Quantstamp Security Assessment Certificate

September 16th 2022 — Quantstamp Verified

KLEX Token

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

Executive Summary

Type	Governance Token
Auditors	Guillermo Escobero, Security Auditor Ibrahim Abouzied, Auditing Engineer Zeeshan Meghji, Auditing Engineer
Timeline	2022-08-19 through 2022-08-30
Languages	Solidity
Methods	Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review
Specification	KLEX Token Docs
Documentation Quality	Low
Test Quality	Medium

Source Code

Repository	Commit
Klaytn-Defi/klex-token	880a2ab
Klaytn-Defi/klex-token (fixes)	3b13d77

Total Issues	**42** (23 Resolved)
High Risk Issues	**10** (8 Resolved)
Medium Risk Issues	**8** (4 Resolved)
Low Risk Issues	**13** (8 Resolved)
Informational Risk Issues	**7** (0 Resolved)
Undetermined Risk Issues	**4** (3 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

The Quantstamp team audited the KLEX token contracts for the Klaytn ecosystem, implemented by the KLEX team.

During the audit, we found several high severity issues. Some of them are critical and can lead to stolen funds or inaccurate accounting, with fatal consequences for the protocol. It is recommended to address all of them before the deployment phase of the project.

It is worth noting the elevated permissions that the administrator accounts have. These are detailed in `QSP-18 Privileged Roles and Ownership`. The privileged accounts should be protected by multi-signature mechanisms. Also, the users should be informed about the critical operations that the administrators can perform.

Although all the tests passed, we recommend implementing a more extensive test suite, as well as coverage metrics (to reach `100%` coverage). The high quantity of issues found and their causes show insufficient testing for the complexity of KLEX. Proper tests would have helped to avoid most of them and will help to implement safer functionalities.

As some KLEX contracts are a fork of KLAP contracts, some of the issues included in this report appeared on the audit of the KLAP protocol. These issues are still not addressed.

Update: In the fix review phase of the audit, the auditing team reviewed the fixes proposed by the KLEX team. Most of the issues were fixed or acknowledged. However, some of them were just partially fixed and their state remains “Unresolved”. We recommend the KLEX team fix all unresolved issues. Regarding testing, we strongly recommend improving the test suite and implementing coverage metrics to ensure all the code branches are properly tested. We would like to emphasize that the contracts are not currently production-ready due to unresolved issues and the lack of sufficient testing.

ID	Description	Severity	Status
QSP-1	Bribe Payouts Are Not Fully Given	High	Fixed
QSP-2	Rewards Are Lost on `veNFT` Burns	High	Fixed
QSP-3	Users Are Not Awarded Full Rewards	High	Fixed
QSP-4	Adding Tokens Breaks `onReward(...)`	High	Fixed
QSP-5	Broken Pause Mechanism	High	Acknowledged
QSP-6	Anyone Can Claim Bribes	High	Fixed
QSP-7	Merging Tokens Does Not Store the Combined `amount_stored` Value	High	Fixed
QSP-8	Reward Boosts After Lock Expiry	High	Unresolved
QSP-9	Blacklisted Users Can Transfer Tokens	High	Fixed
QSP-10	A User Can Be Given Excessive Rewards	High	Fixed
QSP-11	`emissionsSchedule` Can Become Unsorted	Medium	Fixed
QSP-12	`KlexToken` May Have Incorrect Tokenomics	Medium	Acknowledged
QSP-13	Unchecked Return Value	Medium	Fixed
QSP-14	Missing Input Validation	Medium	Unresolved
QSP-15	Total Supply Can Become Incorrect	Medium	Fixed
QSP-16	Forfeiting User Rewards	Medium	Fixed
QSP-17	Pool Rewards Allocated Incorrectly	Medium	Acknowledged
QSP-18	Privileged Roles and Ownership	Medium	Acknowledged
QSP-19	Ownership Can Be Renounced	Low	Fixed
QSP-20	Prematurely Terminated `for` Loops	Low	Acknowledged
QSP-21	Integer Overflow / Underflow	Low	Unresolved
QSP-22	ERC-20 Decimals Are Defined as `uint256`	Low	Fixed
QSP-23	Not Fully Compliant with ERC-721	Low	Acknowledged
QSP-24	Transaction Ordering Dependence for `initialize()` Functions	Low	Unresolved
QSP-25	Emissions May Not Be Updated Properly	Low	Fixed
QSP-26	Inconsistent Allowance Behavior	Low	Fixed
QSP-27	`claimKLEX(...)` and `claimKLAY(...)` Can Be Called Anytime	Low	Fixed
QSP-28	`Lockdrop` and `Vamp` Rewards Can Be Lost	Low	Fixed
QSP-29	Checks-Effects-Interactions Pattern Violation	Low	Fixed
QSP-30	Rewards Stage Does Not Correctly Transition	Low	Fixed
QSP-31	Incorrect Accounting for `withdrawableBalance(...)`	Low	Acknowledged
QSP-32	Misleading Naming	Informational	Acknowledged
QSP-33	Application Monitoring Can Be Improved by Emitting More Events	Informational	Acknowledged
QSP-34	Using `transfer(...)` for KLAY May Fail in the Future	Informational	Acknowledged
QSP-35	Allowance Double-Spend Exploit	Informational	Acknowledged
QSP-36	Implicit Use of Interfaces	Informational	Acknowledged
QSP-37	Duplicated Libraries	Informational	Acknowledged
QSP-38	Upgradability	Informational	Acknowledged
QSP-39	Allocation Points Can Be Updated After Voting Has Started	Undetermined	Fixed
QSP-40	Potential Miscalculation of `totalAllocPoints`	Undetermined	Acknowledged
QSP-41	Dead Code	Undetermined	Mitigated
QSP-42	Address Can Be on Blacklist and Whitelist Simultaneously	Undetermined	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:
If the final commit hash provided by the client contains features that are not in scope of the audit or a re-audit, those features are excluded from the 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 Bribe Payouts Are Not Fully Given

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/MasterChef.sol`

Description: In `claimBribe(...)`, a user can redeem a portion of the unclaimed bribes based on their voting history. Here is an excerpt of the function:

```
uint256 amount = _vote.mul(bribes[_epoch][_pool]).div(allocPoints[_epoch][_pool]);
claimedBribe[_epoch][_tokenId][_pool] = true;
if (amount > unclaimedBribes[_epoch][_pool]) {
  unclaimedBribes[_epoch][_pool] = 0;
  klexToken.transferFrom(address(this), msg.sender, unclaimedBribes[_epoch][_pool]); // <- unclaimedBribes == zero. Nothing is transferred.
  emit ClaimBribe(_pool, msg.sender, unclaimedBribes[_epoch][_pool], _epoch);
}
else {
  unclaimedBribes[_epoch][_pool] = unclaimedBribes[_epoch][_pool].sub(amount);
  klexToken.transferFrom(address(this), msg.sender, amount);
  emit ClaimBribe(_pool, msg.sender, amount, _epoch);
```

```
}
```

If the amount the user is entitled to is more than the number of unclaimed bribes, there is no payout because the unclaimed bribes are removed before the transfer call.
Even if this were to be fixed and the user is given the remaining unclaimed bribes, it means that the bribe payout may be lower than the user expected. It is possible that they would not have accepted the bribe and would have voted differently with this information.

Recommendation: Store the `unclaimedBribes[_epoch][_pool]` in an intermediate variable to use for transferring tokens. Make it absolutely clear to users what their bribe payout will be so that they can vote with full information.

Update: Fixed in commit `bcdcf0f`. Reply from the KLEX team:

> *Since there is a nonreentrant modifier, we can just change the order of the two lines. The if statement will only occur if retrieve excess bribes is called, and it will be clear on the UI that users must claim their bribe within a week for them to receive it.*

QSP-2 Rewards Are Lost on `veNFT` Burns

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/ve.sol`

Description: `merge(...)` combines two `veNFT`s into one. The function contains the following calls:

```
...
_burn(msg.sender, _from); // <- Sets rewardPerTokenId[_tokenId] to 0
...
claimVeStakingReward(_from); // <- Awards based on rewardPerTokenId[_tokenId]
claimVeStakingReward(_to);
_deposit_for(_to, value0, end, _locked1, DepositType.MERGE_TYPE);
...
```

Because `_burn(...)` wipes the `rewardPerTokenId`, the rewards are removed from the token before they are claimed.
Additionally, in `withdraw(...)` a token may be burnt while it still has unclaimed rewards.

Recommendation: Always claim token rewards before burning a token.

Update: Fixed in commit `afb4b56` and `3b13d7`.

QSP-3 Users Are Not Awarded Full Rewards

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/MultiRewarder.sol`

Description: In the event that the contract does not have sufficient tokens, `onReward(...)` attempts to pay users with whatever balance it has:

```
...
pendingBal = (userInfo.amount.mul(poolInfo.accTokenPerShare[i]).div(ACC_TOKEN_PRECISION)).sub(
  userInfo.rewardDebt[i]
);
rewardBal = poolInf.rewardToken[i].balanceOf(address(this));
if (pendingBal > rewardBal) { // <- Current balance is not sufficient
  poolInfo.rewardToken[i].safeTransfer(_user, rewardBal); // <- Pay whatever is currently possible
  userInfo.rewardDebt[i] = _lpAmount.add(pendingBal.sub(rewardBal)).mul(poolInfo.accTokenPerShare[i]).div(ACC_TOKEN_PRECISION);
  // ^ The amount unawarded (pendingBal - rewardBal) is added rather than subtracted
} else {
  poolInfo.rewardToken[i].safeTransfer(_user, pendingBal);
  userInfo.rewardDebt[i] = _lpAmount.mul(poolInfo.accTokenPerShare[i]).div(ACC_TOKEN_PRECISION);
}
...
```

However, the missing balance is added to the reward debt rather than subtracted. Not only are users unable to claim these rewards later, but they will also have an excessive reward debt keeping them from claiming future rewards as well.

