CERTIK

Security Assessment

Virtuswap - 2nd audit

CertiK Assessed on May 27th, 2023

CertiK Assessed on May 27th, 2023

Virtuswap - 2nd audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Exchange	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 05/27/2023	N/A

CODEBASE

https://github.com/Virtuswap/v1-core

...View All

COMMITS

base: f7f7f170514f9ce111268bad1a26f424935adf4f

update1: cc560f3ef20e240e6ac77737f0b50a02dc6d0ba1

update2: de375fba7b852b256defd3067dafe4a1c02c5a3a

...View All

Vulnerability Summary

23	20	0	2	1	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
8	Minor	6 Resolved, 2 Partially Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
12	Informational	12 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | VIRTUSWAP - 2ND AUDIT

CODEBASE | VIRTUSWAP - 2ND AUDIT

Repository

https://github.com/Virtuswap/v1-core

Commit

base: f7f7f170514f9ce111268bad1a26f424935adf4f
update1: cc560f3ef20e240e6ac77737f0b50a02dc6d0ba1
update2: de375fba7b852b256defd3067dafe4a1c02c5a3a
update3: f1306da14917a10ddd6c31de0d0a5c0f854e1dfb
update4: 1e9bfe82f1ff07b1e3e9aa82ddbe52aa83d978db
update5: 57447c4b96279358b43575d41656baf3e0504d44

AUDIT SCOPE | VIRTUSWAP - 2ND AUDIT

18 files audited ● 4 files with Acknowledged findings ● 4 files with Resolved findings ● 10 files without findings

ID	File	SHA256 Checksum
ERV	contracts/vExchangeReserves.sol	b7c7e0dc7030cb35743623bf0fdcf83bf078e88147afcb964f4b24ccf09ee37f
PVB	contracts/vPair.sol	a1ed65f5457306e5db946dc611ff16a4201c0e0317f37df70d92ee94c5176fad
PFV	contracts/vPairFactory.sol	048f7516240b1aa49565f7a365498e5f7a08204c10dfb23dd6f8b942d8490ef5
RVB	contracts/vRouter.sol	febc61a47e3118bcc5342f3befa7a0545eb2302f5cb6e8322fc4f7007646b5e7
SLV	contracts/libraries/vSwapLibrary.sol	1b403ff28c5b8baa3c91763e4993fc22b3e175a84e3d359b6bbe80cf95f9f6cd
TYP	contracts/types.sol	edfd5968505ea61e7f238b20c9c3fa14e655390abefb745c75e1eb5d5a5e1b52
PMV	contracts/vPoolManager.sol	2caae9f17694eac3ebd6c874af621cc12df770ee4b133ddfb6b6f347fec5f485
SER	contracts/vSwapERC20.sol	d11d46825148fb26289db7cbcc00ae5fa415dab5b66c26c16557a825b9312d1f
IER	contracts/interfaces/IvExchangeReserves.sol	f4373f438042e0a5af65ea2d8db9175908170f15522e8ad96c6534541378a699
IFS	contracts/interfaces/IvFlashSwapCallback.sol	10cda6df5d8199a1c3386df2f5bed335a334c1a28ec4619b126ad26d69b255ae
IPV	contracts/interfaces/IvPair.sol	8b72dc29e95a0a906565de3e127b3089bbb61bfed9f9e57494f2bf6b5961be6d
IPF	contracts/interfaces/IvPairFactory.sol	398c8ccaa7258afe0f1a5b09c7645b926631edb0820d8119b8b522e97b9da045
IPM	contracts/interfaces/IvPoolManager.sol	c94a9afe51b0c4855b97065ad65816b25f7ee1ecea9ac4d272e336d7840b6148
IRV	contracts/interfaces/IvRouter.sol	7e9676b39f7e05aa5e0757aa2cc667eb63c35ccd7bfc3da3002a56a263bcb882

ID	File	SHA256 Checksum
ISP	contracts/interfaces/IvSwapPoolDeployer.sol	7c105ffa4cca51d5b6feebd6337327d0029870 78cda3236a75f811002144bd6d
WET	contracts/interfaces/external/WETH9.sol	2981934b8a59403f4a3679437783f8f4aa2b95 4ed570a75786c34ac981bd55aa
IWE	contracts/interfaces/external/IWETH9.sol	0e17a4f53e14b2aa10e394afc92a7b0d3ac1fd c78610ae6ce720ac19d7e2f05e
PAV	contracts/libraries/PoolAddress.sol	dcc8034b37dc712faef920baabc859fa944839 b4f3350d1640210c00aeaf3213

APPROACH & METHODS | VIRTUSWAP - 2ND AUDIT

This report has been prepared for Virtuswap to discover issues and vulnerabilities in the source code of the Virtuswap - 2nd audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

DEPENDENCIES | VIRTUSWAP - 2ND AUDIT

Third Party Dependencies

The protocol is serving as the underlying entity to interact with third party protocols. The third parties that the contracts interact with are:

- `WETH9`

The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

Recommendation

We recommend constantly monitoring the third parties involved to mitigate any side effects that may occur when unexpected changes are introduced.

Out-of-Scope Dependencies

Newton's method for approximating roots via the derivative is iterated seven times in function `getMaxVirtualTradeAmountRtoN()` . The margin of error may reduce with the number of steps taken. The assessment of the margin of error through the use of this fixed number of iterations to approximate the derivative is not within the scope of this audit. We recommend the team ensures the sufficient accuracy of the calculation for all possible ranges of values used in the calculation.

FINDINGS | VIRTUSWAP - 2ND AUDIT

23	0	1	2	8	12
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Virtuswap - 2nd audit. Through this audit, we have uncovered 23 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-12	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
CON-07	`vFlashSwapCallback()` In VExchangeReserves` Can Be Called Directly	Logical Issue, Control Flow	Medium	Resolved
PVB-02	Discrepancy In Checks Involving Pair Balances And `vPool` Balances	Inconsistency	Medium	Resolved
CON-02	Missing Input Validation	Volatile Code	Minor	Resolved
CON-04	`allowList` Token Updates Should Be Carefully Considered	Volatile Code	Minor	Resolved
CON-05	Swapping Functions Require A Strict Inequality Check For `amountOut`	Logical Issue	Minor	Resolved
PVB-03	Function Checks Recorded `_balanceOut` After Transfer	Logical Issue	Minor	Resolved
RVB-02	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
RVB-03	Potential Loss Of Ether	Control Flow	Minor	Resolved
SLV-03	Potential Division By Zero	Volatile Code	Minor	Partially Resolved

ID	Title	Category	Severity	Status
SLV-04	Unsafe Integer Cast And Potential Underflow/Overflow	Logical Issue	Minor	Partially Resolved
CON-06	Discussion On `_update()` Design	Inconsistency	Informational	Resolved
CON-08	Shadowing Local Variable	Coding Style	Informational	Resolved
CON-09	Missing Emit Events	Coding Style	Informational	Resolved
ERV-01	Checks Do Not Guarantee Consistent `factory` Address Is Used Across Pairs	Inconsistency	Informational	Resolved
PFV-02	User-Defined Getters	Coding Style	Informational	Resolved
PVB-04	Missing Check Against `reserveRatioWarningThreshold` For Non-Privileged Users	Control Flow	Informational	Resolved
PVB-05	Discussion On Admin Versus EmergencyAdmin In `vPair`	Control Flow	Informational	Resolved
PVB-06	Missing Error Messages	Coding Style	Informational	Resolved
PVB-07	Typo	Coding Style	Informational	Resolved
SLV-01	Documentation Needed On Function `getMaxVirtualTradeAmountRtoN()`	Coding Style	Informational	Resolved
SLV-02	Unused Function	Coding Style	Informational	Resolved
TYP-01	Variable Names Too Similar	Coding Style	Informational	Resolved

