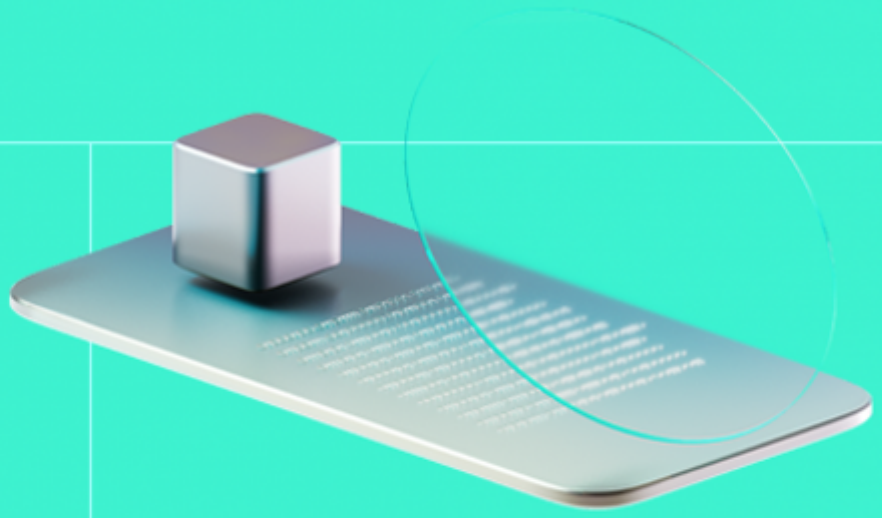
HACKEN

Smart Contract Code Review And Security Analysis Report

Customer: Brickken

Date: 14/02/2024



We express our gratitude to the Brickken team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Brickkenis a solution that provides tools for tokenization of real-world assets, equity, debt and securities.

Language: Solidity

Tags: ERC20, Escrow, Tokenization, Factory

Timeline: 15/01/2024 - 14/02/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Brickken/brickken-protocol
Initial Commit	a4e024b68436ccfcd2d152c47e6179f09fd2d779
Final Remediation Commit	c88c984faaf92c8c841fe129ba214e9bc657fec0

Audit Summary.

10/10	10/10	97.9%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.9/10

The system users should acknowledge all the risks summed up in the risks section of the report

15	14	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	6
Low	8

Vulnerability	Status
F-2024-0589 - Potential Price Manipulation in getBKNPrice() Function in STOFactoryUpgradeable.sol	Accepted
F-2024-0557 - Missing Check for the Return Value of ERC20 Token Transfer	Fixed
F-2024-0558 - Inconsistent Use of tokenERC20Whitelist.multiplier Feature in STOEscrowUpgradeable.sol	Fixed
F-2024-0559 - Incomplete Investor Check in isInvestor() Function in STOEscrowUpgradeable.sol	Fixed
F-2024-0560 - Potential Misconfiguration Risk in STOEscrowUpgradeable.sol	Fixed
F-2024-0561 - Inconsistent Definition of maxSupply in STOToken Contracts leads to maxSupply not being enforced	Fixed
F-2024-0585 - Violation of Check-Effects-Interactions (CEI) Pattern	Fixed
F-2024-0586 - Potential Blocking of STO Token Minting by Issuer in STOEscrowUpgradeable.sol	Fixed
F-2024-0587 - Potential Disruption of Open Positions due to _setPaymentToken() Function in STOEscrowUpgradeable.sol	Fixed
F-2024-0595 - Potential Frontrunning Vulnerability in _addDistDividend() Function in STOTokenDividendUpgradeable.sol	Fixed
F-2024-0596 - Accumulation of Dividends by Blacklisted Users	Fixed
F-2024-0598 - Unclaimed Dividends Affected by _changePaymentToken() Invocation in STOTokenDividendUpgradeable.sol	Fixed
F-2024-0599 - Incorrect Role Assignment in initialize() Function in the STOTokenManagedUpgradeable Contract	Fixed
F-2024-0610 - Missing checks for zero address	Fixed
F-2024-0623 - Missing Data Validation	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Brickken
Audited By	Niccolò Pozzolini, Kornel Światłowski
Approved By	Przemyslaw Swiatowiec
Website	https://www.brickken.com/
Changelog	30/01/2024 - Preliminary Report; 14/02/2024 Second Review

Table of Contents

System Overview **6**

Privileged Roles 6

Executive Summary **8**

Documentation Quality 8

Code Quality 8

Test Coverage 8

Security Score 8

Summary 8

Risks **9**

Findings **10**

Vulnerability Details 10

Observation Details 31

Disclaimers 42

Appendix 1. Severity Definitions **43**

Appendix 2. Scope **44**

System Overview

The Brickken system comprises a factory responsible for generating new instances of an escrow contract and a token contract whenever new tokenization occurs.

Entities authorized to initiate new tokenizations, referred to as issuers, undergo KYC procedures conducted by Brickken. Upon successful verification, issuers are whitelisted in the factory, enabling them to conduct tokenizations. Following issuer whitelisting and the initiation of a new tokenization, both an escrow contract and a token are created. The escrow contract facilitates token offerings, while the token itself incorporates additional functionalities such as dividend distribution and confiscation, while adhering to the ERC20 standard.