Recommendation: Subtract the missing balance rather than adding it.

Update: Fixed in commit `b2702ef`. The KLEX team has fixed the issue by correcting the reward calculation so that the user can claim pending rewards in the future if the contract has an insufficient balance to pay them the full amount.

QSP-4 Adding Tokens Breaks `onReward(...)`

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/MultiRewarder.sol`

Description: `onReward(...)` is called by `MasterChef` on deposits, withdrawals, and claims of LP tokens. The function is meant to track and deliver token rewards for staking. A pool may have several reward tokens.
When `onReward(...)` is first called for a user, their `userInfo.rewardDebt[]` mapping is initialized to be of length `poolInfo.numRewardTokens`. But if a new reward token is later added to the pool by calling `addRewardTokenToPool(...)`, then `userInfo.rewardDebt[]` will be shorter than `poolInfo.numRewardTokens`, leading to an invalid index error when the reward debts of that token are updated.

Recommendation: Before checking the values of `userInfo.amount`, check if `userInfo.rewardDebt.length == poolInfo.numRewardTokens`. If the arrays are not an equal length, initialize any missing tokens.

Update: Fixed in commit `df62d0c`. The KLEX team has fixed the issue by initializing the user's `rewardDebt` array such that its length will equal the pool's number of reward tokens.

QSP-5 Broken Pause Mechanism

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `contracts/MultiFeeDistribution.sol`, `contracts/KlexToken.sol`

Description:

1. `MultiFeeDistribution` uses the `mintingPaused` boolean to guard several functions. However, it is not present in the `mint(...)` function, making it possible to mint tokens when token minting should be paused. Also, it seems that the `mintingPaused` is guarding *exiting* functions instead of minting. This behavior can be confusing due to the variable name.
2. The `KlexToken` contract has a `transferPaused` flag which determines whether token transfers are possible or not. However, even when the `transferPaused` flag has been set to `true`, it is still possible for any user to burn their tokens through the `burn(...)` and `burnFrom(...)` functions. It is also possible for any address with the minter role to call the `mint(...)` function to mint tokens to any address.

Recommendation:

1. Add a guard to require that minting is not paused in `mint(...)`, and document the use cases of `mintingPaused`.
2. Validation should be added to the `burn(...)`, `burnFrom(...)`, and `mint(...)` functions such that only they are only callable by whitelisted addresses.

Update: Reply from the KLEX team:

> *The mintingPaused boolean on MultiFeeDistribution is specifically to pause users from exiting and receiving liquid KLEX tokens. Other functionalities don't need to be paused based on this variable. This is by design to have users start vesting but stay locked in during pre-mining.*

The KLEX team has indicated that the pause mechanism is intended to prevent users from exiting during the pre-mining phase. This behavior is expected by design.

QSP-6 Anyone Can Claim Bribes

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/MasterChef.sol`

Description: The `claimBribe` function in the `MasterChef` contract allows a user to claim a particular `Ve` token's portion of the bribe rewards for a specific pool and epoch. However, the function currently allows a user to claim the `Ve` token's rewards even if the user does not own the token.
The `claimBribe` function accepts a `_tokenId` parameter, but does not at any point validate that the caller owns the `Ve` token corresponding to the `_tokenId`. Furthermore, the bribe rewards are sent to the caller of the `claimBribe` function as seen on `L301:klexToken.transfer(msg.sender, unclaimedBribes[_epoch][_pool]);` and `L306:klexToken.transfer(msg.sender, amount);`. Thus any caller of the `claimBribe` function can claim the bribe rewards for any `_tokenId` provided.

Recommendation: Verify that the caller is the owner of `_tokenId`:

```
require(IVe(_ve).ownerOf(_tokenId) == msg.sender, "You do not own this token id");
```

Update: Fixed in commit `eb3aaa5`.

QSP-7 Merging Tokens Does Not Store the Combined `amount_stored` Value

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/ve.sol`

Description: When `merge(...)` calls `_deposit_for(...)` with `DepositType.MERGE_TYPE`, the `amount_stored` value is not updated to the reflect the new value of the `to` token. The `amount_stored` value is also left untouched for the `for` token when it should be set to zero. When a user calls `withdraw(...)`, the funds that were in the `for` token will be inaccessible.

Recommendation: When merging tokens, move the `amount_stored` in the `from` token into the `to` token.

Update: Fixed in commit `0c409c0`. The Klex team has fixed the issue by incrementing the `amount_stored` for the target token of the merge.

QSP-8 Reward Boosts After Lock Expiry

Severity: *High Risk*

Status: Unresolved

File(s) affected: `contracts/MasterChef.sol`

Description: By locking KLEX tokens into the `Ve` contract, users gain reward boosts based on the amount of tokens they lock and the lock duration. It follows that users should not continue to have a reward boost after the expiry of all their locks in the `Ve` contract. However, it is possible to continue to possess reward boosts even after lock expiry.
A user's reward boost within the `MasterChef` contract is determined by the user's factor. The user's factor is updated whenever a user modifies or creates new locks in the `Ve` contract. The user's factor is also updated when a user calls the `deposit(...)` or `withdraw(...)` functions within the `MasterChef` contract. However, if the user does not claim for the duration of their lock and does not modify their lock in any way through functions on the `Ve` contract, the user's factor remains unchanged. Therefore, when a claim happens even after a lock expiry, the user will gain the reward boost from the time of the lock creation to the time of the claim rather than to the time of the lock expiry.

Exploit Scenario:

1. A user locks KLEX tokens for `2` years using the `create_lock(...)` function on the `Ve` contract.
2. After `3` years have passed, the user calls `claim(...)` on the `MasterChef` contract.
3. The user has received the maximum boost for `3` years, even though they had only locked for `2` years.

Recommendation: We recommend implementing either of the following solutions:

1. Modify the staking protocol design so that the user's factor is always updated after a lock expiry.
2. Modify the `claim(...)` function so that it only applies reward boosts until the user's lock expires.
3. Curve Finance liquidity gauges implement an external function `kick(address user)` that can be called by anyone to update (checkpoint) the user's boost if the lock has expired.

QSP-9 Blacklisted Users Can Transfer Tokens

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/KlexToken.sol`

Description: The `KlexToken` contract contains a `blacklist` of addresses that should not be allowed to transfer tokens. However, by simply approving another address to transfer the blacklisted address' tokens, they can be transferred from the blacklisted address.
The check to prevent transfers from blacklisted addresses can be seen on the first line of the `_transfer(...)` function definition below from `L58` to `L68`.

```
function _transfer(
  address _from,
  address _to,
  uint256 _value
) internal {
  require((!transferPaused || whitelist[msg.sender]) && !blacklist[msg.sender], "Transfer paused");
  require(balanceOf[_from] >= _value, "Insufficient balance");
  balanceOf[_from] = balanceOf[_from].sub(_value);
  balanceOf[_to] = balanceOf[_to].add(_value);
  emit Transfer(_from, _to, _value);
}
```

Notice that the check only verifies whether the caller is on the `blacklist`. The check does not verify whether the tokens are being transferred from an address on the `blacklist`. Furthermore, we also note that an address on the `blacklist` can still freely burn tokens using the `burn(...)` and `burnFrom(...)` functions. It is also possible for a blacklisted address to mint tokens by calling the `mint(...)` function as long as the address possesses the minter role as defined by the `minters` mapping on `L19:mapping(address => bool) public minters;`.

Exploit Scenario:

1. A malicious user's address called `addressX` is placed on the blacklist by calling `setBlacklist(addressX,true)`. `addressX` currently possesses `10` KLEX tokens.
2. The malicious user has another address called `addressY`. Using `addressX`, the malicious user calls `approve(addressY, 10)`.
3. Using `addressY`, the malicious user then calls `transferFrom(addressX, addressY, 10)` to transfer all `10` tokens to `addressY`. The malicious user can now freely transfer tokens from `addressY` without any restrictions.

Recommendation:

1. Validate that tokens cannot be transferred from an address on the `blacklist`.
2. Determine whether it is desirable for blacklisted addresses to mint or burn tokens. If not, then steps should be taken to prevent minting and burning tokens from the blacklisted address through the `mint(...)`, `burn(...)`, and `burnFrom(...)` functions.

Update: Fixed in commit `34b2754`. The KLEX team has fixed the issue by adding validation to ensure that tokens cannot be transferred from a blacklisted account. They have also added validation to ensure that blacklisted accounts cannot be minted to and cannot burn their tokens.

QSP-10 A User Can Be Given Excessive Rewards

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/MultiRewarder.sol`

Description: `onReward(...)` is called by `MasterChef` on deposits, withdrawals, and claims of LP tokens. The function is meant to track and deliver token rewards for staking. A pool may have several reward tokens.