CON-12 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/vExchangeReserves.sol (updated base): 22~23; contracts/vPair.sol (updated base): 51, 542~543, 560~561, 571~572, 580~581, 587, 593; contracts/vPairFactory.sol (updated base): 90, 102, 112, 119, 129, 136, 146; contracts/vRouter.sol (updated base): 496**	**Acknowledged**

Description

`vExchangeReserves.sol`

In the contract `vExchangeReserves` the role `admin` (from `factory`) has authority over the following functions:

- `changeIncentivesLimitPct()`;

Any compromise to the `admin` account may allow the hacker to take advantage of this authority and set `newLimit` to 0, making `_leftoverAmount` 0 in `swapNativeToReserve()`.

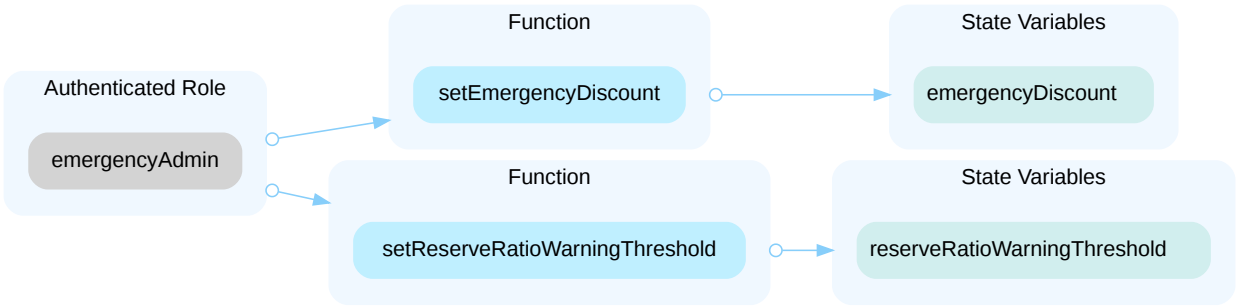
`vPair.sol`

In the contract `vPair` the role `admin` (from `factory`) has authority over the following functions:

- `setAllowList()`;
- `setFee()`;
- `setMaxReserveThreshold()`;
- `setMaxAllowListCount()`;

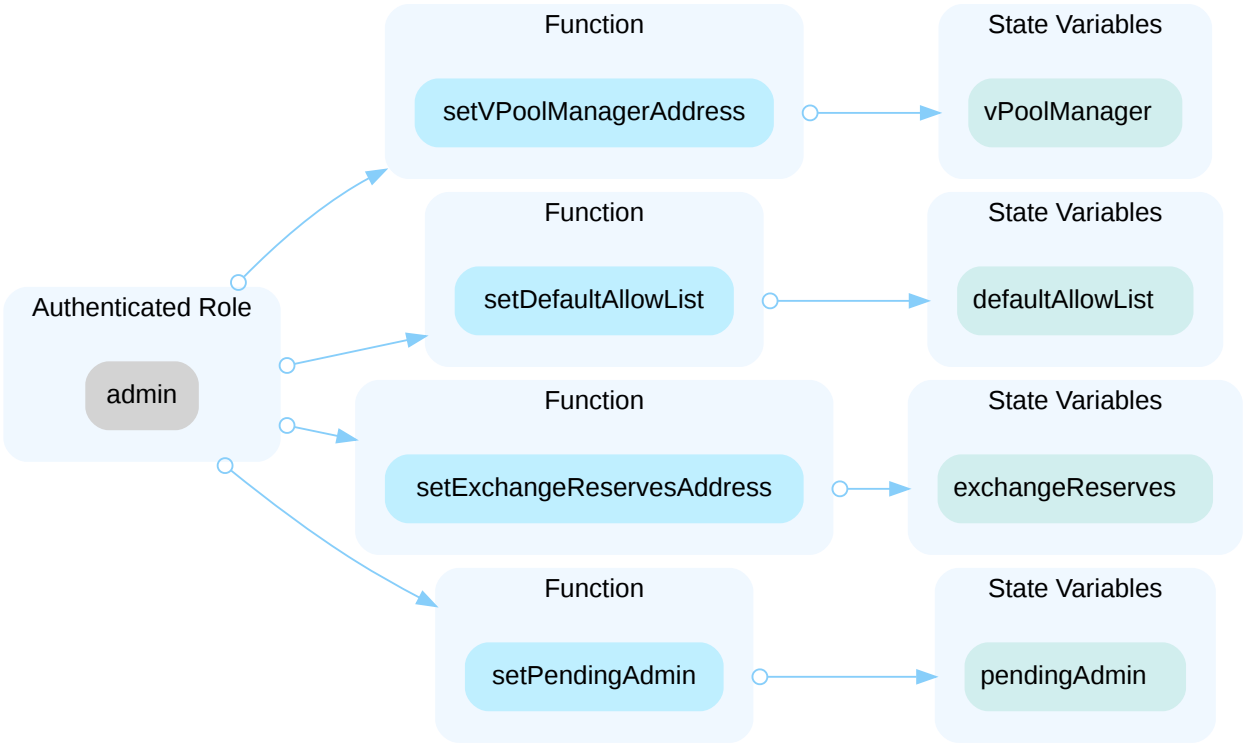
Any compromise to the `admin` account may allow the hacker to take advantage of this authority and extract most funds from the `vPair` contract by calling function `swapNativeToReserve()` to swap out a given specified token on the `allowList` without paying back a token amount.

In the contract `vPair` the role `emergencyAdmin` has authority over the functions shown in the diagram below. Any compromise to the `emergencyAdmin` account may allow the hacker to take advantage of this authority and set the `emergencyDiscount` at 100% so that either the `emergencyAdmin` or `admin` role can use function `swapNativeToReserve()` to extract most funds from the `vPair` contract.

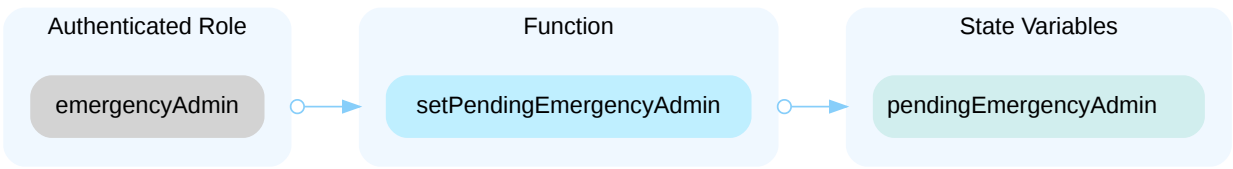


`vPairFactory.sol`

In the contract `vPairFactory` the role `admin` has authority over the functions shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and set a malicious contract for `vPoolManagerAddress`, or update `defaultAllowList` in such a way to steal all funds from each `vPair` via a token address owned by the hacker.