Each escrow contract features a base "payment token" in which the issuer withdraws all escrowed funds from investors. Investors can utilize the same payment token for investment or any whitelisted ERC20 tokens. The issuer has the flexibility to modify the whitelist, and each ERC20 token eligible for investment must have a Uniswap v3 pool against the base payment token to facilitate the system's operation. The designated "payment token" is intended to be a stablecoin, although it can take any form as long as a valid Chainlink price feed is available.

Privileged roles

STOFactory

- DEFAULT_ADMIN_ROLE = grant/revoke roles (brickken)
- FACTORY_WHITELISTER_ROLE = allow whitelisting (brickken);
- FACTORY_ISSUER_ROLE = whitelisted issuers (brickken by default);
- FACTORY_PAUSER_ROLE = pause / unpause factory (brickken);

STOToken

- DEFAULT_ADMIN_ROLE = grant/revoke roles (brickken)
- TOKEN_URL_ROLE = change url (brickken,issuer);
- TOKEN_DIVIDEND_DISTRIBUTOR_ROLE = distribute dividend (issuer)
- TOKEN_MINTER_ROLE = mint new tokens (issuer, escrow contract)
- TOKEN_MINTER_ADMIN_ROLE = add/remove minters (issuer)
- TOKEN_WHITELIST_ADMIN_ROLE = change investors whitelist (issuer)
- TOKEN_WHITELIST_ROLE = whether the user is whitelisted or not (issuer)
- TOKEN_CONFISCATE_EXECUTOR_ROLE = execute confiscation (brickken)
- TOKEN_CONFISCATE_ADMIN_ROLE = pause / unpause or disable confiscation (brickken)

STOEscrow:

- DEFAULT_ADMIN_ROLE = grant/revoke roles (brickken)
- ESCROW_WITHDRAW_ROLE = who can withdraw / partially withdraw to issuer (issuer)
- ESCROW_NEW_OFFERING_ROLE = starts a new offering (issuer)
- ESCROW_OFFERING_FINALIZER_ROLE = finalize an offering (brickken, issuer)
- ESCROW_ERC20WHITELIST_ROLE = add/remove ERC20 from whitelist (brickken, issuer);
- ESCROW_OFFCHAIN_REPORTER_ROLE = report offchain USD tickets for current offering (issuer)

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
- Technical description is robust.

Code quality

The total Code Quality score is **10** out of **10**.

- The development Environment is configured.

Test coverage

Code coverage of the project is **97.9%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative test cases are included.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **6** medium, and **8** low severity issues. All issues were fixed in the remediation phase of an audit, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.9**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The withdrawal fees in the `STOEscrowUpgradeable.sol` smart contract are only constrained to be less than 100% (as indicated by the `MAX_FEE_LIMIT`).

Findings

Vulnerability Details

F-2024-0561 - Inconsistent Definition of maxSupply in STOToken Contracts leads to maxSupply not being enforced - Medium

Description:

The `STOToken` contract, composed of multiple inherited contracts, has two different definitions of `maxSupply`. This inconsistency can lead to confusion and the max supply constraint not being enforced.

`STOTokenCheckpointsUpgradeable`, derived from OpenZeppelin's `ERC20Votes`, includes the following method:

```
function _maxSupply() internal view virtual returns (uint224) {
return type(uint224).max;
}
```

This function defines an internal constraint derived from the Checkpoint structure, which contains the `uint224 balance` variable.

On the other hand, `STOTokenUpgradeable` declares a `uint256 public` variable `maxSupply`.

Despite the inconsistency, the two variables behave correctly throughout the function flows. However, the `maxSupply` variable, despite being public, will not be easily accessible from an external context because its default getter function is already defined in `STOTokenCheckpointsUpgradeable`. As a result, when accessed externally, `STOToken.maxSupply()` will always return `type(uint224).max` instead of the intended `STOTokenUpgradeable.maxSupply` variable. This happens, for example, in `STOTokenUpgradeable._newOffering()`.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenCheckpointsUpgradeable.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenUpgradeable.sol

Status: Fixed

Classification

Severity: Medium

Impact: 3/5

Likelihood: 3/5

Recommendations

Recommendation:

To resolve this issue and improve code quality and readability, it is suggested to merge the two `maxSupply` definitions into a single variable. This will ensure consistent behavior and make the `maxSupply` variable consistent.

Remediation: The two `maxSupply` have been differentiated in a base `maxSupply` and a `supplyCap` built on top.

F-2024-0586 - Potential Blocking of STO Token Minting by Issuer in STOEscrowUpgradeable.sol - Medium

Description:

In the `STOEscrowUpgradeable.sol` contract, an issuer can potentially withdraw all the tokens while blocking investors from minting STO tokens. This can be achieved by using solely `_partialWithdraw()` instead of `_withdraw()`, since `_partialWithdraw()` does not change the issuance status to `WITHDRAWN`. If the issuance status is not set to `WITHDRAWN`, investors won't be able to claim their tokens through the function `_redeemToken()`, as it checks that the issuance status is `WITHDRAWN`.