`onReward(...)` does not properly update the `rewardDebt` for a user who has their balance increase from zero for a second time. This results in them being rewarded based on outdated `rewardDebt` values.

Exploit Scenario:

1. A user deposits `10` tokens when `accTokenPerShare` is `1`. `userInfo.amount == 0 && userInfo.rewardDebt.length == 0`, so their `rewardDebt` array is initialized, and their `rewardDebt` is set to `10` (`10` tokens * `1` `accTokenPerShare`).
2. The `accTokenPerShare` increases to `2`, and the user withdraws all `10` of their tokens. `userInfo.amount > 0`, so they are awarded `20` reward tokens (`10` tokens * `2` `accTokenPerShare` - `10` `rewardDebt`). Their `rewardDebt` is set to `20`.
3. A large amount of time passes with the user taking no action. During this time, the `accTokenPerShare` rises to `1` million.
4. A user deposits `1` token. When `onReward(...)` begins, their `userInfo.rewardDebt.length` is set to `1` since the array has already been initialized, and their `userInfo.amount` is `0`, so neither the `if (userInfo.amount == 0 && userInfo.rewardDebt.length == 0)` nor the `else if (userInfo.amount > 0)`, and **the users `rewardDebt` is still equal to 20.**
5. The user immediately withdraws their token. `userInfo.amount > 0`, so they are awarded **999,980 reward tokens** (`1` tokens * `1` million `accTokenPerShare` - `20` `rewardDebt`). Their `rewardDebt` is set to `0`.

Recommendation: The recommendation for this issue is the same as the one for **QSP-4: Adding tokens breaks onReward(...)**. Before checking the values of `userInf.amount`, check if `userInfo.rewardDebt.length == poolInfo.numRewardTokens`. If the arrays are not an equal length, initialize any missing tokens.

Update: Fixed in commits `b3bba0a` and `df62d0c`. The KLEX team has fixed the issue by ensuring that the user's `rewardDebt` array is always reset if the user's previously deposited amount was 0.

QSP-11 `emissionsSchedule` Can Become Unsorted

Severity: *Medium Risk*

Status: Fixed

Description: The `emissionSchedule` array is described as being sorted in reverse chronological order. However, the `setEmissionSchedule(...)` function appends emission points onto the existing `emissionSchedule` array. There is no guarantee that all of the emissions will be earlier than any existing emission in the array. This validation is also missing in `initialize(...)` when the array is first set.

Recommendation: Validate that the soonest emission point on the existing `emissionSchedule` is later than the point being added (monotonically decreasing).

Update: Fixed in commit `2feb27a`.

QSP-12 `KlexToken` May Have Incorrect Tokenomics

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `contracts/KlexToken.sol`

Description: With regards to the `KlexToken` tokenomics, the documentation mentions:

> *KLEX has a maximum total supply of 1,000,000,000 (1 billion) tokens.*

However, `KlexToken.initialize()` only requires that the `maxTotalSupply` be greater than 100,000:

```
function initialize(uint256 _maxTotalSupply) public initializer {
    __Ownable_init();
    require(_maxTotalSupply > 100000000000000000000000, "Max must be over 100000");
    maxTotalSupply = _maxTotalSupply;
    emit Transfer(address(0), msg.sender, 0);
}
```

Additionally, if tokens are burned, they can still be reminted. This means they are not deflationary as the documentation might suggest.

Recommendation: Require that `maxTotalSupply` be equal to 1 billion. Track the maximum supply after burns and guard against the total supply increasing past it when minting.

Update: Reply from the KLEX team:

> *This is by design - the community could vote on a change to the tokenomics that mints tokens for rewards/incentives/partnerships that we may not be able to predict right now. The current setup increases flexibility/optionality.*

We recommend that this should be clearly reflected in the documentation.

QSP-13 Unchecked Return Value

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/Lockdrop.sol`, `contracts/Vamp.sol`, `contracts/MasterChef.sol`, `contracts/ve.sol`

Related Issue(s): SWC-104

Description: Most functions will return a `true` or `false` value upon success. Some functions, like `send(...)`, are more crucial to check than others. It is important to ensure that every necessary function is checked. If not, it can lead to wrong accounting for failed transactions. For example, `ERC-20` is not required to revert in wrong transfers.

1. `Lockdrop.sol`: `L57`, `L69`, `L87`
2. `Vamp.sol`: `L63`, `L75`, `L93`
3. `MasterChef.sol`: `L281`, `L301`, `L306`, `L318`
4. `ve.sol`: `L976` (in this case the return value is checked, but we still recommend using a safe wrapper).

Recommendation: Check the return value of the external calls listed. We recommend using the OpenZepppelin library as it includes additional validation: SafeERC20.

Update: Fixed in commit `68001c1`.

QSP-14 Missing Input Validation

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `contracts/KlayFut.sol`, `contracts/KlexToken.sol`, `contracts/Lockdrop.sol`, `contracts/MasterChef.sol`, `contracts/MultiFeeDistribution.sol`, `contracts/MultiRewarder.sol`, `contracts/Vamp.sol`, `contracts/ve.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs to avoid human error, even if they only come from trusted addresses.
There are setter functions that are not validating the input addresses to confirm that contracts are not being set to the zero address.
Here is a non-exhaustive list of functions that need to be updated:

1. `KlayFut`
   - `.mint`
     - Validate that `_to` is not the zero address.
2. `KlexToken.sol`
   - `.setWhitelist`
     - Validate that the addresses are not the zero address.
   - `.setBlacklist`
     - Validate that the addresses are not the zero address.
   - `.setMinter`
     - Validate that the addresses are not the zero address.
   - `.approve`
     - Approvals to zero address are allowed now.
   - `._transfer`

-
    - Validate that `_from` and `_to` are not the zero address to avoid unapproved token burning.
- `_mint`
    - Minting to zero address is allowed now.

3. `Lockdrop`
    - `initialize`
        - Validate that `_klex` is not the zero address.
        - Validate that `_wKLAY` is not the zero address.
        - Validate that `_depositEndTime` is in the future.
        - Validate that `_lockEndTime` is after `_depositEndTime`.
    - `setDepositEndTime`
        - Validate that `_depositEndTime` is in the future.
        - Validate that `_depositEndTime` is before `lockEndTime`.
    - `setLockEndTime`
        - Validate that `_lockEndTime` is in the future.
        - Validate that `depositEndTime` is before `_lockEndTime`.

4. `MasterChef`
    - `initialize`
        - Validate that `_maxMintable` is less than the maximum supply of KLEX.
    - `claimBribe`
        - Validate that `_pool` is an existing pool.
    - `retrieveExcessBribes`
        - Validate that `_pool` is an existing pool.
    - `deposit`
        - Validate that `_amount` is greater than `0`.
    - `withdraw`
        - Validate that `_amount` is greater than `0`.
    - `claim`
        - Validate that the length of `_tokens` is greater than `0`.
    - `vote`
        - Validate that the lengths of `_pools` and `_weights` are equal.

5. `MultiFeeDistribution`
    - `initialize`
        - Validate that `rewardPercentage` is less than `100%`.
    - `exit`
        - Validate that `amount` is greater than `0`.
    - `exitForVe`
        - Validate that `_amount` is greater than `0`.

6. `MultiRewarder`
    - `addPool`
        - Validate that `_pool` is an existing pool in `MasterChef`.
    - `addRewardTokenToPool`
        - Validate that `_tokenPerBlock` is greater than `0`.
    - `setRewardRate`
        - Validate that `_pool` is an existing pool in `MultiRewarder`.
    - `updatePool`
        - Validate that `_pool` is an existing pool in `MultiRewarder`.

7. `Vamp`
    - `initialize`
        - Validate that `_klex` is not the zero address.
        - Validate that `_wKLAY` is not the zero address.
        - Validate that `_depositEndTime` is in the future.
        - Validate that `_lockEndTime` is after `_depositEndTime`.
    - `withdraw`
        - Validate that `klaySwapToKlexToken[_token]` is not the zero address.
    - `setDepositEndTime`
        - Validate that `_depositEndTime` is in the future.
        - Validate that `_depositEndTime` is before `lockEndTime`.

        - `setLockEndTime`
            - Validate that `_lockEndTime` is in the future.
            - Validate that `depositEndTime` is before `_lockEndTime`.

8. `Ve`
        - `initialize`
            - Validate that `_multi_fee` is not the zero address.
        - `setEverything`
            - Validate that `new_lock_time` does not exceed the maximum lock time.
            - Validate that `ve_multiplier` is less than some reasonable maximum value.
        - `setVoter`
            - Validate that `_voter` is not the zero address.
        - `setVoting`
            - Validate that the token with `tokenId` exists.

Recommendation: We recommend adding the relevant checks.

