Quantstamp Security Assessment Certificate

October 7th 2022 — Quantstamp Verified

Klaytn-DEX

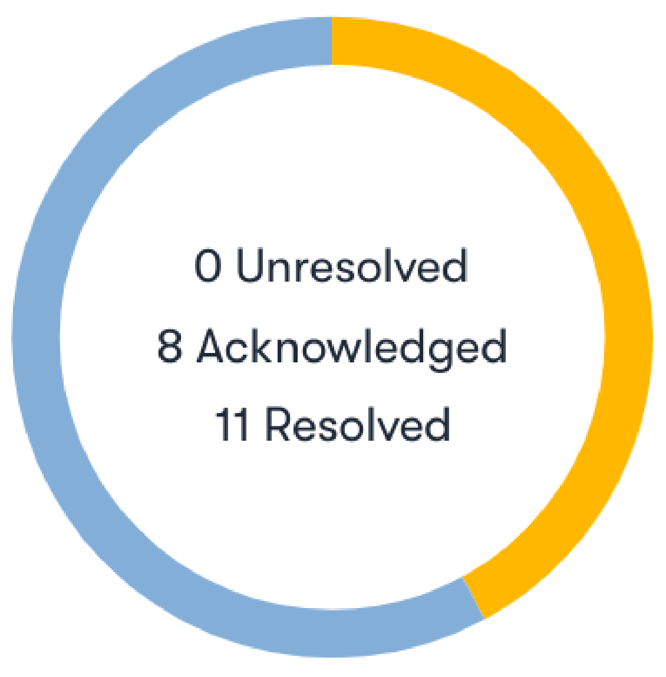
This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Defi
Auditors	Marius Guggenmos, Senior Research Engineer Mostafa Yassin, Security Engineer Jennifer Wu, Auditing Engineer Bohan Zhang, Auditing Engineer
Timeline	2022-09-12 through 2022-09-20
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Klaytn-DEX Specification
Documentation Quality	High
Test Quality	High
Source Code	

Repository	Commit
klaytn/klaytn-dex-contracts	6ad33cb initial audit
klaytn/klaytn-dex-contracts	970f878 fixes

Total Issues	**19** (11 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**1** (1 Resolved)
Low Risk Issues	**8** (5 Resolved)
Informational Risk Issues	**10** (5 Resolved)
Undetermined Risk Issues	0 (0 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Initial audit:
The audit covers four distinct components:

1. The decentralized exchange, which is a fork of the Uniswap V2 code (core, periphery) with improved code documentation and minor code cleanups.
2. The staking/farming contracts, which are using the reward distribution concept popularized by Sushiswap's MasterChef contract.
3. A multi-signature wallet based on Gnosis' MultiSigWallet.
4. The platform token, derived from Openzeppelin's ERC20 implementation.

Since the contracts the project is based off have been battle-tested on Ethereum's Mainnet, we did not expect finding any high severity issues. This turned out to be the case as we mostly found relatively minor issues.

The project's tests exercise around 90% of the code. Due to some small blind spots - which we expand upon in the test section - we classify the test quality as medium. Documentation wise, the code has a sufficient amount of comments and the specification mostly documents the intended behavior of the contracts.

Update after the fix verification:
The Klaytn team has addressed all of the reported issues either through fixes in the code, or by documenting risks in the specification. Additionally, tests have been added that remove any obvious blind spots and pushes the code coverage to above 95% for the common metrics, which lead to an increase in test quality from *medium* to *high*.

ID	Description	Severity	Status
QSP-1	Silent Loss of Farming Rewards	^ Medium	Fixed
QSP-2	Loss of Farming Rewards	⌄ Low	Fixed
QSP-3	Allowance Double-Spend Exploit	⌄ Low	Acknowledged
QSP-4	Zero Amount Emitted for `EmergencyWithdraw` Event	⌄ Low	Fixed
QSP-5	Missing Input Validation	⌄ Low	Mitigated
QSP-6	Revocable Roles	⌄ Low	Acknowledged
QSP-7	Single Step Administrator Change	⌄ Low	Acknowledged
QSP-8	Multisig Owners Can Vote on Past Transactions	⌄ Low	Fixed
QSP-9	Variable Precision Based on Decimals	⌄ Low	Fixed
QSP-10	Inaccurate Accounting if Deflationary Tokens Are Accepted	○ Informational	Acknowledged
QSP-11	Risk of Reentrancy when Transferring `DEXswap` Token	○ Informational	Acknowledged
QSP-12	Missing Events to Signal State Changes	○ Informational	Fixed
QSP-13	Clone-and-Own	○ Informational	Fixed
QSP-14	Not Explicitly Implementing Interface	○ Informational	Fixed
QSP-15	Year 2038 Problem	○ Informational	Acknowledged
QSP-16	`StakingFactory` Does Not Send Reward Tokens to Created Contract	○ Informational	Acknowledged
QSP-17	Adjustable Parameter Used for Address Derivation	○ Informational	Acknowledged
QSP-18	Require Statements that Always Pass	○ Informational	Fixed
QSP-19	Privileged Roles and Ownership	○ Informational	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

DISCLAIMER:
Please note that not all of the contracts in this repository were part of the audit. For the list of audited files, refer to the file signatures in the appendix of this report. Additionally, if the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Silent Loss of Farming Rewards

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `farming/Farming.sol`

Description: When withdrawing LP tokens from the farming protocol using `withdraw()`, the farming rewards are transferred to the user using `safePtnTransfer()`. If the `_amount` requested in `safePtnTransfer()` exceeds the protocol's balance, only the protocol's balance is transferred back to the user.
The `safePtnTransfer()` function does not revert in case the contract does not hold sufficient reward tokens at the time. This is especially problematic due to the contract fully relying on external monitoring and manually supplying reward tokens. A similar issue can arise in the `deposit()` function when claiming pending farming rewards.