Affected withdrawal functions:

```
/// @dev Internal method to partially withdraw payment token if issuance is succe
sfull
/// @param amountToWithdraw Amount of payment token to withdraw
/// @param withdrawTo address to which the issuer wants to send the withdrawn amo
unt of paymentToken
function _partialWithdraw(uint256 amountToWithdraw, address withdrawTo) internal
{
uint256 issuanceIndexCached = issuanceIndex;

uint256 partialWithdrawn = issuances[issuanceIndexCached].partialWithdraw;
uint256 paymentTokensCollected = issuances[issuanceIndexCached].paymentTokensColl
ected;

uint256 available = paymentTokensCollected - partialWithdrawn;
uint256 finalAmount;

if(available == 0 || amountToWithdraw == 0) return;

if(amountToWithdraw > available) {
finalAmount = available; // REPORT - this branch can be exploited to avoid the WI
THDRAWN status
} else {
finalAmount = amountToWithdraw;
}


issuances[issuanceIndexCached].partialWithdraw = partialWithdrawn + finalAmount;

// Withdraw
uint256 fee = finalAmount.mulDiv(
withdrawalFee,
MAX_FEE_LIMIT,
MathUpgradeable.Rounding.Up
);

SafeERC20Upgradeable.safeTransfer(paymentToken, treasuryAddress, fee);

SafeERC20Upgradeable.safeTransfer(paymentToken, withdrawTo, finalAmount - fee);

emit Withdrawn(issuer, issuanceIndexCached, fee, finalAmount - fee);
}

/// @dev Internal method to withdraw the payment token funds after a successfull
issuance
/// @dev Brickken is getting a successfull fee
function _withdraw(address withdrawTo) internal {
uint256 issuanceIndexCached = issuanceIndex;
if (isWithdrawn(issuanceIndexCached)) revert Errors.IssuanceWasWithdrawn(issuer);

uint256 amount = issuances[issuanceIndexCached].paymentTokensCollected - issuance
s[issuanceIndexCached].partialWithdraw;

if(amount > 0) {
_partialWithdraw(amount, withdrawTo);
}

issuances[issuanceIndexCached].status = IssuanceStatuses.WITHDRAWN;
}
```

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

Status: Fixed

Classification

Severity: Medium

Impact: 4/5

Likelihood: 2/5

Recommendations

Recommendation: It is not possible to solve the issue by changing the function `_partialWithdraw()` because it allows for withdrawals while the issuance is ongoing. Therefore, the recommended solution is to modify the condition in `_redeemToken()`.

In order to solve the issue, the function _redeemToken() should change the condition of the line `if (!isWithdrawn(index)) revert Errors.IssuanceNotWithdrawn(issuer);`.

Here's the recommended change:

```
_checkIssuanceCompleteness(caller, index);
if (!isWithdrawn(index) && (issuances[index].paymentTokensCollected != issuances[
index].partialWithdraw))
revert Errors.IssuanceNotWithdrawn(issuer);
```

This change would allow the check to pass if the issuance status is not `WITHDRAWN`, but all tokens have been withdrawn (i.e., `issuances[index].paymentTokensCollected == issuances[index].partialWithdraw`). On top of that it is required to add `_checkIssuanceCompleteness()` to make sure that the issuance has been finished - this was previously verified by the `WITHDRAWN` status.

These changes would prevent the issuer from blocking investors from minting their STO tokens.

Remediation: The client changed the if branch to allow for token redemptions in the problematic case highlighted in this issue.

F-2024-0587 - Potential Disruption of Open Positions due to _setPaymentToken() Function in STOEscrowUpgradeable.sol - Medium

Description:

The `_setPaymentToken()` function in `STOEscrowUpgradeable.sol` can disrupt open positions. If this function is called during an ongoing issuance which later would rollback, users would receive a different token possibly worth a different value. This function should only be called in between issuances.

Here is the current implementation:

```
/// @dev Internal method to change the payment token, its oracle and the twap win
dow
function _setPaymentToken(address _newPaymentToken, address _newPaymentTokenOracl
e, uint256 _twapInterval) internal {
ERC20Token storage tokenStructNew = tokenERC20Whitelist[_newPaymentToken];

tokenERC20Whitelist[address(paymentToken)].status = false;
tokenStructNew.status = true;
tokenStructNew.multiplier = 1 ether;

paymentTokenOracle = IChainlinkPriceFeed(_newPaymentTokenOracle);
paymentToken = IERC20MetadataUpgradeable(_newPaymentToken);

twapInterval = _twapInterval;
}
```

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

Status: Fixed

Classification

Severity: Medium

Impact: 4/5

Likelihood: 2/5

Recommendations

Recommendation:

To prevent this issue, the function should include a check to verify that no issuance is ongoing. Here's the recommended change:

```
function _setPaymentToken(address _newPaymentToken, address _newPaymentTokenOracl
e, uint256 _twapInterval) internal {
// Check to ensure no issuance is ongoing
if (
(issuanceIndex != 0) &&
!(isWithdrawn(issuanceIndex) || isRollback(issuanceIndex))
) revert Errors.IssuanceNotFinalized(issuer);

// Rest of the function...
}
```

This change would ensure that `_setPaymentToken()` can only be called when no issuance is ongoing, preventing potential disruptions to open positions.

Remediation: The proposed fix has been implemented.

F-2024-0595 - Potential Frontrunning Vulnerability in _addDistDividend() Function in STOTokenDividendUpgradeable.sol - Medium

Description: The `_addDistDividend()` function in `STOTokenDividendUpgradeable.sol` is susceptible to a frontrunning attack. A malicious user monitoring the mempool for the `_addDistDividend()` transaction can buy a large amount of STOTokens in the same block. This tactic allows the newly acquired STOTokens to be included in the current dividend distribution, despite not being a conventional frontrunning attack. The simplicity of this exploit lies in the fact that the purchase of STOTokens merely needs to occur in the same block as the `_addDistDividend()` transaction, significantly lowering the barrier for conducting such an attack.