Update: Partially fixed in commit `75091ee`. While the KLEX team has implemented many of the suggested validation checks, the following checks have not been implemented:

- `KlexToken`
    - `_transfer`
        - Does not validate that `_from` is not the zero address.
- `MasterChef`
    - `initialize`
        - Does not validate that `maxMintable` is less than the maximum supply of KLEX.
    - `deposit`
        - Does not validate that `_amount` is greater than `0`.
    - `withdraw`
        - Does not validate that `_amount` is greater than `0`.
- `MultiRewarder`
    - `addPool`
        - Does not validate that `_pool` is an existing pool in `MasterChef`.
    - `addRewardTokenToPool`
        - Does not validate that `_tokenPerBlock` is greater than `0`.
- `Ve`
    - `setEverything`
        - Does not validate that `new_lock_time` does not exceed the maximum lock time.
        - Does not validate that `ve_multiplier` is less than some reasonable maximum value.
    - `setVoter`
        - Does not validate that `_voter` is not the zero address.
    - `setVoting`
        - Does not validate that the token with `_tokenId` exists.

QSP-15 Total Supply Can Become Incorrect

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/KlexToken.sol`

Description: It is currently possible to burn KLEX tokens through the `transfer(...)` and `transferFrom(...)` functions on the `KlexToken` contract by setting the `_to` parameter to the zero address. However, the `totalSupply` is not reduced as it should be when this happens. The `totalSupply` of the KLEX token is critical as it determines how much of the protocol's treasury can be redeemed, and so the `totalSupply` should always remain correct.

Recommendation: We recommend either one of the following two actions:

1. Prevent the burning of tokens from the `transfer(...)` and `transferFrom(...)` functions by validating that the `_to` parameter is not the zero address.
2. Correctly reduce the `totalSupply` in the `transfer(...)` and `transferFrom(...)` functions when the `_to` parameter is set to the zero address.

Update: Fixed in commit `75091ee`.

QSP-16 Forfeiting User Rewards

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/MasterChef.sol`

Description: The `onReward(...)` function of the `MultiRewarder` contract is called by the `MasterChef` contract to provide users with additional reward tokens. However, if the `MultiRewarder` does not have enough tokens to pay the user their allocated reward, the contract incorrectly sets the user's `rewardDebt` too high. This will result in a user not only forfeiting part of their current rewards, but also some portion of their future rewards.
The offending code can be seen within the first `if` block of the following code from `L214` to `L221`:

```
rewardBal = p.rewardToken[i].balanceOf(address(this));
if (pendingBal > rewardBal) {
  p.rewardToken[i].safeTransfer(_user, rewardBal);
  u.rewardDebt[i] = _lpAmount.add(pendingBal.sub(rewardBal)).mul(p.accTokenPerShare[i]).div(ACC_TOKEN_PRECISION);
} else {
  p.rewardToken[i].safeTransfer(_user, pendingBal);
  u.rewardDebt[i] = _lpAmount.mul(p.accTokenPerShare[i]).div(ACC_TOKEN_PRECISION);
}
```

Instead of setting a lower `rewardDebt` when the full amount has not been claimed, the code above sets a higher `rewardDebt` by adding `pendingBal.sub(rewardBal)`. This means that the function effectively makes the user's reward negative.

Recommendation: If the `MultiRewarder` contract does not have a sufficient token balance to pay the user's full rewards, the user's `rewardDebt` should be set such that they can receive their unclaimed rewards when the `MultiRewarder` contract's token balance has been replenished.

Update: This issue is related to QSP-3. Fixed in commit `b2702ef`.

QSP-17 Pool Rewards Allocated Incorrectly

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `contracts/MasterChef.sol`

Description: The `internal _updatePool(...)` function is referenced by many other functions within the `MasterChef` contract to update the rewards allocated to a pool. However, the function will incorrectly update the pool's allocated rewards if it has not been called at the end of the previous epoch. This could result in a pool being allocated too many or too few rewards. We can see that only the allocation points of the current epoch are loaded into memory on `L438:uint256 _allocPoints = allocPoints[epoch][_token];` of the `MasterChef` contract. Furthermore, `L445` to `L446` demonstrate that this value is being used to calculate the additional rewards allocated to the pool:

```
uint256 duration = block.timestamp.sub(pool.lastRewardTime);
uint256 reward = duration.mul(rewardsPerSecond).mul(_allocPoints).div(_totalAllocPoint);
```

If `pool.lastRewardTime` is before the start of the current epoch, then `duration` will also include time from the previous epoch. Previous epochs can have a different allocation point value, and thus the calculation for rewards is incorrect.

Recommendation: If `pool.lastRewardTime` is before the start of the current epoch, then the allocation point values to the corresponding previous epochs should be used for their portion of the additional pool reward.

Update: Reply from the KLEX team:

> *We have a cronjob running that calls massUpdatePools before the end of each epoch since the epochs always have a constant time between them.*

QSP-18 Privileged Roles and Ownership

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `contracts/KlayFut.sol`, `contracts/KlexToken.sol`, `contracts/Lockdrop.sol`, `contracts/MasterChef.sol`, `contracts/MultiFeeDistribution.sol`, `contracts/MultiRewarder.sol`, `contracts/Treasury.sol`, `contracts/Vamp.sol`, `contracts/ve.sol`

Description: Smart contracts will often have `owner` variables or roles to designate people with special privileges to make modifications to the smart contract.
Several contracts have privileged roles which possess many elevated permissions. Of particular note are the ones below:

- `KlayFut`
  - `mint`: Mint an arbitrary amount of `KLAY-FUT` tokens to any address.
  - `sweepAll`: Drain the contract of all KLAY to any address.
  - `burnFrom`: Burn an arbitrary amount of `KLAY-FUT` tokens belonging to any address.
- `KlexToken`
  - `setWhitelist`: Add or remove any user from the `whitelist`. If a user is on the whitelist, they can transfer tokens even when token transfers have been paused.
  - `setBlacklist`: Add or remove any user from the `blacklist`. If a user is on the blacklist, they cannot transfer KLEX tokens.
  - `setTransferPaused`: Pause or unpause token transfers. Pausing token transfers will prevent all users besides whitelisted ones from transferring their tokens.
  - `setMinter`: Add or remove the minter role from any address. Any address with the minter role can mint arbitrary amounts of KLEX tokens to any address.
- `Lockdrop`
  - `setDepositEndTime`: Set the timestamp at which deposits will no longer be accepted.
  - `setLockEndTime`: Set the timestamp at which withdrawals will become possible.
  - `setPoolCaps`: Set the maximum amount of deposits for a particular token.
  - `setKlexRewards`: Set the total amount of KLEX rewards for a particular token.
  - `setKlayRewards`: Set the total amount of KLAY rewards for a particular token.
  - `rescue`: Drain the contract of any ERC-20 tokens and send them to an arbitrary address.
- `MasterChef`
  - `beginVoting`: Allow the users to begin voting.
  - `setRewardMinter`: Set the contract which mints KLEX rewards the user.
  - `setRewarder`: Set the contract which mints additional reward tokens.
  - `setCap`: Set the maximum amount of deposits for a particular token.
  - `setRewardsPerSecond`: Reset the rewards rate for the contract at any time. This essentially subverts the emission schedule mechanism.
  - `setEmissionSchedule`: Append entries into the emission schedule, determining the reward rate and various timestamps.

-
  - `updateEmissionRepartition`: Set what portion of the rewards is dedicated to `Ve` stakers.
  - `addPool`: Add a new pool for any token to the contract with arbitrary allocation points
  - `retrieveExcessBribes`: Transfer any bribes which have not been claimed by users for two weeks to the treasury. This effectively deprives users of their bribe rewards if they have not claimed them on time.
  - `batchUpdateAllocPoint`: Update the allocation points to an arbitrary number at any time.
  - `setClaimReceiver`: Set the reward receiver for any user at any time. This could allow the redirection of a user's rewards to any other address specified by the contract owner.
  - `rescue`: Drain the contract of any ERC-20 tokens at any time.

- `MultiFeeDistribution`
  - `setMinters`: Grant any address the `minter` role which would allow that address to mint token rewards arbitrarily.
  - `revokeMinter`: Revoke the `minter` role from any address.
  - `setIncentivesController`: Set the `incentivesController` to any address.
  - `replenishVeRewards`: Increase the amount of KLEX rewards that can be minted by the contract.
  - `setMintingStatus`: Determine whether a user can call `exit` and `exitForVe`.
  - `recoverERC20`: Drain the contract of any ERC-20 tokens at any time.

- `MultiRewarder`
  - `addPool`: Add a new pool for any token to the contract.
  - `addRewardTokenToPool`: Add any reward token to the contract.
  - `removePool`: Remove any pool from the contract.
  - `setRewardRate`: Reset the reward rate to any value for any pool.
  - `reclaimTokens`: Drain the contract of any ERC-20 tokens and KLAY.

- `Treasury`
  - `initializeBurn`: Determine when users may redeem KLEX tokens to receive a portion of the treasury.
  - `addReward`: Add a new reward token to be distributed to stakers.
  - `rescue`: Drain the contract of any ERC-20 tokens.
  - `sweepAll`: Drain the contract of any KLAY.

- `Vamp`
  - `setDepositEndTime`: Set the timestamp at which deposits will no longer be accepted.
  - `setLockEndTime`: Set the timestamp at which withdrawals will become possible.
  - `setPoolCaps`: Set the maximum amount of deposits for a particular token.
  - `setKlexRewards`: Set the total amount of KLEX rewards for a particular token.
  - `setKlayRewards`: Set the total amount of KLAY rewards for a particular token.
  - `setKLEXPoolTokenAmounts`: Set the total withdrawable KLEX pool tokens for a particular token. If the wrong amount is set, users will not be able to withdraw their fair share of KLEX pool tokens.
  - `setKlaySwapToKLEXMapping`: Set the KLEX pool token which corresponds to a specific KLAYswap token.
  - `rescue`: Drain the contract of any ERC-20 tokens and send them to an arbitrary address.