Recommendation: Remove the `safePtnTransfer()` function and use `TransferHelper.safeTransfer()` directly.

Update: The function `safePtnTransfer()` has been removed.

QSP-2 Loss of Farming Rewards

Severity: *Low Risk*

Status: Fixed

File(s) affected: `farming/Farming.sol`

Description: When adding a new pool through the `add()` function or updating an existing pool setting through the `set()` function, the owner can set `_withUpdate` to control whether pools' rewards calculations are updated before setting the new allocations. If there is a pending reward calculation, and a pool is updated or added with the `_withUpdate` flag set to `false`, then the pending reward calculation will be less than expected due to the usage of the new allocations in the `updatePool()` function.

Exploit Scenario:

1. The owner adds a Pool A with 100% allocation using `add()` with option `_withUpdate` set to `true`.
2. Alice deposits LP tokens to Pool A using the `deposit()` function at time `t0`.
3. At time `t1`, the owner adds Pool B to 100% allocation using `add()` with option `_withUpdate` set to `false`. The owner sets Pool A to 0% allocation using `set()` with option `_withUpdate` set to `false`.
4. At time `t1`, Alice notices that Pool A allocation is set at 0% and tries to withdraw her LP tokens from Pool A. Alice expects 100% of the reward allocation between `t0` and `t1`, and tries to withdraw her LP tokens using the `withdraw()` function.
5. In the `withdraw()` function, `updatePool()` is called and calculates Alice's pending reward using the new `PoolInfo.allocPoint`, which is zero. Alice does not receive any farming rewards between `t0` and `t1`.

Recommendation: All pools should be updated using `massUpdatePools()` before proceeding with `add()` or `set()`. Remove the `_withUpdate` parameter and unconditionally call `massUpdatePools()` in `add()` and `set()`.

Update: The `_withUpdate` parameter has been removed and pools are now always updated before pool changes.

QSP-3 Allowance Double-Spend Exploit

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `swap/DexKIP7.sol`

Description: In `DexKIP7.sol`, the `approve()` function sets the allowance of a spender on behalf of the `msg.sender`. This `approve()` function is vulnerable to sandwich attacks, where the spender can use both the old and the new allowance due to transaction ordering.

Exploit Scenario:

1. Alice allows Bob to transfer N amount of Alice's tokens (N>0) by calling the `approve()` function on `DexKIP7` smart contract (passing Bob's address and N as function arguments)
2. After some time, Alice decides to change from N to M (M>0) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` function again, this time passing Bob's address and M as function arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` function to transfer N Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain an ability to transfer another M tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` function again, this time to transfer M Alice's tokens.

Recommendation: The exploit (as described above) can be mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()` (see: OpenZeppelin ERC20). We also recommend informing the developers of applications that are dependent on `approve()` / `transferFrom()` to first set allowance to zero and verify if it was used before setting the new value.

Update: Specification has been updated with a warning.

QSP-4 Zero Amount Emitted for `EmergencyWithdraw` Event

Severity: *Low Risk*

Status: Fixed

File(s) affected: `farming/StakingFactoryPool.sol`

Description: When withdrawing using `emergencyWithdraw()` in a staking pool, the amount emitted for the `EmergencyWithdraw` event will always be zero. The `user.amount` is set to zero at `StakingFactoryPool.sol#204`.

Recommendation: Update `user.amount` to `amountToTransfer` in `StakingFactoryPool.sol#215`.

Update: Implemented recommendation.

QSP-5 Missing Input Validation

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `farming/Farming.sol`, `farming/StakingFactory.sol`, `farming/StakingFactoryPool.sol`, `swap/DexFactory.sol`, `swap/DexKIP7.sol`, `swap/DexRouter.sol`

Description: It is crucial to validate inputs, even if the inputs come from trusted addresses, to avoid human error. A lack of robust input validation can only increase the likelihood and impact in the event of mistakes.
Following is the list of places that can potentially benefit from stricter input validation:

1. `Farming.sol#82`: The parameter `_ptn` should not be the zero address.
2. `Farming.sol#82`: The parameter `_multisig` should not be the zero address.
3. `Farming.sol#82`: The parameter `_startBlock` should be capped to ensure that the `startBlock` is not set too far in the future.
4. `Farming.sol#99`: The parameter `_pid` should confirm that `_pid` exists before updating the multiplier.
5. `Farming.sol#196`: The parameter `_pid` should confirm that `_pid` exists before the updating the `allocPoint`.
6. `Farming.sol#258`: The parameter `_pid` should confirm that `_pid` exists before proceeding with the deposit.
7. `Farming.sol#288`: The parameter `_pid` should confirm that `_pid` exists before proceeding with the withdraw.
8. `Farming.sol#326`: The parameter `_pid` should confirm that `_pid` exists before proceeding with the withdraw.
9. `Farming.sol#344`: The parameter `_pid` should confirm that `_pid` exists before proceeding with the calculation.
10. `StakingFactory.sol#11`: The parameter `_multisig` should not be the zero address.
11. `StakingFactory.sol#29`: The parameter `_multisig` should not be the zero address.
12. `StakingFactoryPool.sol#82`: The parameter `_startBlock` should be capped to ensure that the `startBlock` is not set too far in the future. In addition `_startBlock` should be less than `_rewardEndBlock`.
13. `StakingFactoryPool.sol#82`: The parameter `_rewardEndBlock` should be capped to ensure that the `_rewardEndBlock` is not set too far in the future.
14. `StakingFactoryPool.sol#224`: The parameter `_recipient` should not be the zero address.
15. `StakingFactoryPool.sol#234`: The parameter `_recipient` should not be the zero address.
16. `DexFactory.sol#43`: The parameter `_feeToSetter` should not be the zero address.
17. `DexFactory.sol#87`: The parameter `_feeToSetter` should not be the zero address.
18. `DexKIP7.sol#150`: The parameter `to` should not be the zero address; otherwise it is possible to burn tokens using `transfer()`.
19. `DexKIP7.sol#155`: The parameter `from` and `to` should not be the zero addresses; otherwise it is possible to burn tokens using `transferFrom()`. Before updating the allowance balance, check that `currentAllowance` is greater than the `amount` requested before updating the `allowance`.
20. `DexPair.sol#110`: The parameter `to` of `mint()` should not be the zero address. Note the code documentation states that this low level function should be called from another contract with the safety checks in place.
21. `DexPair.sol#176`: The parameter `to` of `burn()` should not be the zero address. Note the code documentation states that this low level function should be called from another contract with the safety checks in place.
22. `DexRouter.sol#24`: The parameter `_factory` and `_WKLAY` should not be the zero addresses.
23. `DexRouter.sol#104`: The parameter `to` should not be the zero address; otherwise the minted LP tokens will be locked. Note the `DexPair.mint()` low-level function does not include input validation.
24. `DexRouter.sol#160`: The parameter `to` should not be the zero address; otherwise the minted LP tokens will be locked. Note the `DexPair.mint()` low-level function does not include input validation.
25. `DexRouter.sol#212`: The parameter `to` should not be the zero address; otherwise the LP tokens will be locked. Note the `transferFrom()` function does not include input validation.
26. `DexRouter.sol#428`: The parameter `to` should not be the zero address.
27. `DexRouter.sol#691`: The parameter `_to` should not be the zero address.

Recommendation: Consider adding the checks listed in the description.

Update: Most of the checks have been added. Item 20 and 21 were not implemented. Item 12 and 13 were implemented partially.

QSP-6 Revocable Roles

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `farming/Farming.sol`, `farming/StakingFactory.sol`, `farming/StakingFactoryPool.sol`, `tokens/PlatformToken.sol`

Description: OpenZeppelin's `AccessControl` and `Ownable` contracts contain `revokeRole()`and `renounceOwnership()` functions which allow privileged addresses to give up their privileged roles. If privileged roles were renounced, all associated functionality associated with said role would be inaccessible.

Recommendation: For each privileged role, consider if role revocation is a necessary feature. If it is not, override the role revocation functionality to disable it. If it is, add end-user documentation stating the risk.

Update: The risks have been documented in the specification.

QSP-7 Single Step Administrator Change

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `swap/DexFactory.sol`

Description: The function `setFeeToSetter()` modifies the account that can control where the DEX fees are sent to. Since this is currently a single step process, a mistake when setting the address can result in loss of control over the fee destination.

Recommendation: Implement a two-step process as follows:

1. The current `feeToSetter` proposes a new fee setter.
2. The proposed address claims the ownership.

This process ensures that the new address is still under control of the team.

Update: The risks have been documented in the specification.

QSP-8 Multisig Owners Can Vote on Past Transactions

Severity: Low Risk

Status: Fixed

File(s) affected: `multisig/MultiSig.sol`

Description: The `MultiSigWallet` contract allows any owner to submit a new transaction. Once the required number of owners have confirmed this transaction, the final confirmation will execute it. Since the multisig accounts are intended to hold administrator privileges, it is important that only intended transactions are executed.
Since transactions never expire, removing and adding new owners might lead to transactions being passed that were never truly confirmed by the correct number of people.

Exploit Scenario:

1. The team consists of three people and they decide to deploy a 2/3 multisig for their project, i.e. at least two of the three accounts need to confirm the transaction.
2. One of the owners decides they want to pass a malicious transaction that transfers protocol tokens to them, so they submit a transaction that does exactly that.
3. The malicious owner now tells the other two that they lost access to their wallet, so they delete the old address and add the new one.
4. The new account can now be used to confirm the malicious transaction, passing the 2/3 check.

Recommendation: Keep track of the timestamps when owners and transactions were added. Enforce that owners can only confirm a transaction if the timestamp when the owner has been added is before the one of the transaction.

Update: Implemented recommendation.