Assets:

- contracts/sto/UpgradeableTemplate/token/STOTokenDividendUpgradeable.sol

Status: Fixed

Classification

Severity: Medium

Impact: 4/5

Likelihood: 2/5

Recommendations

Recommendation: To fix this issue, `dividendDistributions[numberOfDistributions].blockNumber` should be set to `(block.number - 1)` inside `_addDistDividend()`. Here's the recommended change:

```
dividendDistributions[numberOfDistributions].blockNumber = block.number - 1;
```

This change would ensure that STOTokens bought in the same block as the `_addDistDividend()` transaction do not count towards the current dividend distribution, mitigating the potential frontrunning attack.

Remediation: The proposed fix has been implemented.

F-2024-0598 - Unclaimed Dividends Affected by _changePaymentToken() Invocation in STOTokenDividendUpgradeable.sol - Medium

Description: STOTokens holders accrue dividends possibly spanning multiple distributions, which can be claimed at any time. The `_changePaymentToken()` function in `STOTokenDividendUpgradeable.sol` can potentially affect unclaimed dividends from past distributions. When this function is invoked, users would receive a different token possibly worth a different value for their unclaimed dividends.

Assets:

- contracts/sto/UpgradeableTemplate/token/STOTokenDividendUpgradeable.sol

Status: Fixed

Classification

Severity: Medium

Impact: 3/5

Likelihood: 3/5

Recommendations

Recommendation: If this functionality has to be kept, to fix the issue a new field `address paymentToken` need to be added to the struct `DividendDistribution`, and when starting a new cycle (i.e., when `_addDistDividend()` gets called by the issuer) the current `paymentToken` must be saved in `dividendDistributions[numberOfDistributions].paymentToken` and referenced from there when claiming dividends.

By doing so, when users claim their dividends from a past distribution cycle, they will receive the token specified when that distribution cycle started. This would ensure that the value of the dividends remains consistent with the value at the time of the distribution.

Resolution: The proposed fix has been implemented alongside a pagination feature for the dividends claiming process.

F-2024-0599 - Incorrect Role Assignment in initialize() Function in the STOTokenManagedUpgradeable Contract - Medium

Description: The `initialize()` function in `STOTokenManagedUpgradeable.sol` is not assigning roles correctly. When assigning roles to the issuer, it uses the storage `issuer` variable which is empty, instead of the intended parameter `newIssuer`.

The situation can still be recovered by the protocol owners, who can use their `DEFAULT_ADMIN_ROLE` to properly assign the roles to the issuer, but it would heavily affect the operations efficiency and the user experience.

Status: Fixed

Classification

Severity: Medium

Impact: 2/5

Likelihood: 5/5

Recommendations

Recommendation: To fix this issue, the `newIssuer` parameter should be used instead of the `issuer` variable when assigning roles.

Remediation: The `newIssuer` parameter has been used for role assignment.

F-2024-0557 - Missing Check for the Return Value of ERC20 Token Transfer - Low

Description:

In the `buyToken()` function of the `STOEscrowUpgradeable.sol` file, when a user deposits an excessive amount of `paymentToken`, he gets refunded, but the return value of the `transfer()` operation is not checked.

```
paymentTokenCached.transfer(caller, actualAmount - maxAmount);
```

The `transfer()` return value determines the success of the operation. Whenever the `transfer()` function fails, the overall `buyToken()` function call finishes without revert. Thus, it provides a false sense of successful operation. Various tokens may not follow the ERC20 standard and in case of transfer failure, they might revert or not return any value at all.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation:

To mitigate this risk, it is recommended to use the `SafeERC20` library for ERC20 tokens operations, which handles these edge cases and ensures that transfers fail gracefully.

Remediation: The proposed fix has been implemented.

F-2024-0558 - Inconsistent Use of tokenERC20Whitelist.multiplier Feature in STOEscrowUpgradeable.sol - Low

Description:

The `tokenERC20Whitelist.multiplier` feature in the `STOEscrowUpgradeable.sol` file is not consistently used across different functions, leading to potential confusion and inconsistent expectations for users.

In the `buyToken()` function, the `multiplier` feature is not used at all. However, it is used in the `PriceAndSwapManager.getEstimationSTOToken()` function. This inconsistency can lead to different outcomes than what users might expect.

Furthermore, in the `_setPaymentToken()` function of `STOEscrowUpgradeable`, the `multiplier` is defaulted to `1e18`, suggesting that the feature might be intended for removal.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation:

To resolve this issue, the `multiplier` feature should either be correctly implemented across all relevant functions or removed entirely to avoid confusion and ensure consistent behavior.

Remediation: The `multiplier` feature has been removed.

F-2024-0560 - Potential Misconfiguration Risk in STOEscrowUpgradeable.sol - Low

Description:

The `getUSDPriceOfPaymentToken()` method in `STOEscrowUpgradeable.sol` retrieves the USD price of the `paymentToken` using its `paymentTokenOracle` with 18 decimals of precision. The `paymentTokenOracle` is allowed to be `address(0)` to indicate that the `paymentToken` should not be priced. This is necessary for utility to equity conversion, as explained in the comments of the `PriceAndSwapManager.getUSDPriceOfPaymentToken()` function.