- `Ve`
  - `setEverything`: Set references to many critical contracts and variable values at any time including:
    - The `MasterChef` contract
    - The `MultiFeeDistribution` contract
    - The minimum lock time
    - The reward multiplier
  - `setVoter`: Set the contract through which users vote at any time.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner. Highly privileged roles should be assigned to timelock contracts to give users an opportunity to opt-out of the protocol before the changes take effect. Privileged roles should also be secured by large multi-signature wallets or by a decentralized governance process.

1. For `Treasury.rescue(...)`, consider putting a guard to only allow transferring tokens that are not in `rewardTokens`.
2. Remove `KlayFut.burnFrom(...)` if it is not necessary.

Update: Reply from the KLEX team:

> *Privileged roles are moving to 4 of 6 multisig once launch processes are complete. Until then, we need the flexibility. In the future, we should be moving to fully on-chain governance with time locks. This has been made clear in the docs: https://docs.klex.finance/privileged-roles*

QSP-19 Ownership Can Be Renounced

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/KlexToken.sol`, `contracts/KlayFut.sol`

Description: Many of the contracts have an owner role which is derived from OpenZeppelin's `Ownable.sol` or `OwnableUpgradeable.sol`. In both cases, it is possible to renounce ownership by calling the `renounceOwnership(...)` function.
If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Override the `renounceOwnership(...)` function in each of the affected contracts so that the function always reverts.

Update: Fixed in commit `28da215`.

QSP-20 Prematurely Terminated `for` Loops

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/Treasury.sol`

Description: In `Treasury.addReward(...)`, the loop uses a counter `i` of type `uint8`. This means that if `i` ever reaches `255`, it will overflow to `0` on the next iteration. This will lead to an infinite loop, where `i` keeps getting to `255` and then overflowing again.

Recommendation: Change `i` to be of type `uint256`.

Update: Reply from the KLEX team:

> *In practice, there should never be more than 10 reward tokens.*

QSP-21 Integer Overflow / Underflow

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `contracts/ve.sol`

Related Issue(s): SWC-101

Description: Integer overflow/underflow occurs when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at `11:59`, the minute hand goes to `0`, not `60`, because `59` is the largest possible minute.

Integer overflow and underflow may cause many unexpected kinds of behavior and can be the core reason for attacks.

Solidity versions under `0.8` are vulnerable to integer underflows and overflows.
In the codebase, some potential overflows are:

- `ve.sol`: `L515`

Recommendation: We recommend using the library SafeMath to prevent overflows and underflows.

Update: The KLEX team decided not to modify this operation.

QSP-22 ERC-20 Decimals Are Defined as `uint256`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/KlexToken.sol`

Description: The token deviates from the EIP-20 as `uint256` is used for `decimals` variable instead of `uint8`.

Recommendation: Declare `decimals` as `uint8`.

Update: Fixed in commit `727f401`.

QSP-23 Not Fully Compliant with ERC-721

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/ve.sol`

Description:

1. As stated in EIP-721 and the comments of the functions in the codebase, `_balance(...)` and `balanceOf(...)` must revert for queries about the zero address. This makes the implementation not fully compliant with `ERC721`.
2. As stated in EIP-721, `ownerOf(...)` must revert on queries about tokens assigned to zero the address (which are considered invalid).
3. `safeTransferFrom(...)` does not check the return value of `onERC721Received` (i.e. `retval == IERC721Receiver.onERC721Received.selector`)

Recommendation:

1. Revert for queries about the zero address.
2. Revert for queries about invalid tokens (burned, i.e. assigned to zero address).
3. Check the value returned by `onERC721Received` (see OpenZeppelin's ERC-721 implementation).

Update: The KLEX team has chosen to proceed with a non-standard ERC-721 implementation. We note that this could cause unforeseen issues when integrating with protocols or applications which expect ERC-721 compliance.

QSP-24 Transaction Ordering Dependence for `initialize()` Functions

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `contracts/KlayFut.sol`, `contracts/KlexToken.sol`, `contracts/Lockdrop.sol`, `contracts/MasterChef.sol`, `contracts/MultiFeeDistribution.sol`, `contracts/MultiRewarder.sol`, `contracts/Treasury.sol`, `contracts/Vamp.sol`, `contracts/ve.sol`

Related Issue(s): SWC-114

Description: The various `initialize(...)` functions of upgradeable contracts are not constructors. There's a low-but-not-zero chance that someone can call these after the contracts have been deployed but before the development team calls them. Consequently, contracts may get initialized with values that are not desirable by the development team.
The following contracts are implementation contracts that are currently not automatically initialized:

- `KlayFut`
- `KlexToken`
- `Lockdrop`
- `MasterChef`
- `MultiFeeDistribution`
- `MultiRewarder`
- `Treasury`
- `Vamp`
- `Ve`

Recommendation: Add a constructor with an initializer modifier to each of the affected implementation contracts (see Open Zeppelin's `Initializable` documentation).

Update: The KLEX team has indicated that they feel this is not a real concern. However, we strongly recommend initializing all implementation contracts through the suggested solution or by other means. Not initializing implementation contracts can lead to an attacker self-destructing the implementation contracts.

QSP-25 Emissions May Not Be Updated Properly

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/MasterChef.sol`

Description: Throughout `MasterChef`, `_updatePools(...)` and `_updateEmissions(...)` are called in tandem. However, this is not the case in `_updateFactorIndividual(...)`. This could lead to the `_updatePools(...)` call using an outdated `rewardsPerSecond`.

Recommendation: Add `_updateEmissions(...)` to `_updateFactorIndividual(...)`.

Update: Fixed in commit `9a1fbd7`.

QSP-26 Inconsistent Allowance Behavior

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/KlexToken.sol`

Description: Some ERC-20 tokens such as DAI allow for infinite token approvals by setting the allowance of an approved address to `uint256(-1)`. The `KlexToken` contract implements inconsistent behavior for infinite approvals. The `transferFrom(...)` function appears to respect infinite approval through `L95` to `L97`, which prevent a reduction in allowance if the allowance is set to `uint256(-1)`:

```
if (allowed != uint256(-1)) {
  allowance[_from][msg.sender] = allowed.sub(_value);
}
```

However, the `burnFrom(...)` function does not respect infinite approval, as it will always subtract the transferred amount from the allowance as shown in the function definition from `L155` to `L165` below:

```
function burnFrom(address account, uint256 amount) public {
  uint256 decreasedAllowance = allowance[account][msg.sender].sub(
    amount,
    "ERC20: burn amount exceeds allowance"
  );

  allowance[account][msg.sender] = decreasedAllowance;
  emit Approval(account, msg.sender, decreasedAllowance);

  _burn(account, amount);
}
```

Recommendation: Decide whether the KLEX token supports infinite approvals. If the KLEX token should support infinite approvals, then the `burnFrom(...)` function should be modified so that the allowance is not reduced when the allowance is set to `uint256(-1)`. If the KLEX token should not support infinite approvals, then the `transferFrom(...)` function should be modified so that the allowance is always reduced by the transferred amount.

Update: The KLEX team has fixed the issue by removing support for infinite approvals from the `transferFrom(...)` function. Fixed in commit `cfa9df9`.

QSP-27 `claimKLEX(...)` and `claimKLAY(...)` Can Be Called Anytime

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/Lockdrop.sol`, `contracts/Vamp.sol`

Description: `claimKLEX(...)` and `claimKLAY(...)` in `Lockdrop` and `Vamp` contracts do not have any restriction on when to be called, but they can be called just once per `msg.sender`. However, they rely on `totalDeposits` to calculate the final amount of `KLEX` or `KLAY`, and this variable can change as long as deposits are permitted (`deposit(...)` is available until `depositEndTime`).

Recommendation: Confirm this is the intended behavior. If not, do not allow users to claim before `depositEndTime`.

Update: The KLEX team has fixed the issue by validating that `claimKLEX(...)` and `claimKLAY(...)` are only callable after `depositEndTime`. Fixed in commit `92be884`.

QSP-28 `Lockdrop` and `Vamp` Rewards Can Be Lost

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/Lockdrop.sol`, `contracts/Vamp.sol`

Description: The `claimKLAY(...)` and `claimKLEX(...)` functions on the `Lockdrop` and `Vamp` contracts allow a user to claim their rewards for depositing tokens during the lock period. However, if the user calls either of these functions before rewards have been added to the contract, then they will receive zero tokens and forfeit future rewards.
The implementation for the `claimKLEX(...)` function on the `Lockdrop` contract is shown below. Even if `amountToSend` is `0`, the function still completes execution normally and sets `claimedKLEX[msg.sender][_token]` to `true`, thereby preventing a future valid claim by the user.

```
function claimKLEX(address _token) external nonReentrant {
  require(!claimedKLEX[msg.sender][_token], "Already claimed KLEX");
  claimedKLEX[msg.sender][_token] = true;
  uint256 amountToSend = userInfo[msg.sender][_token].mul(klexRewardsForPool[_token]).div(totalDeposits[_token]);
  IMultiFeeDistribution(multiFee).mint(msg.sender, amountToSend);
  emit ClaimedKLEX(_token, msg.sender, amountToSend);
}
```

The implementation for `claimKLAY(...)` is very similar to `claimKLEX(...)` and has the same issue mentioned above. The `claimKLAY(...)` and `claimKLEX(...)` functions on the `Vamp` contract also have a similar implementation and suffer from the same problem.