QSP-9 Variable Precision Based on Decimals

Severity: Low Risk

Status: Fixed

File(s) affected: `farming/StakingFactoryPool.sol`

Description: The `StakingInitializable` contract computes a `PRECISION_FACTOR` that is used for extra precision in the rewards computation. While it works out in the most common case where the decimals are equal to 18, higher decimals can lead to severe rounding errors.

Recommendation: Since the decimals are already enforced to be lower than 30, using a constant factor of `10e18` instead of a dynamic one is most likely sufficient to avoid significant rounding errors in the rewards computation without leading to overflows.

Update: The code now uses a constant precision of `10e18`.

QSP-10 Inaccurate Accounting if Deflationary Tokens Are Accepted

Severity: Informational

Status: Acknowledged

File(s) affected: `farming/Farming.sol`, `farming/StakingFactoryPool.sol`

Description: The `deposit()` function updates the user balance using the `_amount` deposited in the contract. If deflationary tokens are accepted, internal accounting will be inaccurate for tokens that subtract fees during transfer. The value stored in the farming protocol may be different from the value tracked using the `_amount` in the `deposit()` function. Inaccurate accounting of deflationary tokens can lead to insolvency during withdrawal.
Deflationary token incompatible deposit functions are identified in the following functions:

1. `StakingFactoryPool.sol#127`: `deposit()`.
2. `Farming.sol#258`: `deposit()`.

Recommendation: Since the team controls which tokens are accepted, this finding is marked as informational. If the team wish to handle tokens with fees on transfer correctly, only account for the difference between the pre-transfer balance and the post-transfer balance of the treasury. The difference will capture the amount of tokens that have been transferred, regardless of the token transfer mechanism.

Update: Specification has been updated with documentation that tokens with "fee on transfer" functionality are not supported.

QSP-11 Risk of Reentrancy when Transferring `DEXswap` Token

Severity: Informational

Status: Acknowledged

File(s) affected: `swap/DexKIP7.sol`

Description: In `DexKIP7.sol#174`, the `safeTransfer()` and `safeTransferFrom()` functions invoke external calls to verify if the recipient address is a `KIP7Receiver` in `_checkOnKIP7Received()`. Since it is an external call to an unknown contract, where the `recipient` contract may define any arbitrary logic to be executed, it is possible to re-enter functions through usage of `safeTransfer()` and `safeTransferFrom()` functions.

Recommendation: Update `DexKIP7.sol`'s documentation to inform users of the risk of reentrancy when using the safe transfer functions `safeTransfer()` and `safeTransferFrom()`.

Update: A warning has been added to the specification.

QSP-12 Missing Events to Signal State Changes

Severity: Informational

Status: Fixed

File(s) affected: `swap/DexFactory.sol`

Description: Events are important for signalling state changes in a contract. This helps debug the contract in the event of any attacks or critical bugs and helps indicate to users any configuration changes in the contract.
A non-exhaustive list of functions where events can be useful includes:

1. `DexFactory.setFeeTo()`
2. `DexFactory.setFeeToSetter()`

Recommendation: Add events to signal the contract's state changes.

Update: Events have been added.

QSP-13 Clone-and-Own

Severity: Informational

Status: Fixed

File(s) affected: `utils/*`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from a security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as using libraries. If the file is cloned anyway, a comment including the repository, the commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve the traceability of the file.

Update: The utility contracts have been removed from the repository and replaced by the klaytn contracts npm package, which is a direct fork of the OpenZeppelin repository on github. This allows for easily viewing the differences between the OpenZeppelin and Klaytn contracts.

QSP-14 Not Explicitly Implementing Interface

Severity: Informational

Status: Fixed

File(s) affected: `tokens/PlatformToken.sol`

Description: The `PlatformToken` contract does not explicitly implement the `IPlatformToken` interface. This can lead to unnoticed divergences of the function signatures.

Recommendation: Make sure `PlatformToken` inherits from `IPlatformToken`. Note that this might require reordering the `IPlatformToken` inheritance order to

```
interface IPlatformToken is IAccessControl, IKIP7Metadata, IKIP7Permit, IVotes
```

Update: The `PlatformToken` contract now inherits from `IPlatformToken`.

QSP-15 Year 2038 Problem

Severity: Informational

Status: Acknowledged

File(s) affected: `swap/DexPair.sol`

Description: `DexPair.sol#105` casts the `block.timestamp` to `uint32`, which will wrap around in the year 2038, known as the *Year 2038 problem*.

Recommendation: This only affects the variables `price{0,1}CumulativeLast`, which are used for external price reporting and is not detrimental to the functionality of the DEX. It is important however to know that contracts relying on the price changes might not function correctly after the year 2038.

Update: The issues has been documented in the specification.

QSP-16 `StakingFactory` Does Not Send Reward Tokens to Created Contract

Severity: Informational

Status: Acknowledged

File(s) affected: `farming/StakingFactory.sol`

Description: The `StakingFactory` contract is responsible for deploying new `StakingInitializable` contracts. While the factory fully initializes the new contract, the reward tokens have to be sent manually after the deployment, which might be forgotten.

Recommendation: Send the reward tokens after deploying the pool using either `transferFrom()` with approvals or `transfer()` directly from the contract to ensure the contract is fully initialized and usable from the start.