However, since `paymentTokenOracle` is allowed to be `address(0)`, it is not validated when provided to `STOEscrowUpgradeable.__STOEscrowUpgradeable_init()`. This is particularly dangerous and opens the possibility of misconfiguration, which could lead to severe mispricing of the payment token during protocol operations.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol
- contracts/sto/helpers/PriceAndSwapManager.sol

Status: Fixed

Classification

Severity: Low

Impact: 3/5

Likelihood: 2/5

Recommendations

Recommendation:

To minimize the chances of misconfigurations, the need for `paymentTokenOracle` to be `address(0)` should be explicitly defined. This could be achieved by introducing an additional boolean input variable, `paymentTokenOracleUnused`, in the `__STOEscrowUpgradeable_init()` function. This variable would indicate that `paymentTokenOracle` is not meant to be used and can be left unvalidated.

Remediation: The proposed fix has been implemented.

F-2024-0585 - Violation of Check-Effects-Interactions (CEI) Pattern - Low

Description:

State variables are updated after the external calls to the token contract.

As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

- The `_redeemToken()` function in `STOEscrowUpgradeable.sol` does not respect the Check-Effects-Interactions (CEI) pattern. The effect `investors[index][caller].redeemed = true;` should occur before the interaction `stoRelatedToken.mint(caller, amountInSTO);`.
- The `_claimDividends()` function in `STOTokenDividendUpgradeable.sol` does not respect the Check-Effects-Interactions (CEI) pattern. The effect `lastClaimedBlock[currentClaimer] = block.number;` should occur before the interaction `SafeERC20Upgradeable.safeTransfer()`.
- The `_addDistDividend()` function in `STOTokenDividendUpgradeable.sol` does not respect the Check-Effects-Interactions (CEI) pattern. The effect `dividendDistributions[numberOfDistributions].totalAmount = _totalAmount; dividendDistributions[numberOfDistributions].blockNumber = block.number; numberOfDistributions++;` should occur before the interaction `SafeERC20Upgradeable.safeTransferFrom()`.
- The `_changeWhitelist()` function in `STOEscrowUpgradeable.sol` does not respect the Check-Effects-Interactions (CEI) pattern. The effect `tokenERC20Whitelist[token].status = statuses[i]; tokenERC20Whitelist[token].multiplier = multipliers[i]; tokenERC20Whitelist[token].fees = fees[i];` should occur before the interaction `SafeERC20Upgradeable.safeIncreaseAllowance()`.
- The `_withdraw()` function in `STOEscrowUpgradeable.sol` does not respect the Check-Effects-Interactions (CEI) pattern. The effect `issuances[issuanceIndexCached].status = IssuanceStatuses.WITHDRAWN;` should occur before the interaction `SafeERC20Upgradeable.safeTransfer()` executed inside `_partialWithdraw()`.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenDividendUpgradeable.sol

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation:

Moving the Effect before the Interaction would make the function adhere to the CEI pattern, which is a best practice in smart contract development to prevent reentrancy attacks. However, since the interaction is with a contract within the protocol, the risk of such attacks is low in this case. Nonetheless, following the CEI pattern is a good habit to maintain code quality and security.

Remediation: CEI pattern have been enforced.

F-2024-0589 - Potential Price Manipulation in getBKNPrice() Function in STOFactoryUpgradeable.sol - Low

Description:

The `getBKNPrice()` function in `STOFactoryUpgradeable.sol` derives the price from on-chain pool reserves. This price is used to debit issuance fees to issuers. However, this approach is susceptible to manipulation. Issuers can sandwich the issuance emission transaction, manipulating the pool reserves to pay a lower issuance fee.

Here's the current implementation:

```
(uint112 reserveBKN, uint112 reserveUSDT, uint32 blockTimestampLast) = IUniswapV2
Pair(bknOracle).getReserves();
uint256 intermediatePrice = uint256(reserveUSDT).mulDiv(1e18, uint256(reserveBKN)
, MathUpgradeable.Rounding.Up);
```

Assets:

- contracts/sto/UpgradeableTemplate/factory/STOFactory.sol

Status: Accepted

Classification

Severity: Low

Impact: 3/5

Likelihood: 2/5

Recommendations

Recommendation:

To mitigate this issue, a Time-Weighted Average Price (TWAP) should be adopted until a proper BKN oracle is implemented. This would provide a more accurate and harder to manipulate measure of the price over a certain period, reducing the risk of this kind of attack.

Remediation: The migration to a Uniswap v3 pool is planned for the next months.

F-2024-0596 - Accumulation of Dividends by Blacklisted Users - Low

Description: In `STOToken` contracts, blacklisted users will continue accruing dividends which will be locked in the STOToken contract since they won't be able to redeem them. In order to fix the situation, the STOTokens of the blacklisted need to get confiscated.

If confiscation is on, it is suggested to automatically confiscate STOTokens when blacklisting users. It's worth noting that because of the check inside `STOTokenManagedUpgradeable._beforeTokenTransfer()`, it is impossible to confiscate tokens from a blacklisted user.

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation: To address this issue, it is either suggested to implement an auto-confiscate feature for blacklisted users, or to allow the confiscation for blacklisted users. This would prevent blacklisted users from continuing to accrue dividends, which could potentially be a significant amount if not addressed.