Recommendation: Consider adding a `bool` flag that prevents `claimKLEX(...)` and `claimKLAY(...)` from being called until the rewards have been properly set and added to the contract.

Update: The KLEX team has fixed the issue by validating that `claimKLEX(...)` and `claimKLAY(...)` are only callable after `depositEndTime`. They have also added validation checks to ensure that the rewards state variables are not set to `0`. Fixed in commit `92be884`.

QSP-29 Checks-Effects-Interactions Pattern Violation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/MasterChef.sol`

Related Issue(s): SWC-107

Description: The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done.
We have listed all violations of the check-effects-interactions pattern below:

- `MasterChef`
  - `removeVote(...)`: State is changed from `L737` to `L738` after external contract calls.
  - `vote(...)`: State is changed from `L792` to `L794` after external contract calls.

Recommendation: Add a `nonReentrant` modifier to the `removeVote(...)` and `vote(...)` functions or modify the functions to follow the checks-effects-interactions pattern.

Update: The KLEX team has fixed the issue by adding the `nonReentrant` modifier on the `vote` and `removeVote` functions. Fixed in commit `6c0b55e`.

QSP-30 Rewards Stage Does Not Correctly Transition

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/ve.sol`

Description: The reward calculation for `Ve` staking is determined by whether the contract is in the first or second rewards stage as defined by the `completedFirstRewardsStage` storage variable. The value of the `completedFirstRewardsStage` flag is set based on whether the rewards threshold defined by the `MultiFeeDistribution` contract has been reached. Thus, whenever the `Ve` contract updates the `MultiFeeDistribution` contract's rewards amount through the `MultiFeeDistribution.decreaseVeRewardsTotal` function, it should also update the `completedFirstRewardsStage` flag if appropriate. However, this is not being done in the `increase_unlock_time` function of the `Ve` contract which does call the `MultiFeeDistribution.decreaseVeRewards` function. This could lead to the reward calculation on the subsequent transaction incorrectly using the first reward stage's calculation. If the subsequent transaction involves a large lock, then the effect can be significant.

Recommendation: Call the `_check_threshold(...)` function within the `increase_unlock_time(...)` function of the `Ve` rewards contract to transition the contract to the second rewards stage.

Update: Fixed in commit `458bf15`.

QSP-31 Incorrect Accounting for `withdrawableBalance(...)`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/MultiFeeDistribution.sol`

Description: `withdrawableBalance(...)` calculates the balance received and penalty paid on a withdrawal. After calculating the penalty, it calculates the amount received as follows:

```
// Sub 1 because of rounding
amount = balances[user] % 2 == 1
  ? balances[user].sub(penaltyAmount).sub(1)
```

```
: balances[user].sub(penaltyAmount);
```

It is checked that `balances[user]` is an odd number to subtract one. This is not correct in the case that `amountWithoutPenalty` is also an odd number.

Recommendation: Although the approach is correct, the value to be checked should be `balances[user].sub(amountWithoutPenalty)`(before the division).

Update: The KLEX team has indicated that the accounting is correct. The auditing team cannot confirm if the accounting is indeed correct. We recommend that if the accounting is truly correct, the KLEX team should add detailed code comments explaining why.

QSP-32 Misleading Naming

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/MultiRewarder.sol`, `contracts/ve.sol`

Description:

1. The `MultiRewarder` contract sets a reward based on timestamps, but all of the variable names imply it is based on block numbers. This makes the code misleading.
2. The `Ve` contract has functions with very similar names that take in different inputs, specifically `balanceOfAtNFT(...)` and `balanceOfNFTAt(...)`. It is easy to see how one might call the wrong function.

Recommendation:

1. Variables should be renamed to indicate that rewards are based on elapsed time rather than elapsed blocks.
2. Rename the functions to `balanceOfNFTAtBlock()` and `balanceOfNFTAtTimestamp()`, or another name of your choice.

Update: The KLEX team has acknowledged the issue and decided not to fix it as it does not pose a security risk.

QSP-33 Application Monitoring Can Be Improved by Emitting More Events

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/KlexToken.sol`, `contracts/MultiFeeDistribution.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:

- `KlexToken.sol`:
  - `.setWhitelist(...)`
  - `.setBlacklist(...)`
  - `.setTransferPaused(...)`
  - `.setMinter(...)`
- `MultiFeeDistribution.sol`:
  - `.setMinters(...)`
  - `.revokeMinters(...)`
  - `.setMintingStatus(...)`

Recommendation: Consider emitting the events.

Update: The KLEX team has acknowledged the issue and decided not to fix it as it does not pose a security risk.

QSP-34 Using `transfer(...)` for KLAY May Fail in the Future

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/KlayFut.sol`, `contracts/MultiRewarder.sol`, `contracts/Treasury.sol`, `contracts/Vamp.sol`

Description: The functions `address.transfer(...)` and `address.send(...)` assume a fixed amount of gas to execute the call. The use of these functions protects against reentrancy attacks. The default amount of gas, however, may change in the future. In the worst case, it could lead to failed transfers.

Recommendation: We want you to be aware of this possibility. Whereas we do not recommend taking any action now, if you need more flexibility regarding the forwarded gas, you can use `call(...)` to perform transfers.

Update: The KLEX team has acknowledged the issue and decided not to fix it as they claim that it does not pose a security risk. We note that `transfer(...)` could fail and cause a denial of service if the receiver is a fallback function with substantial logic.

QSP-35 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/KlexToken.sol`, `contracts/KlayFut.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method 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()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method 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()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`. Furthermore, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. If this recommendation is followed, note that the current version of `KlexToken` does not implement `increaseAllowance(...)` and `decreaseAllowance(...)`.

Update: The KLEX team has acknowledged the issue and decided not to take any action.

QSP-36 Implicit Use of Interfaces

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/KlexToken.sol`, `contracts/MasterChef.sol`, `contracts/MultiFeeDistribution.sol`, `contracts/MultiRewarder.sol`, `contracts/Treasury.sol`, `contracts/interfaces/*.sol`

Description: Some interfaces are used as a target type when casting contracts, but the contracts do not implement these interfaces explicitly. At the point where the contracts are used, the code assumes that they implement these interfaces. This can lead to difficulty maintaining the code, error-prone development, and unexpected behavior during runtime.

Recommendation: Implement the used interfaces explicitly in contracts. Also, where applicable, instead of casting artifacts from `address` to interfaces, use the interfaces explicitly to preserve typing.

Update: The KLEX team has acknowledged the issue and decided not to fix it as it does not pose a security risk.

QSP-37 Duplicated Libraries

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/libraries/Errors.sol`, `contracts/libraries/MathUtils.sol`, `contracts/libraries/PercentageMath.sol`, `contracts/libraries/WadRayMath.sol`, `contracts/Errors.sol`, `contracts/MathUtils.sol`, `contracts/PercentageMath.sol`, `contracts/WadRayMath.sol`

Description: Some libraries are duplicated in folders `contracts` and `contracts/libraries`.

Recommendation: To keep a good project structure, delete the libraries in `contracts/`, and import the ones in `contracts/libraries` if needed.

Update: The KLEX team has acknowledged the issue and decided not to fix it as it does not pose a security risk.

QSP-38 Upgradability

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/KlayFut.sol`, `contracts/KlexToken.sol`, `contracts/Lockdrop.sol`, `contracts/MasterChef.sol`, `contracts/MultiFeeDistribution.sol`, `contracts/MultiRewarder.sol`, `contracts/Treasury.sol`, `contracts/Vamp.sol`, `contracts/ve.sol`

Description: Several contracts under audit are upgradeable. While this is not a vulnerability, users should be aware that the behavior of the protocol could drastically change if the contracts are upgraded. Furthermore, new vulnerabilities not present during the audit could be introduced in upgraded versions of the contract. We have listed all upgradeable contracts under audit below:

- `KlayFut`
- `KlexToken`
- `Lockdrop`
- `MasterChef`
- `MultiFeeDistribution`
- `MultiRewarder`
- `Treasury`
- `Vamp`
- `Ve`

Recommendation: The fact that the contract can be upgraded and reasons for future upgrades should be communicated to users beforehand.

Update: The KLEX team has indicated that their current documentation informs the users of upgradeability. The Klex team should additionally inform users of any new upgrades to the contracts and upgrade contracts with extreme caution due to the possibility of introducing new bugs.

QSP-39 Allocation Points Can Be Updated After Voting Has Started

Severity: Undetermined

Status: Fixed

File(s) affected: `contracts/MasterChef.sol`

Description: The allocation points of pools can be arbitrarily modified even after voting has begun. This undermines the voting system which functions by increasing allocation points of a pool when a user votes for it. A new pool can be added to the `MasterChef` contract with an arbitrary number of allocation points even after voting has begun. The `addPool(...)` function accepts a `_allocPoint` parameter, the value of which is not constrained by validation. The value of the `_allocPoint` parameter is simply set as the pool's allocation points as seen on `L266:allocPoints[epoch][_token] = _allocPoint;`. Furthermore, it is also possible to update the existing allocation points of any pool even after the voting has begun through the `batchUpdateAllocPoint(...)` function. The function accepts a `_allocPoints` parameter which contains an array of allocation point values. The allocation points of the pools indicated by the `_tokens` parameter, are then updated to these values as seen on `L336:allocPoints[epoch][_tokens[i]] = _allocPoints[i];`.