Update: The specification for the `StakingFactory` has been updated to point out that tokens need to be sent manually to the created pool.

QSP-17 Adjustable Parameter Used for Address Derivation

Severity: Informational

Status: Acknowledged

File(s) affected: `farming/StakingFactoryPool.sol`

Description: `StakingFactory#46` uses the `startBlock` of the pool to derive the address via the `CREATE2` opcode. While the `startBlock` has not been reached, the owner of the `StakingInitializable` contract can change the start and end blocks. In addition to not being very useful due to the fact that adjustments can only be made before the rewards accumulation has started, it might be confusing to have an address derived from a parameter that is different on the actual contract.

Recommendation: Consider removing the `updateStartAndEndBlocks()` and `updateRewardPerBlock()` functions and fall back to deploying a new contract in case the parameters need to be adjusted.

Update: The fact that the start block can be changed after the fact has been documented in the specification.

QSP-18 Require Statements that Always Pass

Severity: Informational

Status: Fixed

File(s) affected: `farming/StakingFactory.sol`

Description: `StakingFactory.sol#39-40` performs the following input validation:

```
require(IKIP7(_stakedToken).totalSupply() >= 0);
require(IKIP7(_rewardToken).totalSupply() >= 0);
```

Since the return type of `totalSupply()` is `uint256`, this statement is always true.

Recommendation: Depending on whether the intent of the code was to verify that both tokens have some sort of supply, either modify it to check for greater than 0 or remove it entirely.

Update: The comparison now checks that the total supply of the tokens is strictly greater than 0.

QSP-19 Privileged Roles and Ownership

Severity: Informational

Status: Fixed

File(s) affected: `farming/StakingFactoryPool.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows the owner.
Contracts that have privileged roles:

- `PlatformToken.sol`: Admin can change who gets to mint and burn tokens.
- `Farming.sol`: Owner can update pool settings such as allocation points and rewards.
- `StakingFactory.sol`: Owner can deploy new pools.
- `StakingFactoryPool.sol`: Owner can withdraw reward tokens at any time.

Recommendation: Make sure to publish end-user documentation that clearly informs the users of the potential risks when privileged accounts are involved.

Update: Documentation of the privileged roles has been added to the specification.

Automated Analyses

Slither

Slither reported 401 results, many of which were false positives. The relevant results have been integrated into the findings or best practices of this report.

Adherence to Specification

1. The `user limit` concept is not documented.
   Update: Documentation has been added.
2. The `Type 2 Smart Contract` diagram states that each smart contract contains several pools when in reality each contract contains exactly one pool.
   Update: The diagram has been fixed.

Code Documentation

1. `Multisig.sol#200` the code documentation should be corrected to "Allows an owner to execute a confirmed transaction". The `ownerExists()` modifier is applied at `executeTransaction()` to confirm `msg.sender` ownership before invoking the function.
   Update: Fixed.
2. Correct typo at `Multisig.sol#296` from `filers` to `filters`.
   Update: Fixed.
3. At `DexPair.sol::L43`, "occured" should be "occurred".
   Update: Fixed.
4. At `DexPair.sol::L208`, "Arbitary" should be "Arbitrary".
   Update: Fixed.
5. At `DexPair.sol::L254`, "2112-1" should be "reserves".
   Update: Fixed.
6. At `StakingFactoryPool.sol::L357`, "rewads" should be "rewards".
   Update: Fixed.
7. At `StakingFactoryPool.sol::L222`, "trasfer" should be "transfer".
   Update: Fixed.
8. The documentation of parameter `_bonusEndBlock` in function `Farming.add()` is missing.

Update: Fixed.

9. The comment on `DexKIP7.sol#123` about needing to check that `from` has sufficient balance before calling `_transfer()` can be removed, as the check is performed by the function itself.
   Update: Fixed.

Adherence to Best Practices

1. `StakingFactoryPool.sol#210`: Redundant address cast applied to `msg.sender` can be removed.
   Update: Fixed.
2. `Multisig.sol#171`: `confirmTransaction()` should be renamed to `confirmAndExecuteTransaction()` to reflect its functionality.
   Update: Fixed.
3. Emit the event `OwnerAddition(owner)` at `Multisig.sol#100`.
   Update: Fixed.
4. The constraint of `ownerDoesNotExist(owner)` can be achieved by checking the return value of `owners.add(owner)` in `Multisig.addOwner()`. The constraint of `ownerExists(owner)` can be achieved by checking the return value of `owners.remove(owner)` in `Multisig.removeOwner()`. Similar gas optimizations can be applied to `Multisig.sol#L180` and `Multisig.sol#L196`.
   Update: Fixed.
5. Replace precomputed constants with the actual computation. Since these computations are constant, the compiler should generate the same code. Relevant locations:
   Update: Fixed.
   - `swap/DexKIP7.sol#21`
   - `swap/DexKIP7.sol#26`
   - `libraries/TransferHelper.sol#8`
   - `libraries/TransferHelper.sol#14`
   - `libraries/TransferHelper.sol#20`
6. Consider merging the two math libraries `utils/Math.sol` and `libraries/Math.sol`.
7. `DexPair.sol#105` performs a modulo `2**32` operation before casting the value to `uint32`. Since the cast will truncate the value anyway, the modulo operation is redundant and can be removed.
   Update: Fixed.
8. Always initialize values instead of relying on implicit zero-initialization. Relevant locations:
   Update: Fixed.
   - `libraries/DexLibrary.sol#65`
   - `swap/DexRouter.sol#434`
   - `swap/DexRouter.sol#695`
9. Remove usage of redundant ternary operator in `MultiSig.sol#264-267`.
   Update: Fixed.