Remediation: The `confiscateOnBlacklist` feature has been implemented.

F-2024-0610 - Missing checks for zero address - Low

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The "Missing zero address control" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.
For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

Validation should be added in:

- STOEscrowManagedUpgradeable: initialize(_admin),
- STOFactoryUpgradeable: constructor(), _changeConfig(),
- BeaconProxy: constructor(),
- STOTokenManagedUpgradeable: initialize(_admin)

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowManagedUpgradeable.sol
- contracts/sto/UpgradeableTemplate/factory/STOFactory.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenManagedUpgradeable.sol
- contracts/sto/helpers/BeaconProxy.sol

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation:

It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Remediation: The proposed validations have been implemented.

Evidences

Severity Formula Standart

Reproduce:

Likelihood [1-5]: 2
Impact [1-5]: 2
Exploitability [1-2]: 1
Complexity [0-2]: 0
Final Score: 2.0 (Low)

F-2024-0623 - Missing Data Validation - Low

Description:

Several places of missing data validation were identified:

- The `_setPaymentToken()` function in `STOEscrowUpgradeable.sol` is not validating the new values. The parameters `_newPaymentToken` and `_newPaymentTokenOracle` should be checked against zero. The parameter `newTwapInterval` must also be checked to be within `type(uint32).max` because of its usage in `PriceAndSwapManager.getPriceInPaymentToken`.
- The `changeBKNPriceValidityPeriod()` function in `STOFactoryManagedUpgradeable.sol` does not validate the `newPeriod` parameter. If `newPeriod` is set to zero, it could potentially cause a Denial of Service (DoS) on the protocol.
- The `changeConfig()` function in `STOFactoryManagedUpgradeable.sol` does not validate its inputs. This is particularly dangerous because this function updates many parameters and does not allow for a partial update, opening the risk of misconfiguration.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol
- contracts/sto/UpgradeableTemplate/factory/STOFactoryManaged.sol

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation:

It is suggested to carefully validate the input parameters when updating storage variables.

Remediation: The suggested validations have been implemented.

F-2024-0559 - Incomplete Investor Check in isInvestor() Function in STOEscrowUpgradeable.sol - Info

Description:

The `isInvestor()` function in `STOEscrowUpgradeable.sol` is designed to validate if a user is an investor in a specific issuance. It checks if the user has a non-zero amount of STO tokens and payment tokens, and if they have not yet redeemed their STO tokens.

However, the function does not check if the user has been refunded, which occurs if the issuance is not successful. This means that a user who has been refunded could still be considered an investor according to this function, which is not accurate.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

Status: Fixed

Classification

Severity: Info

Impact: 1/5

Likelihood: 2/5

Recommendations

Recommendation:

To resolve this issue, it is recommended to modify the aforementioned function, so it also checks if the user has not been refunded.

Remediation: The proposed fix has been implemented.

Observation Details

F-2024-0556 - Lack of Named Mappings Feature - Info

Description: The `investors` mapping in the `STOEscrowUpgradeable.sol` file is currently using an indexed mapping structure. As of Solidity version 0.8.18, named parameters can be used in mappings for better readability and understanding of the code. The current code also includes an explanatory comment above the mapping declaration, which could be removed if named parameters are used.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

Status: Fixed

Recommendations

Recommendation: It is recommended to refactor the code to include a named mapping feature. Here is an example how the code could be refactored:

```
/// @dev Issuance Index ---> Address of investor ---> Investor struct
mapping(uint256 => mapping(address => Investor)) public investors;

mapping(uint256 issuanceIndex => mapping(address investorAddress => Investor inve
stor)) public investors;
```

Remediation: The proposed fix has been implemented.

F-2024-0584 - Unnecessary else if Condition in _finalizeIssuance() Function in STOEscrowUpgradeable.sol - Info

Description: In the `_finalizeIssuance()` function of `STOEscrowUpgradeable.sol`, there is an unnecessary `else if` condition that checks the result of `isSuccess(issuanceIndexCached)`. Since `isSuccess()` returns a boolean, the `else if` branch could be simplified to just `else`, which would save gas.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

Status: Fixed

Recommendations

Recommendation: Here's the current implementation:

```
if (isSuccess(issuanceIndexCached)) {
_withdraw(withdrawTo);
} else if (!isSuccess(issuanceIndexCached)) { // Unnecessary 'else if'
_rollBack();
}
```

And here's the recommended change:

```
if (isSuccess(issuanceIndexCached)) {
_withdraw(withdrawTo);
} else { // Simplified to 'else'
_rollBack();
}
```

This change would make the code more efficient by saving gas and also improve readability by simplifying the control flow.

Remediation: The proposed fix has been implemented.

F-2024-0588 - Unnecessary Storage Read in getFeesInBkn() Function in STOFactoryUpgradeable.sol - Info

Description: In the `getFeesInBkn()` function in `STOFactoryUpgradeable.sol`, the storage variable `priceInBKN` is read regardless of whether it's going to be used or not. This could lead to unnecessary gas consumption.

Assets:

- contracts/sto/UpgradeableTemplate/factory/STOFactory.sol

Status: Fixed

Recommendations

Recommendation: Here's the current implementation:

```
amountToPay = priceInBKN;
if (priceInUSD > 0) {
// ...
}
```

And here's the recommended change:

```
if (priceInUSD > 0) {
// ...
} else {
amountToPay = priceInBKN; // Moved inside else block
}
```

This change would ensure that `priceInBKN` is only read when `priceInUSD` is 0, saving gas by avoiding unnecessary storage reads.

Remediation: The proposed fix has been implemented.

F-2024-0592 - Use of Number Literals Instead of Constants in initialize() Function in STOFactoryManagedUpgradeable.sol - Info

Description:

In the `initialize()` function in `STOFactoryManagedUpgradeable.sol`, the variables `priceInBKN` and `priceInUSD` are assigned through number literals. This reduces the code readability.

Here's the current implementation:

```
priceInBKN = uint256(31250e18);
priceInUSD = uint256(5000e18);
```

Assets:

- contracts/sto/UpgradeableTemplate/factory/STOFactoryManaged.sol

Status: Fixed

Recommendations

Recommendation:

For better code readability and to reduce gas consumption, these values should be defined as constants. Here's the recommended change:

```
uint256 constant PRICE_IN_BKN = 31250e18;
uint256 constant PRICE_IN_USD = 5000e18;

priceInBKN = PRICE_IN_BKN;
priceInUSD = PRICE_IN_USD;
```

To further save GAS on operations, the variables `priceInBKN` and `priceInUSD` can be removed in favor of `PRICE_IN_BKN` and `PRICE_IN_USD`, since they would contain the same values.

Remediation: The proposed fix has been implemented.

F-2024-0593 - Redundant Invocation of _moveBalances() in _startTracking() Function in STOTokenCheckpointsUpgradeable.sol - Info

Description:

In the `_startTracking()` function in `STOTokenCheckpointsUpgradeable.sol`, the invocation of `_moveBalances()` is redundant. This function is only called right before the first transfer to a user, so the user's balance will always be zero at this point.

Here's the current implementation:

```
function _startTracking(address account) internal virtual {
address currentTracking = trackings(account);
if(currentTracking != address(0)) return;

uint256 currentTrackingBalance = balanceOf(account);
_trackings[account] = account;

emit TrackingChanged(account, currentTracking, account);

_moveBalances(currentTracking, account, currentTrackingBalance);
}
```

Assets:

- contracts/sto/UpgradeableTemplate/token/STOTokenCheckpointsUpgradeable.sol

Status: Fixed

Recommendations

Recommendation:

Since `currentTrackingBalance` will always be zero, the `_moveBalances()` function does not need to be called. Removing the redundant `_moveBalances()` invocation is recommended to simplify the code and save gas.

Remediation: The proposed fix has been implemented.

F-2024-0594 - Unnecessary Storage Read in getMaxAmountToClaim() Function in STOTokenDividendUpgradeable.sol - Info

Description: In the `getMaxAmountToClaim()` function in `STOTokenDividendUpgradeable.sol`, the variable `dividendDistributions[i].blockNumber` is loaded into memory as `blockNumber`, but it is not used to compute `pastBalance`. Instead, it is read from storage again, which would lead to unnecessary gas consumption.

Assets:

- contracts/sto/UpgradeableTemplate/token/STOTokenDividendUpgradeable.sol

Status: Fixed

Recommendations

Recommendation: Here's the current implementation:

```
uint256 blockNumber = dividendDistributions[i].blockNumber;
uint256 pastBalance = getPastBalance(_claimer, dividendDistributions[i].blockNumb
er);
```

And here's the recommended change:

```
uint256 blockNumber = dividendDistributions[i].blockNumber;
uint256 pastBalance = getPastBalance(_claimer, blockNumber);
```

This change would ensure that `blockNumber` is used to compute `pastBalance`, saving gas by avoiding unnecessary storage reads.

Remediation: The proposed fix has been implemented.

F-2024-0609 - Improve Readability Of Address Tracking - Info

Description:

The current implementation utilizes the `_trackings` mapping structure as follows: `address => address`. In instances where an address is tracked, the key and value are identical; conversely, when an address is not being tracked, the value associated with the key is address `0x0`.

To enhance code readability, it is recommended to consider utilizing the bool data type as the value in the mapping.

Assets:

- contracts/sto/UpgradeableTemplate/token/STOTokenCheckpointsUpgradeable.sol

Status: Fixed

Recommendations

Recommendation:

To improve code clarity and readability, it is advised to refactor the `_trackings` mapping by using bool as the value type. This modification will make the tracking logic more intuitive and straightforward, contributing to a more easily understandable codebase.

Remediation: The proposed fix has been implemented.

F-2024-0622 - IPriceAndSwapManager Contract Located In the Incorrect Folder - Info

Description: Well-structured and organized files enhance the readability of the entire project.

The `IPriceAndSwapManager` contract is labeled as an `interface` and is located inside the "helpers" directory. The project includes a designated directory for `interface` contracts called "interfaces."

Assets:

- contracts/sto/helpers/PriceAndSwapManager.sol

Status: Fixed

Recommendations

Recommendation: It is recommended to move `IPriceAndSwapManager` interface to “interfaces” directory.

Remediation: The `IPriceAndSwapManager` has been moved to the ‘interfaces’ folder.

F-2024-0624 - Commented code parts and TODO comments - Info

Description:

Several instances of commented code parts and TODO comments were observed:

- Within the `STOEscrowUpgradeable` contract, commented sections of code are present within the `__STOEscrowUpgradeable_init()` function.
- Within the `STOFactoryUpgradeable` contract, TODO comment is present within the `getBKNPrice()` function.
- Within the `STOFactoryUpgradeable` contract, TODO comment is present within the `_chargeFee()` function.
- Within the `STOTokenManagedUpgradeable` contract, commented sections of code are present within the `initialize()` function.
- Within the `STOTokenDividendUpgradeable` contract, commented sections of code are present within the `__STOTokenDividend_init()` function.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol
- contracts/sto/UpgradeableTemplate/factory/STOFactory.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenDividendUpgradeable.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenManagedUpgradeable.sol

Status: Fixed

Recommendations

Recommendation:

It is recommended to remove commented parts of the code and resolve TODO comments.

Remediation: The commented-out code lines have been removed.

F-2024-0625 - Increments can be `unchecked` in for-loops - Info

Description:

In Solidity version 0.8 and above, arithmetic operations automatically include checks for underflows and overflows. Although these checks are useful for preventing calculation errors, they consume additional gas, leading to higher transaction costs.

In scenarios where underflows and overflows are not possible, the additional checks introduced by Solidity 0.8 can be bypassed to save gas. This can be done by placing the increment operation inside an `unchecked{}` block. This block enables developers to perform arithmetic operations without the automatic underflow and overflow checks, thus conserving gas when they are not needed.

Function where unchecked can be used:

- STOFactoryManagedUpgradeable: initialize(), changeWhitelist(),
- STOFactoryUpgradeable: _changeConfig(), _newTokenization(),
- PriceAndSwapManager: resetAllowances(),
- STOTokenConfiscateUpgradeable: _confiscate()

Assets:

- contracts/sto/UpgradeableTemplate/factory/STOFactory.sol
- contracts/sto/UpgradeableTemplate/factory/STOFactoryManaged.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenConfiscateUpgradeable.sol
- contracts/sto/helpers/PriceAndSwapManager.sol

Status: Fixed

Recommendations

Recommendation:

To improve gas efficiency, consider placing the post-iteration increment operation at the end of the loop inside an `unchecked{}` code block. This avoids the standard overflow checks, thereby conserving gas. Ensure that this technique is only employed in cases where an overflow is not possible.

Remediation: The proposed fix has been implemented.

F-2024-0627 - Out-Of-Bounds Array Access - Info

Description:

Out-of-bounds array access occurs when a smart contract attempts to read from or write to an index that is beyond the bounds of the array's declared size. This can be due to improper input validation. When an out-of-bounds array access happens, it can disrupt the contract's intended functionality, reverting the transaction.

In each identified case, arrays are utilized in for loops based on the length of the first array, without proper validation to ensure that other arrays used inside these loops share the same length.

The affected functions lacking this crucial validation are:

- STOEscrowUpgradeable: `_changeWhitelist()`,
- STOFactory: `_changeConfig()`,
- STOTokenConfiscateUpgradeable: `_confiscate()`,
- STOFactoryManagedUpgradeable: `initialize()`.

Assets:

- contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol
- contracts/sto/UpgradeableTemplate/factory/STOFactory.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenConfiscateUpgradeable.sol
- contracts/sto/UpgradeableTemplate/token/STOTokenManagedUpgradeable.sol

Status: Fixed

Recommendations

Recommendation:

It is recommended to incorporate proper validation for the length of arrays passed to these functions. This validation will help ensure that arrays used in for loops have consistent lengths, preventing potential disruptions to the contract's intended functionality and avoiding transaction reversion.

Remediation: Array lengths are now validated.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Brickken/brickken-protocol
Commit	a4e024b68436ccfcd2d152c47e6179f09fd2d779
Whitepaper	Not provided
Requirements	https://github.com/Brickken/brickken-protocol/assets
Technical Requirements	https://github.com/Brickken/brickken-protocol/assets

Contracts in Scope

contracts/sto/UpgradeableBeacon/UpgradeableBeaconEscrow.sol

contracts/sto/UpgradeableBeacon/UpgradeableBeaconToken.sol

contracts/sto/UpgradeableTemplate/escrow/STOEscrowManagedUpgradeable.sol

contracts/sto/UpgradeableTemplate/escrow/STOEscrowUpgradeable.sol

contracts/sto/UpgradeableTemplate/factory/STOFactory.sol

contracts/sto/UpgradeableTemplate/factory/STOFactoryManaged.sol

contracts/sto/UpgradeableTemplate/token/STOTokenCheckpointsUpgradeable.sol

contracts/sto/UpgradeableTemplate/token/STOTokenConfiscateUpgradeable.sol

contracts/sto/UpgradeableTemplate/token/STOTokenDividendUpgradeable.sol

contracts/sto/UpgradeableTemplate/token/STOTokenManagedUpgradeable.sol

contracts/sto/UpgradeableTemplate/token/STOTokenUpgradeable.sol

contracts/sto/helpers/BeaconProxy.sol

contracts/sto/helpers/Errors.sol

contracts/sto/helpers/PriceAndSwapManager.sol

contracts/sto/helpers/Roles.sol

contracts/sto/helpers/UniswapLibraries.sol

contracts/sto/interfaces/ChainlinkInterfaces.sol

contracts/sto/interfaces/ISTOToken.sol

contracts/sto/interfaces/UniswapInterfaces.sol