1 (100%)`
	
	The comments are wrong, they should be:
	
	`* @dev 100,000,000,000 means 0.1 (10%) / 1 means 0.000000000001 (0.0000000001%) / 1,000,000,000,000 means 1 (100%)`
	
	Line 877
	`* @notice It initializes the contract (for proxy patterns)`
	
	This contract is not a proxy contract.
	
	Line 1022
	`function recoverWrongTokens(address _tokenAddress, uint256 _tokenAmount) external onlyOwner {`
	
	The `_tokenAddress` could be casted directly to IERC20.
	
	Lines 788 / 1179
	`* @notice External view function to see user offering and refunding amounts for both pools`
	
	The comment should mention that it also returns the tax amount

	Line 1301 `return _userInfo[_user][_pid].amountPool.mul(1e18).div(_poolInformation[_pid].totalAmountPool.mul(1e6));` Multiplying by 1e18 and dividing by 1e6 in the same line is equivalent to multiplying by 1e12. Doing this in two steps has no benefit with regards to precision.
Recommendation	Consider fixing the typographical errors.
Resolution	✓ RESOLVED

Issue #26	**`msg.sender` is unnecessarily cast to `address(msg.sender)`**
Severity	● INFORMATIONAL
Description	`msg.sender` is cast to `address(msg.sender)` throughout the contract. This is unnecessary.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	✓ RESOLVED

Issue #27	**`lpToken` and `offeringToken` can be made `immutable`**
Severity	● INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variables explicitly immutable.
Resolution	✓ RESOLVED

2.5 VersaToken

VersaToken is a simple ERC-20 token which will be used as the main reward tokens for the different staking contracts.

This contract allows for the token to be minted when the mint function is called by the owner of the token contract, which at the time of deployment would be the VersaGames team.

As the different contracts need to receive VERSA to distribute them, the team will maintain the privileges to mint tokens. Users should therefore carefully check that the team does not mint a large amount of tokens to themselves and not use it as rewards for the different contracts.

2.5.1 Token Overview

Address	`0x00D7699b71290094CcB1a5884cD835bD65a78c17`
Name	VersaGames
Symbol	VERSA
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Max Wallet Size	None
Transfer Fees	None
Pre-mints	320,000,000

2.5.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `mint`
- `transferOwnership`
- `renounceOwnership`

2.5.2 Issues & Recommendations

Issue #28	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	● LOW SEVERITY
Description	The `mint` function allows the owner (contract deployer) to mint tokens before ownership is transferred to the Masterchef. This could be used to mint a large amount of tokens and potentially dump them on user generated liquidity when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be used by letting your community know how much was minted, where the tokens are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	✓ RESOLVED At the time of writing, the VersaToken ownership has been transferred to a TimeLock with a minimum delay of 7 days (0x24734eac8901743F897702663E3d356D22306A7a). Even though minting is still possible, users could be alerted to this 7 days in advance which Paladin deems more than reasonable.

Issue #29	Governance functionality is broken
Severity	INFORMATIONAL
Description	Although there is YAM-related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism. It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap but it does render this whole mechanism useless. Because of this, projects like SushiSwap and PancakeSwap all use snapshot.org nowadays.
Recommendation	The broken delegation-related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, used by many of the larger projects.
Resolution	ACKNOWLEDGED The client has indicated that they have already deployed the token and can therefore no longer remove this code. Given the informational nature of this issue, this does not pose any user risk.

Issue #30	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if `delegateBySig` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.
Recommendation	Similar to the broken governance functionality issue, the delegate logic can just be removed.
Resolution	ACKNOWLEDGED The client has indicated that they have already deployed the token and can therefore no longer remove this code. Given the informational nature of this issue, this does not pose any user risk.

Issue #31	`mint` can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the function as external.
Resolution	ACKNOWLEDGED The client has indicated that they have already deployed the token and can therefore no longer remove this code. Given the informational nature of this issue, this does not pose any user risk.

2.6 Timelock

The Timelock contract is a clean fork of Compound Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

Parameter	Value	Description
Delay	7 days	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	7 days	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Maximum Delay	30 days	The maximum delay indicates the highest value that the `delay` can be set.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.6.1 Issues & Recommendations

No issues found.

PALADIN
BLOCKCHAIN SECURITY