Test Results

Test Suite Results

```
Multisig
  Should remove owner
     ✓ should submit transaction, proposal 0
     ✓ should revoke confirmation
     ✓ should not revoke if owner has not confirmed
     ✓ should confirm transaction
     ✓ should reject if owner confirmed
     ✓ should reject if not an owner
     ✓ should reject if owner was added after proposal 0 submission, adding proposal-1
     ✓ should reject if owner was replaced after proposal submission
     ✓ should reject execution if owner has not confirmed
     ✓ should execute transaction
     ✓ replaceOwner:fail, both owners exist
     ✓ send 0 ETH to the contract
     ✓ send ETH to the contract
  Should remove owner with changing requirments
     ✓ deploy:fail
     ✓ removeOwner:fail, owner5 does not exist
     ✓ removeOwner:req changed
     ✓ removeOwner:req changed
     ✓ replaceOwner:fail, same owners
     ✓ replaceOwner:fail, owner does not exist
     ✓ replaceOwner:fail, new owner is zero address
     ✓ replaceOwner:fail, destination is zero address
     ✓ addOwner:fail, same owner
     ✓ revokeConfirmation:fail, tx is already executed
     ✓ addOwner:fail, caller is not an owner

PlatformToken
   ✓ deploy:fail, multisig cannot be the zero address
   ✓ initial nonce is 0
   ✓ supportInterface
   ✓ chainID
   ✓ minting restriction
   ✓ should have correct name and symbol and decimal
   ✓ should only allow owner to mint token
   ✓ should supply token transfers properly
   ✓ should fail if you try to do bad transfers
  Compound test suite
    balanceOf
       ✓ grants to initial account
    numCheckpoints
       ✓ returns the number of checkpoints for a delegate
       ✓ does not add more than one checkpoint in a block
    getPastVotes
       ✓ reverts if block number >= current block
    getPastTotalSupply
       ✓ reverts if block number >= current block
       ✓ returns 0 if there are no checkpoints
```

```
        ✓ returns the latest block if >= last checkpoint block
        ✓ returns zero if < first checkpoint block
        ✓ generally returns the voting balance at the appropriate checkpoint

  Farming
    ✓ add:fail, should not stake reward token
    ✓ add:fail, pool was already added
    ✓ real case
    ✓ deposit/withdraw
    ✓ withdraw:fail, staked amount is less than amount to withdraw
    ✓ withdraw:fail, pool does not exsist
    ✓ deposit:fail, pool does not exsist
    ✓ update multiplier
    ✓ updateMultiplier:fail, pool does not exsist
    ✓ set:fail, pool does not exsist
    ✓ set:fail, totalAllocPoint cannot be 0
    ✓ pendingPtn:fail, pool does not exsist
    Farming Redeployable test
      ✓ deploy:fail, ptn cannot be the zero address
      ✓ deploy:fail, multisig cannot be the zero address
      ✓ should set correct state variables
      ✓ should allow emergency withdraw
      ✓ emergency withdraw: fail, pool does not exsist
      ✓ should give out PTNs only after farming time [ @skip-on-coverage ]
      ✓ should not distribute ptns if no one deposit [ @skip-on-coverage ]
      ✓ should distribute ptns properly for each staker [ @skip-on-coverage ]
      ✓ should give proper ptns allocation to each pool [ @skip-on-coverage ]
      ✓ should update allocation to each pool
      ✓ should update allocation to each pool with multiplier

  Staking
    Staking #1 - no limit pool
      ✓ Deploy pool with StakingFactory
      ✓ Deploy pool:fail, multisig cannot be the zero address
      ✓ Deploy pool:fail, staking token - no supply
      ✓ Deploy pool:fail, reward token - no supply
      ✓ Deploy pool:fail, tokens must be defferent
      ✓ Deploy pool:fail, invalid start block
      ✓ Deploy pool:fail, wrong parameters
      ✓ Deploy pool:fail, not an owner
      ✓ initialize:fail, already initilized
      ✓ initialize:fail, not factory
      ✓ change startBlock end endBlock:fail, start block < endBlock
      ✓ change startBlock end endBlock:fail, start block < block.timestamp
      ✓ change startBlock end endBlock
      ✓ Initial parameters are correct
      ✓ Users deposit
      ✓ Advance to startBlock
      ✓ change startBlock end endBlock:fail, pool has started
      ✓ Advance to startBlock + 2
      ✓ Advance to startBlock + 10
      ✓ Carol can withdraw
      ✓ Can collect rewards by calling deposit with amount = 0
      ✓ Can collect rewards by calling withdraw with amount = 0
      ✓ Carol cannot withdraw more than she had
      ✓ updatePoolLimitPerUser:failed, limit must be set for this pool
      ✓ updateRewardPerBlock:failed, pool has started
      ✓ Advance to end of IFO