The `batchUpdateAllocPoint(...)` function has the following comment above:

```
// Update the given pool's allocation point Can only be called by the owner. Refrain from calling post voting start.
```

However, the code does not enforce this restriction.

Recommendation:

1. Checks should be added to the `batchUpdateAllocPoint(...)` function so that it cannot be called after `startTime` has been set.
2. If a new pool is been added after `startTime` has been set, the allocation points for the pool should be set to `1` in the `addPool(...)` function.

Update: The Klex team has fixed the issue by adding validation to the `batchUpdateAllocPoint(...)` function so that it cannot be called after voting has started. The Klex team has also modified the `addPool(...)` function so that a pool can only be added with `_allocPoint` equal to `1` after voting has started. Fixed in `64c2540`.

QSP-40 Potential Miscalculation of `totalAllocPoints`

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `contracts/MasterChef.sol`

Description: `_reset(...)` removes the votes of a `tokenId` from the allocations of pools. `totalWeight` is the sum of the `_votes`, and is subtracted from `totalAllocPoints`. However, the change in allocations is not always the same quantity as `_votes`. If `_votes > allocPoints[_pool]`, `allocPoints[_pool]` is set to zero, which means the allocation will change by a quantity that is not `_votes`.
Here is the code segment:

```
function _reset(uint256 _tokenId) internal {
    address[] storage _poolVote = poolVote[_tokenId];
    uint256 _poolVoteCnt = _poolVote.length;
    uint256 _totalWeight = 0;
    for (uint256 i = 0; i < _poolVoteCnt; i++) {
        address _pool = _poolVote[i];
        uint256 _votes = votes[_tokenId][_pool];

        if (_votes != 0) {
        //_updateFor(gauges[_pool]);
        if (_votes > allocPoints[_pool]) {
          allocPoints[_pool] = 0;
        } else {
          allocPoints[_pool] = allocPoints[_pool].sub(_votes);
        }
        if (_votes > votes[_tokenId][_pool]) {
          votes[_tokenId][_pool] = 0;
        } else {
          votes[_tokenId][_pool] = votes[_tokenId][_pool].sub(_votes);
        }
        _totalWeight = _totalWeight.add(_votes);
        emit Abstained(_tokenId, _votes);
        }
    }
    totalAllocPoints = totalAllocPoints.sub(uint256(_totalWeight));
    usedWeights[_tokenId] = 0;
    delete poolVote[_tokenId];
}
```

Recommendation: If this is not the intended design, change `_totalWeight` so that it is the sum of the changes to `allocPoints`.

Update: Reply from the KLEX team:

> *In practice, _votes should never be greater than allocPoints[_pool] so this issue should never occur. The only place _votes get updated is in the removeVote and vote functions.*

QSP-41 Dead Code

Severity: *Undetermined*

Status: Mitigated

File(s) affected: `contracts/MasterChef.sol`, `contracts/MultiFeeDistribution.sol`, `contracts/ve.sol`

Related Issue(s): SWC-131, SWC-135

Description: Dead code refers to code that is never executed and hence makes no impact on the final result of running a program. Dead code raises concern since either the code is unnecessary or the necessary code's results were ignored.

1. `MasterChef.sol`
   - `EmergencyWithdraw` declared but not used.
   - `usedWeights` and `emissionScheduleLength` are never read. Remove them if they are not needed in outside contracts.
   - The `if (_poolWeight <= 0)` block in `vote(...)` can be removed given that the weights are unsigned integers.
2. `MultiFeeDistribution.sol`
   - Remove `setIncentivesController(...)`.
   - If the contract has not already been deployed, remove the unused variables: `incentivesController`, `lockedSupply`, and `rewards`. If the contract is being upgraded, consider renaming the variables to indicate that they are unused. Remove the unused events: `RewardAdded`, `Withdrawn`, `RewardPaid`, `RewardsDurationUpdated`. Note that `lockedSupply` is defined in `IMultiFeeDistribution`.
   - Remove the unused parameter `slippage` in `exitForVe(...)`.
3. `ve.sol`

- `iMAXTIME` is not used.
- `attachments` is not used.
- Delete `L161` as `voter` is set again in `L164`.

Recommendation: Double-check if this code is needed. If not, delete it.

Update: Fixed in commit `a6e85bb`. The Klex team has removed some of the dead code, but not the following:

- `MasterChef`
  - `usedWeights` and `emissionScheduleLength` are never read. Remove them if they are not needed in outside contracts.
  - The `if (_poolWeight <= 0)`` block in`vote(...)` can be removed given that the weights are unsigned integers.
- `MultiFeeDistribution`
  - • Remove the unused parameter `slippage` in `exitForVe(...)`.

QSP-42 Address Can Be on Blacklist and Whitelist Simultaneously

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/KlexToken.sol`

Description: The `KlexToken` contract features both a `blacklist` and a `whitelist`. However, it is possible for an address that is already on the `blacklist` to be added to the `whitelist` and vice versa. If an address is both on the `blacklist` and the `whitelist` it is unclear what the expected behavior should be.

Recommendation:

1. Add validation to the `setWhitelist(...)` function such that an address on the `blacklist` cannot be added to the `whitelist`.
2. Add validation to the `setBlacklist(...)` function such that an address on the `whitelist` cannot be added to the `blacklist`.

Update: Fixed in commit `cb0beef`.

Automated Analyses

Slither

Slither found 608 results. Most of them were classified as false positives.

Adherence to Specification

KLEX documentation should be reviewed, as we found some inconsistencies (e.g. *KLEX-KLAY LPs can lock their tokens in order to receive veNFTs*).

Code Documentation

1. Each repository's README.md file should contain basic instructions on how to run the test suite and any prerequisites for running tests. For each of these prerequisites also specify the versions supported/tested. At this point, the repository shows a default `README` generated by the *hardhat* library.
2. Every function should at least have a short description of its purpose, its input parameters, and its return value(s) if any. Several functions in the code base do not have such comments which increases the effort of maintainability and the probability of human error when subsequent features are added/modified/removed.
3. In `Lockdrop.sol` and `Vamp.sol` `claimKLEX(...)`, the comment `available as soon as rewards are transferred into contract` is outdated. **Update**: fixed in commit `c2c110a`.
4. `ve.sol`: Update the comment `/// @param token_addr ERC20CRV token address` for `initialize(...)`. **Update**: fixed in commit `c2c110a`.

Adherence to Best Practices

1. `KlexToken.sol`: Update the `require` message in `_transfer(...)` to indicate that the address may be blacklisted as well.
2. `MultiFeeDistribution.sol`, `ve.sol`, `Treasury.sol`: Use the OpenZeppelin implementation of the `ReentrancyGuard`. If the contract is being upgraded, rename `_entered_state` to indicate that it is not being used. The constants can be removed without effect.
3. `MultiFeeDistribution.sol`: Add a require message to the require statement in `setMinters(...)`.
4. `MultiFeeDistribution.sol`: In `penaltyForWithdrawAmount(...)`, store the value of `fullyVestedAmount(...)` in a variable to avoid a redundant call and reduce gas fees.
5. `MultiFeeDistribution.sol`: Move `veStakingRewardsClaimable` to `ve.sol`. There is not a clear reason why it should be in this contract.
6. `ve.sol`: In `getUserFactor()`, there is no need to make an external call to `this.balanceOfNFT(...)` given that the contract has access to the internal function that it wraps. Additionally, replace the reference to `ownerToNFTokenCount` with a call to `_balance(address)`.
7. `ve.sol`: In `increase_unlock_time(...)`, change the `assert` statement to a `require` statement to save gas.
8. Some function names do not follow the Solidity Style Guide naming convention. Internal and private functions and state variables should start with an underscore.
9. Missing SPDX License Identifier. Before publishing the source code, consider adding a comment containing `SPDX-License-Identifier:` to each source file. Use `SPDX-License-Identifier: UNLICENSED` for non-open-source code. See https://spdx.org for more information.
10. Testing and production code should not be mixed. File `contracts/MockERC20.sol` contains a mock token for testing that should be in the `mock` folder.