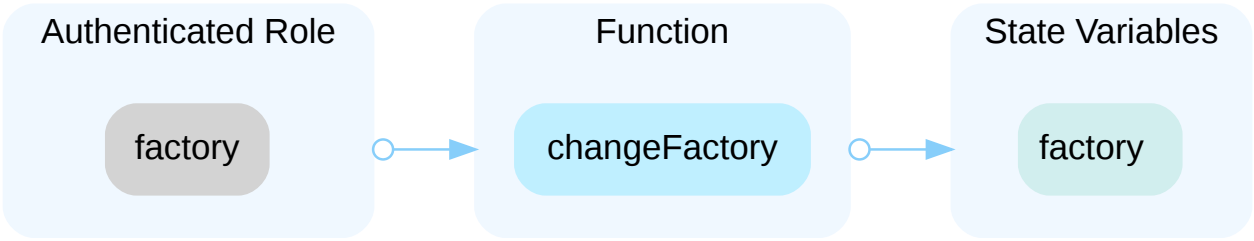


In the contract `vPairFactory` the role `emergencyAdmin` has authority over the functions shown in the diagram below. Any compromise to the `emergencyAdmin` account may allow the hacker to take advantage of this authority and disrupt functionality within any `vPair` contract (see `vPair` section above).



`vRouter.sol`

In the contract `vRouter` the role `factory` has authority over the functions shown in the diagram below. Any compromise to the `factory` account may allow the hacker to take advantage of this authority and change the `factory` address to an unknown implementation which can affect the expected pair addresses returned from function `getPairAddress()` .



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND

- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

`[CertiK]` : The team acknowledges the finding and states they will address the issue in the future, which will not be included in this audit engagement.

CON-07 `vFlashSwapCallback()` IN VEXCHANGERESERVES` CAN BE CALLED DIRECTLY

Category	Severity	Location	Status
Logical Issue, Control Flow	Medium	contracts/vExchangeReserves.sol (updated base): 39~40, 84~88, 93~94; contracts/vPair.sol (updated base): 202~203, 266~278	Resolved

Description

It is inferred that function `vFlashSwapCallback()` in contract `vExchangeReserves` is only meant to be called by a `vPair` contract after the call is initiated in function `exchange()`. Instead, the function can be called directly through a custom contract by encoding the `msg.sender` contract address as the `jkPair1` address.

This address is only used as the `to` parameter in the call to `swapNativeToReserve()` in decoded address `jkPair2`. As such, the call can be used to directly collect the incentive amount calculated from the difference in the actual current balance of the swapped native token and its previously recorded pair balance.

Similarly, function `exchange()` can be called with a custom `jkPair2` contract address, so that function call to `swapNativeToReserve()` includes this contract as the `to` address. As long as the custom `jkPair2` contract logic returns the `requiredAmountIn` needed for the swap, the transaction can successfully be completed.

Recommendation

The custom functionality appears to be in place in order to control the way in which a `vPair` contract's function `swapNativeToReserve()` is referenced. Please specify if it is intended design to allow the `vFlashSwapCallback()` function to be called directly, and whether it is intentional to allow interaction with `exchange()` in the manner described above.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit acf31ecefc41c8643c49fd0afa0b4c9d9a1eb9a6 and d78d6ffa2c93c1177883cbe5bb33bd8fe4931b39.

PVB-02 DISCREPANCY IN CHECKS INVOLVING PAIR BALANCES AND `vPool` BALANCES

Category	Severity	Location	Status
Inconsistency	Medium	contracts/vPair.sol (updated base): 231~232, 235~236, 291~310, 313~314, 353~354, 370~371, 385~403	Resolved

Description

Function `swapNativeToReserve()` and `swapReserveToNative()` check that `amountOut` does not exceed `vPool.balance1`. This balance in `vPool` is meant to represent the amount of funds that the virtual pool holds for that token, however, it may not be an up-to-date representation of the actual balance that the `vPair` token contract holds. This is because the update to the returned `vPool` values depends on whether the current `block.number` compared to the last updated block, and the last pair balances used in calculating `vPool` information may not have been recently updated.

In the case of `swapNativeToReserve()`, if the `amountOut` exceeds the last recorded `reserves` mapping value for `vPool.token1`, then the function will revert due to underflow during transfer of the `amountOut` without a specific error message.

For function `swapReserveToNative()`, if the `amountOut` exceeds the corresponding pair balance value, then the function will revert due to underflow during transfer of the `amountOut` without a specific error message.

Likewise, the conversion of `_reserveBaseValue` in both `swapNativeToReserve()` and `swapReserveToNative()` first converts the `_reserveBaseValue` using `vPool` balances, then converts again (if needed) using the pair balances of the `vPair` contract. For instance, in function `swapNativeToReserve()`, if `vPool.balance0` is notably distinct from `pairBalance1` this could cause issues with the resulting conversion of `_reserveBaseValue` from the out-token to `token0` of the `vPair` contract.

The fact that this difference in conversion only occurs if the reserve swap did not involve `token0` of the `vPair` opens the update up to further possible inconsistency, since for some virtual swaps, the pair balances will be involved and for other virtual swaps, only `vPool` balances will be involved.

Recommendation

We recommend adding checks for the `amountOut` against the corresponding values before attempted transfer of the `amountOut`.

We additionally recommend accounting for the possibility of differences in the `vPair` pair balances and the `vPool` recorded balances in updates to the `reserveBaseValue` mapping in each function.

Alleviation

`[CertiK]` : The team resolved the first item in the description in commit de375fba7b852b256defd3067dafe4a1c02c5a3a.

The team makes the following statement regarding the second item in the description:

`[Virtuswap]` : "In quote function the ratio between two balances is crucial, not the actual values. So even if vPool.balance1 is not equivalent pairBalance0, the conversion will be correct."

`[CertiK]` : Since the team states that it is intended design for these values of `vPool.balance1` and `pairBalance0` to be distinct, the finding is considered resolved. Furthermore, we note that, whether two references to the same `jkPair` and `ikPair` are used in the `vPoolManager`'s function `getVirtualPool()` in the same block or in two different blocks, the `vPool.balance` value should always be less or equal to its corresponding `pairBalance0` value, either through the use of function `_reduceBalances()` in `vPoolManager` in the case where both calls have occurred in the same block, or through the function `calculateVPool()` in `vSwapLibrary` in the case where the calls have occurred in separate blocks. In both cases, the reduction to virtual pool balances appears consistent.