      ✓ Deposit/withdraw after pool ended
    Staking #1 - owner can use recoverToken
      ✓ Owner can recover token
      ✓ Owner cannot recover token if balance is zero
      ✓ Owner cannot recover staked token
      ✓ Owner cannot recover reward token
      ✓ Owner cannot recover to the zero address
    Staking #2 - user limit pool
      ✓ Deploy user limit pool with StakingFactory
      ✓ Initial parameters for limited pool are correct
      ✓ Staking #2, updateRewardPerBlock, correct pool reward before pool has been started
      ✓ Staking #2, Users deposit:fail, amount is higher than userLimit
      ✓ Staking #2, Users deposit
      ✓ Staking #2, Advance to startBlock
      ✓ Staking #2, Advance to startBlock + 2
      ✓ Staking #2, Advance to startBlock + 10
      ✓ Staking #2, Carol can withdraw
      ✓ Staking #2, Can collect rewards by calling deposit with amount = 0
      ✓ Staking #2, Can collect rewards by calling withdraw with amount = 0
      ✓ Staking #2, Carol cannot withdraw more than she had
      ✓ Staking #2, updatePoolLimitPerUser:failed, new limit must be higher
      ✓ Staking #2, updatePoolLimitPerUser, deposit:fail - above limit
      ✓ Staking #2, deposit 50 PTN more
      ✓ Staking #2, disable pool limit for users
      ✓ Staking #2, deposit more PTN
      ✓ Staking #2, updateRewardPerBlock:fail, not an owner
      ✓ Staking #2, updateRewardPerBlock:fail, pool has started
    Staking #2 - pool emergency case
      ✓ Staking #2, stopReward:fail, not an owner
      ✓ Staking #2, stopReward
      ✓ Staking #2, emergencyRewardWithdraw:fail, not an owner
      ✓ Staking #2, emergencyRewardWithdraw:fail, recipien is zero address
      ✓ Staking #2, emergencyRewardWithdraw
      ✓ Staking #2, users perform emergency withdrawal
      ✓ Staking #2, users perform second emergency withdrawal

  DexFactory
    ✓ deployFactory: fail, feeToSetter cannot be the zero address
Init code hash: 0xf642c5ae86cfb4b6c9722ae01efc63d5e5b1c91b970fb76c62ebdaddc7aacd5e
    ✓ feeTo, feeToSetter, allPairsLength
    ✓ createPair
    ✓ createPair:reverse
    ✓ createPair:identical
    ✓ createPair:zero address
    ✓ createPair:gas [ @skip-on-coverage ]
    ✓ setFeeTo:fail, Unauthorized
    ✓ setFeeTo
    ✓ setFeeToSetter:fail
    ✓ setFeeToSetter
    ✓ math lib extra test

  DexKIP7
    ✓ name, symbol, decimals, totalSupply, balanceOf, DOMAIN_SEPARATOR, PERMIT_TYPEHASH
    ✓ approve
    ✓ supportInterface
    ✓ transfer
    ✓ safeTransfer:fail
    ✓ safeTransfer:to the contract
    ✓ transfer:fail
    ✓ transferFrom
    ✓ transferFrom:max
    ✓ transferFrom:fail
    ✓ safeTransferFrom
    ✓ safeTransferFrom:max
    ✓ safeTransferFrom:fail
    ✓ permit
    ✓ permit:fail expired
    ✓ permit:fail invalid signature

  DexPair
    ✓ mint
    ✓ getInputPrice:0
    ✓ getInputPrice:1
    ✓ getInputPrice:2
    ✓ getInputPrice:3
    ✓ getInputPrice:4
    ✓ getInputPrice:5
    ✓ getInputPrice:6
    ✓ optimistic:0
    ✓ optimistic:1
    ✓ optimistic:2
    ✓ optimistic:3
    ✓ swap:token0
    ✓ swap:token1
    ✓ swap:gas [ @skip-on-coverage ]
    ✓ burn
    ✓ price{0,1}CumulativeLast
    ✓ feeTo:off
    ✓ feeTo:on
    ✓ skim
    ✓ swap fail
    ✓ mint fail
    ✓ burn fail
    ✓ init fail

  DexRouter
    ✓ deploy:fail, wrong address parameters
    ✓ quote
    ✓ getAmountOut
    ✓ getAmountIn
    ✓ getAmountsOut
    ✓ getAmountsIn
    ✓ factory, WKLAY
    ✓ addLiquidity
    ✓ addLiquidityNonExistedPool
    ✓ addLiquidity:fail
    ✓ addLiquidityKLAY:fail
    ✓ addLiquidityKLAY
    ✓ removeLiquidity
    ✓ removeLiquidity:fail
    ✓ removeLiquidityKLAY
    ✓ removeLiquidityWithPermit
    ✓ removeLiquidityKLAYWithPermit
    swapExactTokensForKLAY
      ✓ happy path
    swapKLAYForExactTokens
      ✓ happy path
      ✓ happy path with extra KLAY
    swapExactKLAYForTokens
      ✓ happy path
      ✓ gas [ @skip-on-coverage ]
    swapTokensForExactKLAY
      ✓ happy path
    swapExactTokensForTokens
      ✓ happy path
      ✓ gas [ @skip-on-coverage ]
    swapTokensForExactTokens
      ✓ happy path

  DexRouter fee-on-transfer tokens
    ✓ removeLiquidityKLAYSupportingFeeOnTransferTokens
    ✓ removeLiquidityKLAYWithPermitSupportingFeeOnTransferTokens
    ✓ swapExactKLAYForTokensSupportingFeeOnTransferTokens
    ✓ swapExactTokensForKLAYSupportingFeeOnTransferTokens
    swapExactTokensForTokensSupportingFeeOnTransferTokens
      ✓ DTT -> WKLAY
      ✓ WKLAY -> DTT

  DexRouter fee-on-transfer tokens: reloaded
    swapExactTokensForTokensSupportingFeeOnTransferTokens
      ✓ DTT -> DTT2
```