11. When declaring time periods and for the sake of readability, it is recommended to use the Solidity keywords (e.g. `1 years, 5 weeks`) instead of using the value in seconds. (Seen in `MasterChef` voting functions and `ve` constants).
12. `Treasury.sol`: the variables `_entered` and `_enteredState` can be declared as constants.
13. `ve.sol`: `_isApprovedOrOwner.owner` shadows `owner` from `OwnableUpgradeable`.
14. `KlayFut.sol`: `L35` can be modified to `(bool sent, ) = msg.sender...` as `data` is not used.
15. `MultiRewarder.sol`, `IVe.sol`, `IProtocolFeesCollector.sol`: ABI encoder V2 is no longer experimental (since Solidity 0.7.4). Use `pragma abicoder v2`.
16. `MultiFeeDistribution.sol`: `lockDuration` should be declared as constant.
17. Some functions should be declared external to save gas:

    - `MasterChef.renounceOwnership() (contracts/MasterChef.sol#141)`
    - `MultiFeeDistribution.renounceOwnership() (contracts/MultiFeeDistribution.sol#97)`
    - `MultiRewarder.renounceOwnership() (contracts/MultiRewarder.sol#59)`
    - `Treasury.renounceOwnership() (contracts/Treasury.sol#53)`
    - `Ve.renounceOwnership() (contracts/ve.sol#184)`
    - `KlayFut.initialize() (contracts/KlayFut.sol#7-13)`
    - `KlayFut.mint(address,uint256) (contracts/KlayFut.sol#16-19)`
    - `KlexToken.initialize(uint256) (contracts/KlexToken.sol#27-32)`
    - `KlexToken.burn(uint256) (contracts/KlexToken.sol#140-142)`
    - `KlexToken.burnFrom(address,uint256) (contracts/KlexToken.sol#155-165)`
    - `Lockdrop.initialize(address,address,address,uint256,uint256) (contracts/Lockdrop.sol#41-50)`
    - `MasterChef.initialize(...) (contracts/MasterChef.sol#143-180)`
    - `MasterChef.setEmissionSchedule(uint128[],uint128[]) (contracts/MasterChef.sol#229-243)`
    - `MultiFeeDistribution.initialize(address,address,uint256,address) (contracts/MultiFeeDistribution.sol#72-93)`
    - `MultiRewarder.initialize() (contracts/MultiRewarder.sol#54-57)`
    - `MultiRewarder.reclaimTokens(address,uint256,address) (contracts/MultiRewarder.sol#141-151)`
    - `Treasury.initialize(address) (contracts/Treasury.sol#34-44)`
    - `Treasury.sweepAll() (contracts/Treasury.sol#90-94)`
    - `Vamp.initialize(address,address,address,uint256,uint256) (contracts/Vamp.sol#47-56)`
    - `Ve.initialize(address,address,address) (contracts/ve.sol#153-182)`
    - `Ve.approve(address,uint256) (contracts/ve.sol#426-439)`

Test Results

Test Suite Results

Steps taken to run the test suite:

1. Install project dependencies: `yarn install`
2. Run test script: `yarn test`

All the tests passed.

Update: All the tests passed. One test was added in the fixes commits, although the test quality has not improved. We recommend adding more testing to try to cover all the corner cases that KLEX could have. Proper testing could discover issues in the project.

```
KlayFut
   ✓ test klay fut contract (314ms)

KlexToken
   ✓ check state variables (86ms)
   ✓ mint, balanceOf, transfer, and supply (187ms)
   ✓ pause transfer and play with whitelist / blacklist (516ms)

Lockdrop
   ✓ check state variables (102ms)
   ✓ rescue (54ms)
   ✓ deposit, claim, withdraw (2926ms)
   ✓ pool cap (436ms)

  MasterChef
   ✓ check state variables (72ms)
   ✓ set emission schedule (93ms)
   ✓ add pools and update allocation points (611ms)
   ✓ deposit, withdraw, claim (pre-voting) (2533ms)
   ✓ pool caps (278ms)
   ✓ begin voting and emission schedule kick-off (953ms)
   ✓ vote, remove vote (2248ms)
   ✓ bribes (1363ms)
test ve rewards
   ✓ Ve based rewards

  MultiFee
   ✓ test multi-fee functionalities (1410ms)

  MultiRewarder
   ✓ add multirewarder pool and play with settings (223ms)
   ✓ deposit/withdraw with multirewarder pool in place (2191ms)
1
BigNumber { _hex: '0x00', _isBigNumber: true }
2
BigNumber { _hex: '0x05f5e100', _isBigNumber: true }
3
BigNumber { _hex: '0x0bebc200', _isBigNumber: true }
4
BigNumber { _hex: '0x0bebc200', _isBigNumber: true }
5
BigNumber { _hex: '0x119557c0', _isBigNumber: true }
6
BigNumber { _hex: '0x178b38c0', _isBigNumber: true }
```

```
    ✓ catch this bug (1878ms)

  Treasury
    ✓ initialized state (63ms)
    ✓ rescue functions (223ms)
    ✓ add reward token and burn & redeem (874ms)

  Vamp
    ✓ check state variables (94ms)
    ✓ rescue (52ms)
    ✓ deposit, claim, withdraw (3266ms)
    ✓ pool cap (460ms)


  28 passing (60s)
```

Code Coverage

Code coverage is not supported. We highly recommend the KLEX team enable that feature.

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

```
6f879fab7ce4790aaa252b99a359ec1a6dd29a1dbc86091c8136881c0dc46aaa ./contracts/Lockdrop.sol
59025d7b8ed757d791ed3e7dbdde38162bed6054f034ce5f2d5ca8ff4aa57df6 ./contracts/KlayFut.sol
ada9f6dcb88cd0f2960eb1edbb6fa7ed51016461ebf14fcf36538387c6ea0ffa ./contracts/MasterChef.sol
81035fcde24b854d98e6e0210c095ba49f16e76545e8e74a93de3d868e7aab85 ./contracts/MultiFeeDistribution.sol
2a046ac4790e7cc448e1c01d261469f50e6bb7434d855515116f269abfd1aeae ./contracts/KlexToken.sol
5585d92ce64feb363ef2755e49ccd9001d52f77070f6ef0f5cbee7a22a523284 ./contracts/MultiRewarder.sol
d89c2d0baf91fbdeb89b1f937a85c530c838f753e0990666de95011888fb9c5c ./contracts/Vamp.sol
e04188e0d933ac4f0bf061fbb5c8ad80a3cc2a923a0da09497974ce72078d07f ./contracts/Treasury.sol
aecc522f8ef7945e11c102c6ae4d88284b98396c1caf7d80d038fdde3bdf7ccb ./contracts/ve.sol
75892636cf751e5d5b7efb335db21d3b248215d63a1c08b79eaf5d278ef6ea8c ./contracts/interfaces/ITreasury.sol
e673c182ac486f0e05123ff819a4eb72e51bea6a65ed28571631789b72c0e9bb ./contracts/interfaces/VeInterfaces.sol
13fba376c7381e052c5d30b35904f096008e64c69dde33e70c97958bf397de4a ./contracts/interfaces/IRewarder.sol
5720ba6fbc968dd4334ace766a810cb66b455ba101a1da0715814d3105532f44 ./contracts/interfaces/IMasterChef.sol
2de4b195894f9043de2ab9c26974797a226007768916794 06a696fd8a5f41c14 ./contracts/interfaces/IMultiFeeDistribution.sol
d0e71b8a068cf23c3f720f928c90938103a422606823ce4a0b3af2cb003f818d ./contracts/interfaces/IERC20Detailed.sol
02ddb9462badcd92e17118a899a6513b8af6573acb0376401ac0c20cc307161b ./contracts/interfaces/IVe.sol
1a4345ec5a6a232323670aa822122c91c6a348a8099b6e19e78ac67dfce7f2d1 ./contracts/interfaces/ISwapRouter.sol
e61c88bab974038a1df56108234c28cab4f9cf7335c2ccf584299f7e43f527d3 ./contracts/interfaces/IKlexToken.sol
1cd46fed1610a45411bc783bc7bc605e50b0703c790bd564144bc1c55e090bab ./contracts/interfaces/Balancer/IFlashLoanRecipient.sol
8d18a2a224b329613659963fb9d68d465d4e0a7bd2c34ce0574a61f877c949ee ./contracts/interfaces/Balancer/IProtocolFeesCollector.sol
48d471f11a013feca7b8b01015395561b601c2a87f308f99550828dac0f6924d ./contracts/interfaces/Balancer/IAuthorizer.sol
6372524f9a13d4a2cb1aa605e8969c9de734f9cec79374dcb1339284f999c1d5 ./contracts/interfaces/Balancer/IAsset.sol
```

Tests

```
905cf2576e32f4173ef46b0b4a466f7829d1a38502d957c36f6b329d50d114df ./test/multiRewarder.spec.ts
4b0fcb6ab1e53d490bcab527a26342f50fcd3f74c361e35a0a7ba7756347eb08 ./test/klayFut.spec.ts
dd7dac5def2560538469073ce2fd28246c5d53166c3feaf23c90d7dd0856882b ./test/multifee.spec.ts
acb0977699859abcf8413ee195455537971f3df11f1c17ec0caf58b5785fa40f ./test/masterchef.spec.ts
42405a116b7dffaf27cafdc4cf9acf974629c5c5598249258317bb538f80a844 ./test/treasury.spec.ts
d6378491a6b95c6d2ea05db62a1c143ff73726bcf2db504e6f7b77f144b7cec0 ./test/utils.ts
7b8f6957e30feb17caf6e28615d56055f4050cb4463d0d65b0d711557293eb83 ./test/vamp.spec.ts
60a530f815abef63bfaef8ec0f403934c1a2bef8bc6c5473e2b647ece3a04a9b ./test/Lockdrop.spec.ts
371d3d7cff0d814ae3f96fcb914cb10b293c3ce87e2ad9b5098e5e52127b36af ./test/klexToken.spec.ts
```

Changelog

- 2022-08-30 - Initial report
- 2022-09-15 - Final report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.