CON-02 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/vPair.sol (updated base): 268~269, 542~543; contracts/vPairFactory.sol (updated base): 70~71, 91~92, 103~104, 146~151	Resolved

Description

Contract `vPairFactory`

- The function `setDefaultAllowList()` is missing a check that the length is not larger than $2^{24} - 1$; if it is larger, the `maxAllowListCount` will overflow due to the `uint24` casting;
- The function `setDefaultAllowList()` is missing a check that there are no repeated addresses in the array. Any repeated addresses will cause the included address to be double counted in `reserveBaseSum()`, which causes a miscalculation in library function `getMaxVirtualTradeAmountRtoN()`;
- The functions `setVPoolManagerAddress()` and `setExchangeReservesAddress` are missing checks that the respective variables `pairFactory` and `factory` are set to `address(this)`;

Contract `vPair`

- The function `setAllowList()` is missing a check that there are no repeated addresses in `_allowList`. Any repeated addresses will cause the included address to be double counted in `reserveBaseSum()`, which causes a miscalculation in library function `getMaxVirtualTradeAmountRtoN()`;
- The function `swapNativeToReserve()` is missing a check that the `balanceDiff` is at least the `requiredAmountIn`. The function will revert without returning a specific error message due to underflow if this is not the case.
- The function `setReserveRatioWarningThreshold()` is missing a check that the updated input lies within a predetermined expected range for the value.

Recommendation

We recommend adding the input validation in each scenario described above.

Alleviation

`[CertiK]`: The team resolved the finding in commit fc4027cc39d01657000dbf6361f0f9ad4e4ec90a and 2a42a79077e67c375e2448ed3af978d924047089.

CON-04 `allowList` TOKEN UPDATES SHOULD BE CAREFULLY CONSIDERED

Category	Severity	Location	Status
Volatile Code	Minor	contracts/vPair.sol (updated base): 542~543; contracts/vPairFactory.sol (updated base): 146~147	Resolved

Description

The update of `allowList` and `defaultAllowList` should be executed with caution. Discrepancies between the `allowList` in two related `vPair` contracts may cause unintended relationships between certain tokens.

- `vPair` token addresses should not be included on the `allowList` of a given pair contract.
- `allowList` in a `vPair` should not be allowed to contain any of the tokens used that make up the `vPair` itself, but there is no prevention from this occurring in `vPairFactory` or in function `updateAllowList()` in the `vPair` contract itself. If either `token0` or `token1` is present on the `allowList`, then the virtual swap functionality may be used rather than `swapNative()`.

Recommendation

We recommend adding in prevention in `vPairFactory` that prevents a pair from being created which contains a token on its own `allowList`.

Additionally, we recommend vetting each token on the `allowList` carefully for compatibility with the logic of the exchange.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit 5d98982301028517ba1fb75dd240d7381f89c754.

CON-05 SWAPPING FUNCTIONS REQUIRE A STRICT INEQUALITY CHECK FOR `amountOut`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/libraries/vSwapLibrary.sol (updated base): 55~56; contracts/vPair.sol (updated base): 157~158, 159~164, 352~353, 357~362	Resolved

Description

In function `swapReserveToNative()`, `amountOut` cannot exceed `vPool.balance1`. However, in the case where `amountOut` is equivalent to `vPool.balance1`, the check passes, but the function will revert in the call to `getAmountIn()`, since in that case `amountOut` and `pairBalanceOut` are equal, causing the denominator to be 0.

The same issue is present in function `swapNative()`.

Recommendation

We recommend the use of a strict inequality when comparing `amountOut` to `vPool.balance1` so that a specific error message is thrown in this case.

Alleviation

`[CertiK]`: The team resolved the finding in commit 0d093cb610de71f7e57134bbe704760fafd65d82.

PVB-03 FUNCTION CHECKS RECORDED `_balanceOut` AFTER TRANSFER

Category	Severity	Location	Status
Logical Issue	Minor	contracts/vPair.sol (updated base): 146~147, 157~158	Resolved

Description

Function `swapNative()` checks that `amountOut` does not exceed `_balanceOut` (the corresponding pair balance), after `safeTransfer()` has already been called to transfer the `amountOut`.

If `amountOut` exceeds the `tokenOut` balance of the `vPair` contract, then the call will revert due to underflow without a specific error message.

Recommendation

We recommend moving the safe transfer of the `tokenOut` after the check that `amountOut` does not exceed `_balanceOut`.

Alleviation

`[CertiK]`: The team resolved the finding in commit 038938e99ddd9bb5f50120fea3775aa2a441184b.

RVB-02 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/vRouter.sol (updated base): 101	Resolved

Description

The return value of the transfer()/transferFrom() call is not checked.

```
101             IWETH9(WETH9).transfer(msg.sender, requiredBackAmount);
```

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We recommend using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

`[CertiK]`: The team resolved the finding in commit 4e878abfd05a4f5593e95bf7c3e701b37d7f7c1a.

RVB-03 | POTENTIAL LOSS OF ETHER

Category	Severity	Location	Status
Control Flow	Minor	contracts/vRouter.sol (updated base): 94, 108~115, 125~126, 131, 152~153, 158, 189, 219	Resolved

Description

If a user calls a `payable` function and includes a positive `msg.value` but the `tokenIn` is an address other than `WETH`, where the user has given allowance for the `tokenIn` they specified, then the `msg.value` will be left in `vRouter` and can potentially be extracted by another account.

Recommendation

We recommend preventing the inclusion of `msg.value` in the case where `tokenIn` is not `WETH`.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit e0389006a555c7968bbe9c9d9a755c9e3d17072e.

SLV-03 | POTENTIAL DIVISION BY ZERO

Category	Severity	Location	Status
Volatile Code	Minor	contracts/libraries/vSwapLibrary.sol (remediation_update1): 196~219, 245~246, 247~248, 253~254, 257~258, 267~268, 271~272, 281~282, 285~286, 295~296, 299~300, 309~310, 313~314, 323~324, 327~328, 337~338, 341~342, 348~358	Partially Resolved

Description

It is possible for `params.vBalance0` to be 0, in which case, the resulting values for `temp` and `derivative` will also be 0. In this instance, the following update to `maxAmountIn` will revert due to division by 0 without a specific error message:

```
maxAmountIn +=
  Math.mulDiv(c1, uc2, derivative) -
  Math.mulDiv(maxAmountIn, temp, derivative);
```

Similar cases are cited above.

Recommendation

We recommend adding a check that ensures `derivative` is nonzero before dividing.

Alleviation

`[CertiK]`: The team made changes based on this finding in commit f1306da14917a10ddd6c31de0d0a5c0f854e1dfb.

In the changes, the team provided further documentation on how the newly added requirements should ensure calculated values lie within expected bounds.

Please the points below concerning the argument in lines 241 - 248:

1. In line 248, `r` is replaced by an upper bound of 10^{32}; however, there do not appear to be any restraints within the `vPair` contract enforcing that values in the `reserves` mapping do not exceed this value.
2. The line 248 is incorrect. Based on the previous line, the expression should read

$$10^8 \cdot (10^{64} \cdot 2 \cdot 10^5 + 2 \cdot 10^{64})$$

$$= 10^8 \cdot (2 \cdot 10^{64}) \cdot (10^5 + 1) \leq 2 \cdot 10^{77} + 2 \cdot 10^{72}$$

Even in truncating the trailing term $2 \cdot 10^{72}$, the calculation for `b` could get as large $2 \cdot 10^{77}$ which is larger than the maximum `int256` value by

142103955381341902288214507495656046073365007667179717980271207996043435180033.

Since overflow is still possible, it is still possible for the `derivative` to be 0.

`[CertiK]`:

1. Regarding the first issue stated about `reserves`, the team states they will not add any checks in vPair contract, but intend to find a way to calculate `maxVirtualAmountRtoN` without upper bound of 10^32 (using uint512 library for example).
2. The team made changes related to this finding in commit 1e9bfe82f1ff07b1e3e9aa82ddbe52aa83d978db.

The change to division by the `params.fee` value resolves the issue stated in point 2 of the alleviation above.

The only potential underflow left comes from casting the return `uint256` value of `Math` function `mulDiv()` as an `int256` type, starting in line 323 (and each time it is used after). Function `mulDiv()` will revert if the operations produce an overflow of the `uint256` type, but the `uint256` value returned may still be larger than the `max(int256)` value.

Upon resolution of the underflow due to integer casting, this finding may be partially resolved, since the `reserves` value may still be larger than 10^32.

`[CertiK]`: The team made changes partially resolving this finding in commit 57447c4b96279358b43575d41656baf3e0504d44.

SLV-04 UNSAFE INTEGER CAST AND POTENTIAL UNDERFLOW/OVERFLOW

Category	Severity	Location	Status
Logical Issue	Minor	contracts/libraries/vSwapLibrary.sol (remediation_update1): 193~195, 196~197, 198~202, 203~213, 214~219, 221~222, 222~234, 223~227, 228~234, 248~249, 250~251, 252~254, 256~257, 262~263, 264~265, 266~268, 270~271, 276~277, 278~279, 280~282, 284~285, 290~291, 292~293, 294~296, 298~299, 304~305, 306~307, 306~307, 308~310, 312~313, 318~319, 320~321, 322~324, 326~327, 332~333, 334, 336~338, 340~341, 348~359, 351~354	Partially Resolved

Description

1. If values `reserveRatioFactor` and `params.vBalance1` are allowed to grow over time, then `uint256` value `a` may overflow, since its calculation is inside an `unchecked` box. In instances where pair balance liquidity is deep, it may be possible for this to occur through repeated calls to `swapReserveToNative()` using amounts which cause `requiredAmountIn` to not exceed the value of `getMaxVirtualTradeAmountRtoN(vPool)`.
2. If `params.vBalance0` is larger than $2^{255} - 1$ then the casting to `int256(params.vBalance0)` in `int256` value `b` will result in an overflow of the value, causing it to be an unintended negative value;
3. In the following expression:

```
int256 b = int256(params.vBalance0) *
                (-2 *
                    int256(
                        params.balance0 *
                            params.fee *
                            params.maxReserveRatio
                    ) +
                    int256(
                        params.vBalance1 *
                            (2 *
                                params.fee *
                                params.maxReserveRatio +
                                params.priceFeeFactor *
                                params.reserveRatioFactor) +
                            params.fee *
                            params.reserveRatioFactor *
                            params.reservesBaseValueSum
                    )) +
                int256(
                    params.fee *
                        params.reserves *
                        params.reserveRatioFactor *
                        params.vBalance1
                );
```

Even if each individual integer term is less $2^{255} - 1$, the complete expression may still overflow if the inner sum or final product is greater than this upper bound.

4. In the following expression:

```
negativeB ? (a * maxAmountIn - ub) : (a * maxAmountIn + ub)
```

The expression `a * maxAmountIn - ub` may underflow if `ub` is greater than `a*maxAmountIn`. The expression `a*maxAmountIn + ub` may overflow if the sum exceeds the maximum `uint256` value.

The above is a sample of potential overflow/underflow and truncation due to unsafe integer casting.

Recommendation

We recommend implementing both of the following:

1. Remove each `unchecked` box *and*
2. Check the bounds of the starting input values of `params` so subsequent integer castings can be verified to stay within range.

Additionally we recommend providing explanation for how it is known each item will not cause truncation on conversion or produce an overflow/underflow respectively.

Alternatively to the second recommendation item, one can check the bounds of integer values before casting, so the values will not be truncated or flip the sign. The SafeCast library from OpenZeppelin can also be used in place of type casting.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/71aaca2d9db465560213740392044b2cd3853a3b/contracts/utils/math/SafeCast.sol

Alleviation

`[CertiK]`: The team made changes based on this finding in commit f1306da14917a10ddd6c31de0d0a5c0f854e1dfb.

In the changes, the team provided further documentation on how the newly added requirements should ensure calculated values lie within expected bounds.

Please the points below concerning the argument in lines 241 - 248:

1. In line 248, `r` is replaced by an upper bound of 10^{32}; however, there do not appear to be any restraints within the `vPair` contract enforcing that values in the `reserves` mapping do not exceed this value.
2. The line 248 is incorrect. Based on the previous line, the expression should read

$$10^8 \cdot (10^{64} \cdot 2 \cdot 10^5 + 2 \cdot 10^{64})$$

$$= 10^8 \cdot (2 \cdot 10^{64}) \cdot (10^5 + 1) \leq 2 \cdot 10^{77} + 2 \cdot 10^{72}$$

Even in truncating the trailing term $2 \cdot 10^{72}$, the calculation for `b` could get as large $2 \cdot 10^{77}$ which is larger than the maximum `int256` value by 1421039553813419022882145074956560460733650076671797179802712079960434351800 33.

Since overflow is still possible, it is still possible for the `derivative` to be 0.

`[CertiK]`:

1. Regarding the first issue stated about `reserves`, the team states they will not add any checks in vPair contract, but intend to find a way to calculate `maxVirtualAmountRtoN` without upper bound of 10^32 (using uint512 library for example).
2. The team made changes related to this finding in commit 1e9bfe82f1ff07b1e3e9aa82ddbe52aa83d978db.

The change to division by the `params.fee` value resolves the issue stated in point 2 of the alleviation above.

The only potential underflow left comes from casting the return `uint256` value of `Math` function `mulDiv()` as an `int256` type, starting in line 323 (and each time it is used after). Function `mulDiv()` will revert if the operations produce an overflow of the `uint256` type, but the `uint256` value returned may still be larger than the `max(int256)` value.

Upon resolution of the underflow due to integer casting, this finding may be partially resolved, since the `reserves` value may still be larger than 10^32.

`[CertiK]` : The team made changes partially resolving the finding in commits 57447c4b96279358b43575d41656baf3e0504d44 and 766e62a8499026764a5ca9164eccb3b0c6fe9938.

CON-06 | DISCUSSION ON `_update()` DESIGN

Category	Severity	Location	Status
Inconsistency	Informational	contracts/libraries/vSwapLibrary.sol (updated base): 122~123; contracts/vPair.sol (updated base): 95~98, 105~106; contracts/vPoolManager.sol (updated base): 35~37	Resolved

Description

Function `_update()` always updates the values `pairBalance0` and `pairBalance1`, which act as the official recorded balances for the native tokens in a pair, and are used in determining the price exchange between the two tokens.

However, `_lastBlockUpdated`, `_lastPairBalance0` and `_lastPairBalance1` are only updated if the current `block.number` is 3 blocks past the current value of `_lastBlockUpdated`. These are the values referenced for `ikBalance0` and `ikBalance1` in library function `getVirtualPool()`. This function is called in function `getVirtualPool()` in contract `vPoolManager` whenever the current `block.number` is not the `blockLastUpdated` recorded in mapping `vPoolsCache`.

Recommendation

Please provide more information on the design choice that the values of `_lastPairBalance0` and `_lastPairBalance1` are only updated after three blocks have passed.

Alleviation

`[Virtuswap]`: The 2-block delay for _lastPairBalance0 and _lastPairBalance1 is implemented to counteract potential virtual pool price manipulation, where an individual could artificially alter the price in the oracle pool and instantaneously trade in the virtual pool. Both _lastPairBalance0 and _lastPairBalance1 are solely utilized by the ikPair (oracle) to determine the price during a virtual exchange involving two tokens. Consequently, the focus is on the ratio between the two, rather than their actual values.

CON-08 | SHADOWING LOCAL VARIABLE

Category	Severity	Location	Status
Coding Style	Informational	contracts/vPair.sol (updated base): 461, 501; contracts/vSwapERC20.sol (updated base): 42	Resolved

Description

A local variable is shadowing another component defined elsewhere.

```
461         uint256 _totalSupply = totalSupply();
```

- Local variable `_totalSupply` in `vPair.mint` shadows the variable `_totalSupply` in `vSwapERC20`.

```
501         uint256 _totalSupply = totalSupply();
```

- Local variable `_totalSupply` in `vPair.burn` shadows the variable `_totalSupply` in `vSwapERC20`.

Recommendation

We recommend removing or renaming the local variable that shadows another definition.

Alleviation

`[CertiK]`: The team resolved the finding in commit d05c16de6acce308f1abc42378e8685617f5148f.

CON-09 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/vExchangeReserves.sol (updated base): 22~23; contracts/vPair.sol (updated base): 587, 593; contracts/vPairFactory.sol (updated base): 102	Resolved

Description

Events should be emitted in sensitive functions controlled by centralization roles.

Recommendation

We recommend emitting events in the functions cited above.

Alleviation

`[CertiK]` : The team resolved the finding in commit 87ece3e3f9f3c382652a95f22673cf11ecc43e7e.

ERV-01 CHECKS DO NOT GUARANTEE CONSISTENT `factory` ADDRESS IS USED ACROSS PAIRS

Category	Severity	Location	Status
Inconsistency	Informational	contracts/vExchangeReserves.sol (updated base): 79~88	Resolved

Description

In function `exchange()`, it is checked that a pair exists in `factory` for the tokens comprising `jkPair1` and `jkPair2`.

It is possible that a pair exists in `factory` corresponding to tokens `_jkToken0` and `_jkToken1`, but that the `jkPair1` address is not the same as the address returned from `getPair()` in the `vExchange` stored `factory` address.

This does not appear subject to exploit, since `ikPair1` is required to have the same `factory` address as `jkPair1` within the `vPair` contract logic, and `jkPair1` still requires an amount of a specific `vPool.token0` address to be returned, which is specific to the `factory` corresponding to the `jkPair1` address.

Recommendation

We recommend ensuring that this specification meets the intended design of the project.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit d78d6ffa2c93c1177883cbe5bb33bd8fe4931b39.

PFV-02 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Coding Style	Informational	contracts/vPairFactory.sol (updated base): 13~14, 41~47	Resolved

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

Recommendation

We recommend removing the explicit `getPair()` function and instead renaming `pairs` as `getPair`, relying on the compiler-generated getter function for `pairs`, since the variable is already public.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit fe51258c9fd02be105b828e63a443fdd4191ef24.

PVB-04 MISSING CHECK AGAINST `reserveRatioWarningThreshold` FOR NON-PRIVILEGED USERS

Category	Severity	Location	Status
Control Flow	Informational	contracts/vPair.sol (updated base): 214~215, 236~249	Resolved

Description

Function `swapNativeToReserve()` only performs a check that the return value `calculateReserveRatio` does not exceed `reserveRatioWarningThreshold` when the `msg.sender` is the `admin` specified in the `vPair` contract's `factory`.

Please provide more information on why this check is unnecessary in a typical user's case.

Recommendation

We recommend providing more information on why this check is unnecessary in a typical user's case.

Alleviation

[CertiK]: The team states that the `reserveRatioWarningThreshold` is for admin purposes only, and not meant to be used as prevention for typical users when swapping. They further note this is the threshold at which the admin account may begin manually swapping.

We note for informational purposes that the check against `reserveRatioWarningThreshold` does not prevent the `calculateReserveRatio` value from going below this threshold as a consequence of a swap by an admin. Initially, the reserve ratio may meet the threshold, but after a large enough swap, may be below this lower bound.

PVB-05 DISCUSSION ON ADMIN VERSUS EMERGENCYADMIN IN `vPair`

Category	Severity	Location	Status
Control Flow	Informational	contracts/vPair.sol (updated base): 211~217, 242~249, 595	Resolved

Description

Both `admin` and `emergencyAdmin` are able to call function `swapNativeToReserve()`, with an adjustment to `requiredAmountIn` potentially including a deep `emergencyDiscount` (up to 100%). The `emergencyAdmin` is able to adjust `emergencyDiscount`, and when this role calls `swapNativeToReserve()`, there are no checks that the `reserveRatioWarningThreshold` is respected.

Please provide more information on the intended difference in the respective roles of `admin` and `emergencyAdmin` in a `vPairFactory` contract, since on deploy of `pairFactory`, the role is given to the same account (`msg.sender`).

The **Allowed Reserves** section of the site documentation states "the [allow] list is governed by VirtuSwap DAO and can be updated from time to time, adding or removing certain assets." Is the `admin` role meant to represent the VirtuSwap DAO?

Recommendation

We recommend providing more information on the intended difference in the roles.

Alleviation

`[Virtuswap]`: "DAOs possess a set of privileges on protocols, and there is also an Emergency DAO in place. Within the swapNativeToReserve function, a liquidation feature exists that can be triggered by the DAO, provided that the pool reserve ratio is equal to or greater than the reserveRatioWarningThreshold. In exceptional situations, such as a token crash like Luna, the Emergency DAO should be granted the flexibility to make various decisions regarding the liquidation of the reserve asset.

On deploy both admin and emergencyAdmin are assigned to msg.sender, but we plan to change it on DAO release."

`[CertiK]`: We encourage the team to take necessary precautions when transitioning to a DAO for the privileged roles, such as the use of a time-lock. Please see the Centralization Related Risks finding of the report for more information.

PVB-06 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	contracts/vPair.sol (updated base): 77	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit e7df89253867cbf90541c720949525c9b1782eba.

PVB-07 | TYPO

Category	Severity	Location	Status
Coding Style	Informational	contracts/vPair.sol (updated base): 293, 387	Resolved

Description

In functions `swapNativeToReserve()` and `swapReserveToNative()`, the word "balance" in the referenced comment is misspelled as "blance."

Recommendation

We recommend correcting the typo.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit f68dc5c5fcf3f9ad96e284eff1ffa1adce224e47.

SLV-01 DOCUMENTATION NEEDED ON FUNCTION `getMaxVirtualTradeAmountRtoN()`

Category	Severity	Location	Status
Coding Style	Informational	contracts/libraries/vSwapLibrary.sol (updated base): 146~313	Resolved

Description

Please provide documentation on the derivation of the formula for the returned value `maxAmountIn` so that it may be checked that the implementation meets the specification. If the formula is present within the team's whitepaper, please specify the section in which it may be referenced.

Recommendation

We recommend providing more information on the formula used in function `getMaxVirtualTradeAmountRtoN()`.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit cc560f3ef20e240e6ac77737f0b50a02dc6d0ba1.

SLV-02 | UNUSED FUNCTION

Category	Severity	Location	Status
Coding Style	Informational	contracts/libraries/vSwapLibrary.sol (updated base): 93~100	Resolved

Description

Function `getVirtualPoolBase()` no longer contains any logic, and is not referenced within the protocol.

Recommendation

We recommend removing the function.

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit 261c5d5e2fb76f28c62a1e5355cb01c40dbcc9b0.

TYP-01 | VARIABLE NAMES TOO SIMILAR

Category	Severity	Location	Status
Coding Style	Informational	contracts/types.sol (updated base): 5~16	Resolved

Description

Use of short, similar variable names makes it difficult to keep track of the meaning of each variable when it is used.

Recommendation

We recommend renaming variables so their names are not too similar and convey the meaning more clearly.

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit b3cee05a45a30fbe5286bcdcb9cf68dc8a1e792f.

OPTIMIZATIONS | VIRTUSWAP - 2ND AUDIT

ID	Title	Category	Severity	Status
CON-10	Unnecessary Check On `amountOut`	Gas Optimization	Optimization	Resolved
CON-11	Unnecessary Check That `amountIn` Is Positive	Gas Optimization	Optimization	Resolved
PFV-03	Unneeded Check That `msg.sender` Is Not `address(0)`	Gas Optimization	Optimization	Resolved

CON-10 | UNNECESSARY CHECK ON `amountOut`

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/libraries/vSwapLibrary.sol (updated base): 77~78; contracts/vPair.sol (updated base): 218~219, 236~240	Resolved

Description

Function `swapNativeToReserve()` makes redundant checks that `amountOut` is positive. It is checked first directly in the function logic, and is also checked in call to `vSwapLibrary` function `quote()`.

Recommendation

We recommend only checking this property once.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit 77e3712bb5ddcbd8fb76f596b014f119f1a92655.

CON-11 | UNNECESSARY CHECK THAT `amountIn` IS POSITIVE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/libraries/vSwapLibrary.sol (updated base): 56~57; contracts/vPair.sol (updated base): 382~383	Resolved

Description

The use of `getAmountIn()` to calculate `requiredAmountIn` in function `swapReserveToNative()` guarantees that the value is positive (it is at least 1, provided the calculation does not revert). As such, since the `amountIn` is checked that it is at least the `requiredAmountIn`, there is no need to also check that `amountIn` is positive.

Recommendation

We recommend removing the requirement that `amountIn` is positive, since this is guaranteed by the check that `amountIn` is at least `requiredAmountIn`.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit 82db81195bd1939a6fa881bcb2cfb64ed5c276f0.

PFV-03 | UNNEEDED CHECK THAT `msg.sender` IS NOT `address(0)`

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/vPairFactory.sol (updated base): 121, 138	Resolved

Description

It is not possible for the `msg.sender` to be `address(0)` .

Recommendation

We recommend removing this check since it is a statistical impossibility that the `msg.sender` is `address(0)` .

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit deea29c4618696a65e2b09c5b2c40578aa4e87f0.

FORMAL VERIFICATION | VIRTUSWAP - 2ND AUDIT

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
erc20-transfer-change-state	`transfer` Has No Unexpected State Changes
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transfer-correct-amount-self	`transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Non-self Transfers

Property Name	Title
erc20-transferfrom-succeed-self	`transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	`transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-approve-change-state	`approve` Has No Unexpected State Changes
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address

Property Name	Title
erc20-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address

Verification Results

For the following contracts, model checking established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract vPair (contracts/vPair.sol) In Commit f7f7f170514f9ce111268bad1a26f424935adf4f

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-succeed-self	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-correct-amount	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-change-state	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

Detailed Results For Contract vSwapERC20 (contracts/vSwapERC20.sol) In Commit f7f7f170514f9ce111268bad1a26f424935adf4f

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-correct-amount	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
    - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
    - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this

report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.

- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
  - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
  - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract WETH9 (contracts/interfaces/external/WETH9.sol) In Commit f7f7f170514f9ce111268bad1a26f424935adf4f

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-change-state	Inapplicable	
erc20-transfer-correct-amount	True	
erc20-transfer-false	Inapplicable	
erc20-transfer-correct-amount-self	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-change-state	Inapplicable	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-false	Inapplicable	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-never-return-false	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-correct-value	Inapplicable	
erc20-totalsupply-change-state	Inapplicable	
erc20-totalsupply-succeed-always	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-change-state	Inapplicable	
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-change-state	Inapplicable	
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-change-state	Inapplicable	
erc20-approve-false	Inapplicable	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	True	
erc20-approve-never-return-false	True	

APPENDIX | VIRTUSWAP - 2ND AUDIT

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

Technical Description

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and Simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any function. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled using modular arithmetic based on the bit-width of the underlying numeric Solidity type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for Property Specification

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time step. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates as atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's

public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of the Analyzed ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`. In the following, we list those property specifications.

Properties related to function `transfer`

erc20-transfer-revert-zero

`transfer` Prevents Transfers to the Zero Address. Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address. Specification:

```
[](started(contract.transfer(to, value), to == address(0)) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
    == false)))
```

erc20-transfer-succeed-normal

`transfer` Succeeds on Admissible Non-self Transfers. All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transfer(to, value), to != address(0) && to != msg.sender &&
    value >= 0 && value <= _balances[msg.sender] && _balances[to] + value <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true)))
```

erc20-transfer-succeed-self

`transfer` Succeeds on Admissible Self Transfers. All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transfer(to, value), to != address(0) && to == msg.sender &&
    value >= 0 && value <= _balances[msg.sender] && _balances[msg.sender] >= 0 &&
    _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true)))
```

erc20-transfer-correct-amount

`transfer` Transfers the Correct Amount in Non-self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address. Specification:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender && _balances[to] >= 0
    && value >= 0 && _balances[to] + value <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[msg.sender] >= 0 && _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true ==>
      _balances[msg.sender] == old(_balances[msg.sender]) - value && _balances[to]
      == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

`transfer` Transfers the Correct Amount in Self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`. Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender && _balances[to] >= 0
    && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true ==> _balances[to] ==
      old(_balances[to]))))
```

erc20-transfer-change-state

`transfer` Has No Unexpected State Changes. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses. Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to) ==>
  <>(finished(contract.transfer(to, value), return == true ==> (_totalSupply ==
        old(_totalSupply) && _allowances == old(_allowances) && _balances[p1] ==
        old(_balances[p1]) && other_state_variables ==
        old(other_state_variables)))))
```

erc20-transfer-exceed-balance

`transfer` Fails if Requested Amount Exceeds Available Balance. Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail. Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender] &&
    _balances[msg.sender] >= 0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
      == false)))
```

erc20-transfer-recipient-overflow

`transfer` Prevents Overflows in the Recipient's Balance. Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow. Specification:

```
[](started(contract.transfer(to, value), to != msg.sender && _balances[to] + value
    >= 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000 && value >
    0 && value <= _balances[msg.sender]) ==> <>(reverted(contract.transfer) ||
    finished(contract.transfer(to, value), return == false) ||
    finished(contract.transfer(to, value), _balances[to] > old(_balances[to]) +
      value -
      0x1000000000000000000000000000000000000000000000000000000000000000)))
```

erc20-transfer-false

If `transfer` Returns `false`, the Contract State Is Not Changed. If the `transfer` function in contract `contract` fails by returning `false`, it must undo all state changes it incurred before returning to the caller. Specification:

```
[](willSucceed(contract.transfer(to, value)) ==> <>(finished(contract.transfer(to,
        value), return == false ==> (_balances == old(_balances) && _totalSupply ==
        old(_totalSupply) && _allowances == old(_allowances) &&
        other_state_variables == old(other_state_variables)))))
```

erc20-transfer-never-return-false

`transfer` Never Returns `false`. The transfer function must never return `false` to signal a failure. Specification:

```
[](!(finished(contract.transfer, return == false)))
```

Properties related to function `transferFrom`

erc20-transferfrom-revert-from-zero

`transferFrom` Fails for Transfers From the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-revert-to-zero

`transferFrom` Fails for Transfers To the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-succeed-normal

`transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0) && to !=
    address(0) && from != to && value <= _balances[from] && value <=
    _allowances[from][msg.sender] && _balances[to] + value <
    0x10000000000000000000000000000000000000000000000000000000000000000 && value >=
    0 && _balances[to] >= 0 && _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] >= 0 && _allowances[from][msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))
```

erc20-transferfrom-succeed-self

`transferFrom` Succeeds on Admissible Self Transfers. All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0) && from == to
    && value <= _balances[from] && value <= _allowances[from][msg.sender] && value
    >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))
```

erc20-transferfrom-correct-amount

`transferFrom` Transfers the Correct Amount in Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0 &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] + value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      _balances[from] == old(_balances[from]) - value && _balances[to] ==
      old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

`transferFrom` Performs Self Transfers Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`). Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to && value >= 0 &&
    value < 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

`transferFrom` Updated the Allowance Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] >= 0 && _allowances[from][msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      ((_allowances[from][msg.sender] == old(_allowances[from][msg.sender]) -
      value) || (_allowances[from][msg.sender] ==
      old(_allowances[from][msg.sender]) && (from == msg.sender ||
        old(_allowances[from][msg.sender]) ==
        0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF))))))
```

erc20-transferfrom-change-state

`transferFrom` Has No Unexpected State Changes. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to &&
    (p2 != from || p3 != msg.sender)) ==> <>(finished(contract.transferFrom(from,
      to, amount), return == true ==> (_totalSupply == old(_totalSupply) &&
      _balances[p1] == old(_balances[p1]) && _allowances[p2][p3] ==
      old(_allowances[p2][p3]) && other_state_variables ==
      old(other_state_variables)))))
```

erc20-transferfrom-fail-exceed-balance

`transferFrom` Fails if the Requested Amount Exceeds the Available Balance. Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from] &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-fail-exceed-allowance

`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance. Any call of the form `transferFrom(from,`

`dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), msg.sender != from && value >
    _allowances[from][msg.sender] && _allowances[from][msg.sender] >= 0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom(from, to,
        value), return == false)))
```

erc20-transferfrom-fail-recipient-overflow

`transferFrom` Prevents Overflows in the Recipient's Balance. Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), from != to && _balances[to] +
    value >= 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    value < 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom(from, to,
        value), return == false) || finished(contract.transferFrom(from, to,
        value), _balances[to] > old(_balances[to]) + value -
      0x1000000000000000000000000000000000000000000000000000000000000000)))
```

erc20-transferfrom-false

If `transferFrom` Returns `false`, the Contract's State Is Unchanged. If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value)) ==>
  <>(finished(contract.transferFrom(from, to, value), return == false ==>
    (_balances == old(_balances) && _totalSupply == old(_totalSupply) &&
    _allowances == old(_allowances) && other_state_variables ==
    old(other_state_variables)))))
```

erc20-transferfrom-never-return-false

`transferFrom` Never Returns `false`. The `transferFrom` function must never return `false`. Specification:

```
[](!(finished(contract.transferFrom, return == false)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

`totalSupply` Always Succeeds. The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

`totalSupply` Returns the Value of the Corresponding State Variable. The `totalSupply` function must return the value that is held in the corresponding state variable of contract contract. Specification:

```
[](willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply, return
      == _totalSupply)))
```

erc20-totalsupply-change-state

`totalSupply` Does Not Change the Contract's State. The `totalSupply` function in contract contract must not change any state variables. Specification:

```
[](willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply,
      _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
      _allowances == old(_allowances) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

`balanceOf` Always Succeeds. Function `balanceOf` must always succeed if it does not run out of gas. Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

`balanceOf` Returns the Correct Value. Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      return == _balances[owner])))
```

erc20-balanceof-change-state

`balanceOf` Does Not Change the Contract's State. Function `balanceOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
      _allowances == old(_allowances) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

`allowance` Always Succeeds. Function `allowance` must always succeed, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

`allowance` Returns Correct Value. Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`. Specification:

```
[](willSucceed(contract.allowance(owner, spender)) ==>
  <>(finished(contract.allowance(owner, spender), return ==
    _allowances[owner][spender])))
```

erc20-allowance-change-state

`allowance` Does Not Change the Contract's State. Function `allowance` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.allowance(owner, spender)) ==>
  <>(finished(contract.allowance(owner, spender), _totalSupply == old(_totalSupply)
    && _balances == old(_balances) && _allowances == old(_allowances) &&
    other_state_variables == old(other_state_variables))))
```

Properties related to function `approve`

erc20-approve-revert-zero

`approve` Prevents Approvals For the Zero Address. All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address. Specification:

```
[](started(contract.approve(spender, value), spender == address(0)) ==>
  <>(reverted(contract.approve) || finished(contract.approve(spender, value),
    return == false)))
```

erc20-approve-succeed-normal

`approve` Succeeds for Admissible Inputs. All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas. Specification:

```
[](started(contract.approve(spender, value), spender != address(0)) ==>
  <>(finished(contract.approve(spender, value), return == true)))
```

erc20-approve-correct-amount

`approve` Updates the Approval Mapping Correctly. All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`. Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0) && value >=
    0 && value <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.approve(spender, value), return == true ==>
      _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

`approve` Has No Unexpected State Changes. All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes. Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0) && (p1 !=
      msg.sender || p2 != spender)) ==> <>(finished(contract.approve(spender,
        value), return == true ==> _totalSupply == old(_totalSupply) && _balances
      == old(_balances) && _allowances[p1][p2] == old(_allowances[p1][p2]) &&
      other_state_variables == old(other_state_variables))))
```

erc20-approve-false

If `approve` Returns `false`, the Contract's State Is Unchanged. If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller. Specification:

```
[](willSucceed(contract.approve(spender, value)) ==>
  <>(finished(contract.approve(spender, value), return == false ==> (_balances ==
      old(_balances) && _totalSupply == old(_totalSupply) && _allowances ==
      old(_allowances) && other_state_variables == old(other_state_variables)))))
```

erc20-approve-never-return-false

`approve` Never Returns `false`. The function `approve` must never returns `false`. Specification:

```
[](!(finished(contract.approve, return == false)))
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.