Code Coverage

Code coverage is good, however we identified two blind spots that should be covered by the tests:

1. Staking: No tests check that the concept of user limits works correctly.
2. Farming: No tests check that the multiplier computation works correctly when `_from` is less than `bonusEndBlock`, while `_to` is greater than `bonusEndBlock` (`Farming.sol#132-135`).

Update: The blind spots are now covered as well.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
farming/	100	99.15	100	100	
Farming.sol	100	100	100	100	
StakingFactory.sol	100	100	100	100	
StakingFactoryPool.sol	100	98.39	100	100	
libraries/	96.23	78.13	92.86	96.15	
DexLibrary.sol	100	90	100	100	
Math.sol	100	100	100	100	
TransferHelper.sol	75	37.5	75	75	8,9
multisig/	100	100	100	100	
Multisig.sol	100	100	100	100	
swap/	99.36	94.59	100	99.39	
DexFactory.sol	100	100	100	100	
DexKIP7.sol	100	95.83	100	100	
DexPair.sol	99.03	86.96	100	99.01	137
DexRouter.sol	99.3	98.44	100	99.35	77
Errors.sol	100	100	100	100	
tokens/	100	100	100	100	
PlatformToken.sol	100	100	100	100	
All files	**99.39**	**95.35**	**99.24**	**99.41**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
0d390ec2fd993c4c09dbd7f30811942b66a06363393572fa065d7ff115cdb705 ./tokens/PlatformToken.sol

bdb04b3d6a504344490b4b22cb04fb9e3a85b71ba0611498cad8e5a4cc2c2c89 ./multisig/Multisig.sol

1bd086197989885dbf86a79699a8ee51ca7399f201236b625c0fda176538f19a ./farming/StakingFactoryPool.sol

85ed378b90a5e1f95e5225e2623784861eae6cfe42e45fb2fc59999285bd123c ./farming/StakingFactory.sol

87e5c4f0856b2fad9746934c9de75a8c9773b0ed245ccb7d5afcfb2d076280bd ./farming/Farming.sol

be67efd5d5ed157e8809a91a6b6435392ec72c5e7f7e0af38a13eb9cacc1f12b ./swap/DexFactory.sol

e18ccf27660ea5c96dc5c5ef99801801d2edfaae9369964900725b0341c474fa ./swap/DexPair.sol

b93dcc41a92c2189234bd09d709b52b7aafc77c32a0ab20a7458ab761a6ca7fb ./swap/DexRouter.sol

e8de0841723eb39f25af2cde4e949aac5e284b48fc0943a6ae4c0114ed5ae57e ./swap/DexKIP7.sol

419b348d33fe525bf9958b04a20a4baa64d002b40c478169a33c7973121baa36 ./swap/Errors.sol
```

Tests

```
c2bc112c7106b0e751fd6998658bb49ad00e9680550bf55b88a5550d30eb924c ./test/PlatformToken.spec.ts

ea9b4d830fe921a4a3f679555e6c20214063887d3a15416637eb4d50b87987f3 ./test/Multisig.spec.ts

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 ./test/index.ts

1c9d09cf739da3e15f8ccd8450fb53607aadcad50597d811c49940b7e7d420cd ./test/swap/DexFactory.spec.ts

c20f1d59035996c252923af41068c9d87579a9cba9a1a5d04d3b0f0273a12df5 ./test/swap/DexRouter.spec.ts

56dc18cebd99d7d0688735f4babd7702815341bda0e9e02230d42fa4af19b6a0 ./test/swap/DexKIP7.spec.ts

f0be412933249004d701c56f02d2c31f76be1606c2c1d51df7ec652110b81507 ./test/swap/DexPair.spec.ts

5edf4e4b9170c4c330158d8624c696a79cd9587d35bebbdeb961a00329c1ad62 ./test/shared/fixtures.ts

29d39666c71d6ca2969791613a94a60ffa1e00b43b3add2f236a59660e2a2b6b ./test/shared/utilities.ts

ee4ff3dbca9fffa928abad04d4805867495569ab6096265c22c8268e45a7e338 ./test/shared/interfaces.ts

969ad2026f7315915a41627884b2eacd5722927fe2c861ebca950ba621394ac6 ./test/farming-staking/Farming.spec.ts

9ae6204e76dced388052d8535e36566a69faea16aaae36bce5eafa121261d45c ./test/farming-staking/Staking.spec.ts
```

Changelog

- 2022-09-21 - Initial report
- 2022-10-07 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT