KALP

KALP NETWORK ACCOUNTING SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

JANUARY 30, 2025

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Kalp Network's Accounting Smart Contracts.

Kalp Network is a permissioned, cross-chain blockchain ecosystem designed to integrate regulatory compliance directly into its architecture. It offers a modular infrastructure that supports both public and private sub-networks, ensuring scalability and interoperability across various platforms. The network emphasizes adherence to data privacy laws such as GDPR and incorporates KYC and KYB protocols to maintain a secure and compliant environment. Kalp Network provides tools like Kalp Studio for streamlined decentralized application development and the Kalp Wallet for managing digital assets within its ecosystem.

Kalp Network's Accounting is a specialized token smart contract system designed to operate within the Kalp Chain ecosystem, leveraging Hyperledger Fabric as its underlying technology. The contract manages a fixed-supply token with comprehensive features including KYC integration, role-based access control, and cross-chain bridge support. It implements core token functionalities (transfers, approvals) while maintaining strict security through UTXO-based balance management, address validation, and transaction verification. The system is designed with regulatory compliance in mind, featuring built-in KYC requirements and administrative controls managed by the Kalp Foundation, making it suitable for enterprise-grade financial operations within the Kalp Chain ecosystem.

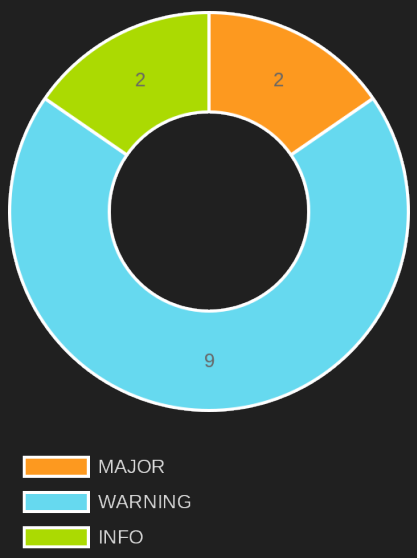
The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the smart contracts to assess the project's security and functionality. The audit covered a total of 8 files, encompassing 1687 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with Kalp Network and referencing the provided documentation to address any questions regarding the expected behavior. For an in-depth explanation of used the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

1.2 SUMMARY OF FINDINGS

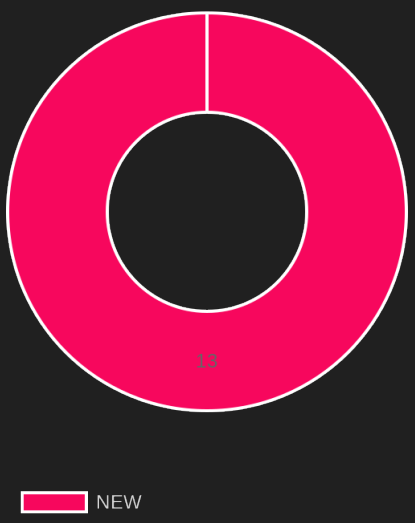
The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**0**	0	0	0	0
MAJOR	**2**	2	0	0	0
WARNING	**9**	9	0	0	0
INFO	**2**	2	0	0	0
TOTAL	**13**	**13**	**0**	**0**	**0**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

1. EXECUTIVE SUMMARY 2

1.1. EXECUTIVE SUMMARY 3

1.2. SUMMARY OF FINDINGS 4

2. AUDIT OVERVIEW 5

2.1. DISCLAIMER 8

2.2. PROJECT BRIEF 9

2.3. PROJECT TIMELINE 10

2.4. AUDITED FILES 11

2.5. PROJECT OVERVIEW 12

2.6. CODEBASE QUALITY ASSESSMENT 13

2.7. FINDINGS BREAKDOWN BY FILE 15

2.8. CONCLUSION 16

3. FINDINGS REPORT 17

3.1. CRITICAL 18

3.2. MAJOR 19

M-01 CompositeKey does not depend on role name in smartcontract.go 19

M-02 Incorrect condition in smartcontract.go 21

3.3. WARNING 23

W-01 Incorrect check of parameter value in internal.go 23

W-02 Approve can be frontrun in smartcontract.go 25

W-03 Missing recipient address check in smartcontract.go 27

W-04 Overpowered GatewayAdmin role in smartcontract.go 28

W-05 Function does not return an error in internal.go 29

W-06 Unreasonable use of PartialCompositeKey in internal.go 31

W-07 Function processes the input array only partially in internal.go 33

W-08 Missing check of KYC status in smartcontract.go 34
W-09 No ability to transfer KalpFoundationRole in smartcontract.go 36
3.4. INFO 37
I-01 Error message ignored 37
I-02 Unused Function 38
4. APPENDIX 39
4.1. SECURITY ASSESSMENT METHODOLOGY 40
4.2. CODEBASE QUALITY ASSESSMENT REFERENCE 42
Rating Criteria 43
4.3. FINDINGS CLASSIFICATION REFERENCE 44
Severity Level Reference 44
Status Level Reference 44
4.4. ABOUT OXORIO 46

2.1 DISCLAIMER

At the request of the client, Oxorio consents to the public release of this audit report. The information contained herein is provided "as is" without any representations or warranties of any kind. Oxorio disclaims all liability for any damages arising from or related to the use of this audit report. Oxorio retains copyright over the contents of this report.

This report is based on the scope of materials and documentation provided to Oxorio for the security audit as detailed in the Executive Summary and Audited Files sections. The findings presented in this report may not encompass all potential vulnerabilities. Oxorio delivers this report and its findings on an as-is basis, and any reliance on this report is undertaken at the user's sole risk. It is important to recognize that blockchain technology remains in a developmental stage and is subject to inherent risks and flaws.

This audit does not extend beyond the programming language of smart contracts to include areas such as the compiler layer or other components that may introduce security risks. Consequently, this report should not be interpreted as an endorsement of any project or team, nor does it guarantee the security of the project under review.

THE CONTENT OF THIS REPORT, INCLUDING ITS ACCESS AND/OR USE, AS WELL AS ANY ASSOCIATED SERVICES OR MATERIALS, MUST NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE. Third parties should not rely on this report for making any decisions, including the purchase or sale of any product, service, or asset. Oxorio expressly disclaims any liability related to the report, its contents, and any associated services, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Oxorio does not warrant, endorse, or take responsibility for any product or service referenced or linked within this report.

For any decisions related to financial, legal, regulatory, or other professional advice, users are strongly encouraged to consult with qualified professionals.

2.2 PROJECT BRIEF

Title	Description
Client	Kalp Network
Project name	Kalp Accounting Smart Contracts
Category	Accounting
Website	www.kalp.network
Documentation	kalp-network.gitbook.io/gini-smartcontracts-documentation
Repository	github.com/p2eengineering/Kalp-Accounting/tree/dev-v1
Initial Commit	`b57b66da2c1a268d2a36ffca19aef67bfe6ef65e`
Platform	L1
Network	Kalp Network
Languages	Go
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Nataly Demidova - nataly@oxor.io

2.3 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
December 26, 2024	Client engaged Oxorio requesting an audit.
January 15, 2025	The audit team initiated work on the project.
January 30, 2025	Submission of the comprehensive audit report.

2.4 AUDITED FILES

The following table contains a list of the audited files. The scc tool was used to count the number of lines and assess complexity of the files.

	File	Lines	Blanks	Comments	Code	Complexity
1	chaincode/constants/constants.go	33	1	0	**32**	0
2	chaincode/events/event.go	129	10	0	**119**	17
3	chaincode/ginierr/error.go	89	19	0	**70**	0
4	chaincode/helper/helper.go	98	21	0	**77**	19
5	chaincode/internal/internal.go	513	57	0	**456**	38
6	chaincode/logger/logger.go	11	3	0	**8**	0
7	chaincode/models/models.go	108	13	0	**95**	21
8	chaincode/smartcontract.go	933	103	0	**830**	57
	Total	**1914**	**227**	**0**	**1687**	**42**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

2.5 PROJECT OVERVIEW

The project represents a GINI token implementation built on the Kalp Network using the Kalp SDK. This is a specialized token smart contract system designed to operate within the Kalp Chain ecosystem, leveraging Hyperledger Fabric as its underlying technology.

The smart contract implements a comprehensive token management system with the following key features:

- **Token Economics**: Implementation of a fixed total supply token with 18 decimals, featuring initial distributions between Foundation and Vesting Contract balances
- **KYC Integration**: Built-in Know Your Customer (KYC) verification system that enforces compliance at the transaction level
- **Gas Fee Management**: Configurable gas fee system with dedicated collection mechanisms for the Kalp Foundation
- **Access Control**: Sophisticated role-based access control system, with special privileges for Kalp Foundation and Gateway Admin addresses
- **Bridge Integration**: Native support for cross-chain operations through a dedicated bridge contract interface

The contract is structured using Golang and implements the following core functionalities:

- Standard token operations (Transfer, Approve, TransferFrom)
- UTXO-based balance management system
- Allowance tracking and management
- Blacklist/Denylist functionality
- Event emission system for transaction tracking

The implementation includes several security mechanisms:

- Strict address validation
- Amount verification systems
- Role-based access controls
- Transaction signing verification
- State management safeguards

This contract serves as a critical component in the Kalp Network's financial infrastructure, facilitating secure token operations while maintaining regulatory compliance through built-in KYC requirements and administrative controls.

2.6 CODEBASE QUALITY ASSESSMENT

The Codebase Quality Assessment table offers a comprehensive assessment of various code metrics, as evaluated by our team during the audit, to gauge the overall quality and maturity of the project's codebase. By evaluating factors such as complexity, documentation and testing coverage to best practices, this table highlights areas where the project excels and identifies potential improvement opportunities. Each metric receives an individual rating, offering a clear snapshot of the project's current state, guiding prioritization for refactoring efforts, and providing insights into its maintainability, security, and scalability. For a detailed description of the categories and ratings, see the Codebase Quality Assessment Reference section.

Category	Assessment	Result
Access Control	The project's codebase implements a robust access control mechanism with multiple differentiated roles to manage system functionalities efficiently. However, the code exhibits undesirable behavior in edge cases during role assignment, as highlighted in issue M-01.	**Good**
Arithmetic	The project has no identified issues related to inadequate handling of arithmetic operations. All arithmetic operations are executed and verified correctly.	**Excellent**
Complexity	The contract appears well-structured; however, attention should be paid to removing unused code. (I-02)	**Good**
Data Validation	The project performs data validation across many components; however, there are gaps in validation under certain conditions. Detailed attention is required to address issues W-01, W-03 and W-07.	**Good**
Decentralization	Contract management is role-based; a decentralized approach is not applicable here.	**Not Applicable**
Documentation	Documentation regarding functionality and limitations was provided, and it is highly helpful in understanding the codebase and its functionality effectively.	**Excellent**
External Dependencies	The project does not interact with any external smart contracts in its logic; therefore, this metric is not applicable in this context.	**Not Applicable**

Category	Assessment	Result
Error Handling	The project demonstrates robust exception handling throughout the codebase, utilizing custom errors with clear naming and descriptions. However, a few minor issues related to error handling (W-05 and I-01) have been identified.	**Good**
Logging and Monitoring	The project exhibits excellent logging capabilities, recording all important events within the system.	**Excellent**
Low-Level Calls	The project is free from low-level calls, ensuring a higher level of security by avoiding potential pitfalls associated with direct, low-level interactions with the blockchain.	**Not Applicable**
Testing and Verification	Working tests were provided for the codebase, with a coverage of 83%, which is generally sufficient. However, not all edge cases are thoroughly tested, as indicated by the identified issues. Expanding test cases to cover these scenarios would enhance the robustness and reliability of the system.	**Good**

2.7 FINDINGS BREAKDOWN BY FILE

This table provides an overview of the findings across the audited files, categorized by severity level. It serves as a useful tool for identifying areas that may require attention, helping to prioritize remediation efforts, and provides a clear summary of the audit results.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
chaincode/smartcontract.go	**8**	0	2	5	1
chaincode/internal/internal.go	**5**	0	0	4	1
chaincode/helper/helper.go	**1**	0	0	0	1

2.8 CONCLUSION

A comprehensive audit was conducted on 8 files, revealing 2 major issues, along with numerous warnings and informational notes. The audit highlighted various attack vectors and potential vulnerabilities, with significant findings related to the management of composite keys in role assignment, the handling of allowance and transaction approval processes, recipient address validation, and logical errors in contract conditions. Additional concerns were identified regarding KYC status checks, overpowered administrative roles, and partial input processing, which could impact the overall reliability and security of the smart contracts.

The proposed changes are aimed at reinforcing role management integrity, ensuring accurate administrative permission enforcement, and enhancing code efficiency and documentation clarity to strengthen the overall security and reliability of the smart contracts. These recommendations are based on adherence to industry best practices, ensuring that these aspects are enhanced to improve the overall security and reliability of the smart contracts. We strongly advise addressing the identified issues to mitigate potential risks, improve the quality of the codebase, and ensure the contracts meet the highest security standards.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

No critical issues found.

3.2 MAJOR

M-01	`CompositeKey` does not depend on role name in `smartcontract.go`
Severity	**MAJOR**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `SetUserRoles`	133-149

Description

In the function `SetUserRoles`, a new role for a user ID is set:

```
key, e := ctx.CreateCompositeKey(constants.UserRolePrefix, []string{userRole.Id,
constants.UserRoleMap})

usrRoleJSON, err := json.Marshal(userRole)

if e := ctx.PutStateWithoutKYC(key, usrRoleJSON); e != nil {
  // ...
}
```

However, since the value of the `key` variable will always be the same for a specific user, adding a new role to a user overwrites (removes) their existing role. This behavior is unexpected and results in users being able to have only one role at a time.

Moreover, this behavior poses a potential risk of losing the `KalpFoundation` role and control over the protocol:

- Initially, during the `Initialize` function, the `KalpFoundation` role is assigned to a special foundation address.
- If the `SetUserRoles` function is later called to assign the `KalpGateWayAdmin` role to the same foundation address, the `KalpFoundation` role will be overwritten and lost.

Recommendation

We recommend making the value of `CompositeKey` depend on the role name. This will prevent unintended overwriting of existing roles when using the `SetUserRoles` function.

M-02	Incorrect condition in `smartcontract.go`
Severity	**MAJOR**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `TransferFrom`	799

Description

In the function `TransferFrom`, the transfer is processed when `signer != sender && signer != recipient && sender == recipient`:

```
func (s *SmartContract) TransferFrom(...) (bool, error) {
  // ...
  if signer != sender && signer != recipient {
    if sender == recipient {
      if sender == constants.KalpFoundationAddress {
        if err = internal.RemoveUtxo(ctx, signer, gasFees); err != nil {
          return false, err
        }
        if err = internal.AddUtxo(ctx, constants.KalpFoundationAddress, gasFees); err != nil
{
          return false, err
        }
      }
    } else {
  // ...
```

Following the current code logic, in this case, the fee is deducted from the `signer` only if `sender == constants.KalpFoundationAddress`. However, the fee should be deducted for any `sender`, and the `signer` should not equal `constants.KalpFoundationAddress`. In the current implementation, the fee is not deducted when `sender != constants.KalpFoundationAddress`, even though it should be.

Recommendation

We recommend replacing the condition `sender == constants.KalpFoundationAddress` with the condition `signer != constants.KalpFoundationAddress`.

3.3 WARNING

W-01	Incorrect check of parameter value in `internal.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
internal.go	function `Mint`	218

Description

In the function `Mint`, there is a validation of the parameters `accAmount1` and `accAmount2`:

```
accAmount1, ok := big.NewInt(0).SetString(amounts[0], 10)
if !ok {
    return ginierr.ErrConvertingAmountToBigInt(amounts[0])
}
if accAmount1.Cmp(big.NewInt(0)) != 1 {
    return ginierr.ErrInvalidAmount(amounts[0])
}
accAmount2, ok := big.NewInt(0).SetString(amounts[1], 10)
if !ok {
    return ginierr.ErrConvertingAmountToBigInt(amounts[1])
}
if accAmount1.Cmp(big.NewInt(0)) != 1 {
    return ginierr.ErrInvalidAmount(amounts[1])
}
```

However, the variable `accAmount1` is mistakenly used instead of the variable `accAmount2` in the second validation check. As a result, the value of the `accAmount2` parameter is not validated and can be negative within the `Mint` function.

Recommendation

We recommend using the `accAmount2` variable in the second validation check instead of the `accAmount1` variable.

W-02	Approve can be frontrun in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `Approve`	316

Description

In the function `Approve`, the allowance value is set by the user's request:

```
func (s *SmartContract) Approve(ctx kalpsdk.TransactionContextInterface, spender string,
amount string) (bool, error) {
  logger.Log.Infoln("Approve invoked.... with arguments", spender, amount)
  if err := models.SetAllowance(ctx, spender, amount); err != nil {
    // ...
  }
  // ...
}
```

The `Approve` function directly sets the spender's allowance, which enables attacker to frontrun the approval transaction. When a user submits multiple `Approve` calls, they inadvertently open a window of opportunity for malicious actors:

- Using the `Approve` function, Alice allows Bob to transfer `x` tokens.
- Later, Alice decides to modify the allowance to `y` and sends another `Approve` request.
- In the meantime, before Alice's new transaction is confirmed, Bob initiates the `TransferFrom` function to transfer `x` tokens from Alice's wallet.
- If Bob's transaction is processed first, followed by Alice's new `Approve` transaction, Bob can also transfer an additional `y` tokens.
- The total unauthorized transfer would amount to `x + y` tokens.

Recommendation

We recommend modifying how allowances are managed. Instead of directly setting new values with `Approve` calls, adopt the use of `increaseAllowance` and `decreaseAllowance`

functions, which specify the allowance changes incrementally. This approach mitigates the risk of frontrunning and improves security.

W-03	Missing recipient address check in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `Transfer`	410

Description

In the function `Transfer`, there is a check ensuring that the sender contract can only be either `bridgeContract` or `vestingContract`:

```
calledContractAddress, err := internal.GetCalledContractAddress(ctx)
if calledContractAddress != s.GetName() {
  if calledContractAddress != bridgeContract && calledContractAddress != vestingContract {
    err := ginierr.New("The called contract is not bridge contract or vesting contract",
http.StatusForbidden)
    logger.Log.Error(err.FullError())
    return false, err
  }
  sender = calledContractAddress
}
```

However, there is no similar check for the recipient contract. This means tokens can be sent to a contract address other than `bridgeContract` or `vestingContract`, and the `Transfer` transaction will succeed. However, retrieving tokens from such a contract may not be possible, effectively freezing the tokens.

Recommendation

We recommend adding a similar check to ensure that if the recipient is a contract, its address must be either `bridgeContract` or `vestingContract`. This will prevent the possibility of token freezing on unsupported contract addresses.

W-04	Overpowered `GatewayAdmin` role in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `Transfer`	378

Description

In the function `Transfer`, there is a special option allowing the owner of the `GatewayAdmin` role to set the sender's address:

```
func (s *SmartContract) Transfer(ctx kalpsdk.TransactionContextInterface, recipient string,
amount string) (bool, error) {
  // ...
    if isGatewayAdmin {
    var gasDeductionAccount models.Sender
    err := json.Unmarshal([]byte(recipient), &gasDeductionAccount)
    // ...

    sender = gasDeductionAccount.Sender
    // ...
```

However, this logic enables the `GatewayAdmin` role holder to transfer tokens from any address at will. This behavior is unexpected for users and should at least be explicitly documented.

Recommendation

We recommend considering limiting the permissions of the `GatewayAdmin` role. For example, it could be restricted to transferring tokens from a user's address only after obtaining prior approval from the user.

W-05	Function does not return an error in `internal.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
internal.go	function `UpdateAllowance`	433

Description

In the function `UpdateAllowance`, the allowance value is updated:

```
func UpdateAllowance(sdk kalpsdk.TransactionContextInterface, owner string, spender string,
spent string) error {
  approvalKey, e := sdk.CreateCompositeKey(constants.Approval, []string{owner, spender})

  approvalByte, e := sdk.GetState(approvalKey)

  var approval models.Allow
  if approvalByte != nil {
    // ...
    approval.Amount = fmt.Sprint(approvalAmount.Sub(approvalAmount, amountSpent))
  }
  approvalJSON, err := json.Marshal(approval)

  e = sdk.PutStateWithoutKYC(approvalKey, approvalJSON)

  return nil
}
```

The `approval.Amount` value is updated only if `approvalByte != nil`. However, if `approvalByte == nil`, which indicates the absence of an allowance, no error is returned, and an empty value is simply saved. Currently, this does not cause an issue as there is a non-zero allowance check in the calling function. However, the `UpdateAllowance` function in isolation does not behave as expected and should return an error when it is unable to decrease the allowance.

Recommendation

We recommend refactoring the `UpdateAllowance` function to return an error when it is unable to decrease the allowance.

W-06	Unreasonable use of `PartialCompositeKey` in `internal.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
internal.go	function `IsGatewayAdminAddress`	131

Description

In the function `IsGatewayAdminAddress`, the `GetStateByPartialCompositeKey` method is used to obtain user roles:

```
func IsGatewayAdminAddress(ctx kalpsdk.TransactionContextInterface, userID string) (bool,
error) {
  prefix := constants.UserRolePrefix
  iterator, e := ctx.GetStateByPartialCompositeKey(prefix, []string{userID,
constants.UserRoleMap})
  if e != nil {
    err := ginierr.NewInternalError(e, fmt.Sprintf("failed to get data for gateway admin:
%v", e), http.StatusInternalServerError)
    logger.Log.Errorf(err.FullError())
    return false, err
  }
  defer iterator.Close()
  // ...
```

However, the use of the `GetStateByPartialCompositeKey` function is unreasonable in this case, as roles are always stored using a full composite key, and in the current implementation, a user can only have one role.

Moreover, according to the documentation, "For a full composite key, an iterator with an empty response would be returned." Based on this, using a full composite key as an argument for the `GetStateByPartialCompositeKey` function makes it impossible to determine if a user has the `KalpGateWayAdmin` role.

Recommendation

We recommend using the `CompositeKey` and `GetState` functions to retrieve a user's role from the storage.

W-07	Function processes the input array only partially in `internal.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
internal.go	function `Mint`	242-247

Description

In the function `Mint`, tokens are minted for the addresses specified in the parameter array:

```
func Mint(ctx kalpsdk.TransactionContextInterface, addresses []string, amounts []string)
error {
  // ...
  if err := MintUtxoHelperWithoutKYC(ctx, addresses[0], accAmount1); err != nil {
  return err
  }
  if err := MintUtxoHelperWithoutKYC(ctx, addresses[1], accAmount2); err != nil {
    return err
  }
  // ...
```

The `Mint` function processes and mints tokens only for the first two addresses in the parameter array. However, there is no check to ensure that the length of the `addresses` array is no greater than 2. In the current implementation, this does not cause an issue as the calling function always passes arrays of length 2. However, the `Mint` function, when used in isolation, does not behave as expected and should either validate that the input array length is 2 or process all elements of the array instead of just the first two.

Recommendation

We recommend refactoring the `Mint` function to either validate that the input array length is 2 or process all elements of the arrays, not just the first two.

W-08	Missing check of KYC status in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `SetUserRoles`	128

Description

In the function `SetUserRoles`, a user is assigned the `KalpGateWayAdminRole`:

```
func (s *SmartContract) SetUserRoles(ctx kalpsdk.TransactionContextInterface, data string)
error {
  // ...
  ValidRoles := []string{constants.KalpGateWayAdminRole}
  if !slices.Contains(ValidRoles, userRole.Role) {
    return fmt.Errorf("invalid input role")
  }
  // ...
```

However, there is no verification to ensure that the new holder of the `KalpGateWayAdminRole` has the required KYC status, as is implemented in the `Initialize` function:

```
func (s *SmartContract) Initialize(...) (bool, error) {
  // ...
  if kyced, e := ctx.GetKYC(constants.KalpGateWayAdminAddress); e != nil {
    err := ginierr.NewInternalError(e, "Error fetching KYC status of Gateway Admin",
http.StatusInternalServerError)
    logger.Log.Errorf(err.FullError())
    return false, err
  } else if !kyced {
    return false, ginierr.New("Gateway Admin is not KYC'd", http.StatusBadRequest)
  }
  // ...
    if _, err := internal.InitializeRoles(ctx, constants.KalpGateWayAdminAddress,
constants.KalpGateWayAdminRole); err != nil {
```

```
    return false, err
    // ...
}
```

Recommendation

We recommend adding a KYC status check for the new holder of the `KalpGateWayAdminRole` to maintain consistency and security.

W-09	No ability to transfer `KalpFoundationRole` in `smartcontract.go`
Severity	**WARNING**
Status	• NEW

Location

File	Location	Line
smartcontract.go	function `SetUserRoles`	128

Description

In the function `SetUserRoles`, a new role is assigned to a user:

```
func (s *SmartContract) SetUserRoles(ctx kalpsdk.TransactionContextInterface, data string)
error {
  // ...
  ValidRoles := []string{constants.KalpGateWayAdminRole}
  if !slices.Contains(ValidRoles, userRole.Role) {
    return fmt.Errorf("invalid input role")
  }
  // ...
```

However, neither this function nor other contract functions provide the ability to transfer the `KalpFoundationRole` to another address after the contract initialization. Such a transfer might be necessary, for example, when transitioning to a different private key or a multisig wallet.

Recommendation

We recommend adding functionality to allow the transfer of the `KalpFoundationRole` to another address.

3.4 INFO

I-01	Error message ignored
Severity	**INFO**
Status	• NEW

Location

File	Location	Line
helper.go	function `IsContractAddress`	40
helper.go	function `IsUserAddress`	50
smartcontract.go	function `TransferFrom`	582

Description

In the mentioned locations, the returned error is ignored and not processed. While this does not currently lead to issues, ignoring errors is a poor programming practice and could result in undesirable consequences.

Recommendation

We recommend processing all returned errors to improve the security and stability of the codebase.

I-02	Unused Function
Severity	**INFO**
Status	• NEW

Location

File	Location	Line
internal.go	function `GetGatewayAdminAddress`	97
internal.go	function `GetUserRoles`	491

Description

In the mentioned locations, there are unused functions that are never utilized within the codebase. These functions add unnecessary clutter and reduce the overall maintainability of the project.

Recommendation

We recommend removing unused functions to improve code readability and maintain overall code quality.

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract security audit methodology is designed to ensure the security, reliability, and compliance of smart contracts throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract’s architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive audit report. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final Audit Report Publication

The final version of the audit report is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the smart contract's security status.

4.2 CODEBASE QUALITY ASSESSMENT REFERENCE

The tables below describe the codebase quality assessment categories and rating criteria used in this report.

Category	Description
Access Control	Evaluates the effectiveness of mechanisms controlling access to ensure only authorized entities can execute specific actions, critical for maintaining system integrity and preventing unauthorized use.
Arithmetic	Focuses on the correct implementation of arithmetic operations to prevent vulnerabilities like overflows and underflows, ensuring that mathematical operations are both logically and semantically accurate.
Complexity	Assesses code organization and function clarity to confirm that functions and modules are organized for ease of understanding and maintenance, thereby reducing unnecessary complexity and enhancing readability.
Data Validation	Assesses the robustness of input validation to prevent common vulnerabilities like overflow, invalid addresses, and other malicious input exploits.
Decentralization	Reviews the implementation of decentralized governance structures to mitigate insider threats and ensure effective risk management during contract upgrades.
Documentation	Reviews the comprehensiveness and clarity of code documentation to ensure that it provides adequate guidance for understanding, maintaining, and securely operating the codebase.
External Dependencies	Evaluates the extent to which the codebase depends on external protocols, oracles, or services. It identifies risks posed by these dependencies, such as compromised data integrity, cascading failures, or reliance on centralized entities. The assessment checks if these external integrations have appropriate fallback mechanisms or redundancy to mitigate risks and protect the protocol's functionality.
Error Handling	Reviews the methods used to handle exceptions and errors, ensuring that failures are managed gracefully and securely.
Logging and Monitoring	Evaluates the use of event auditing and logging to ensure effective tracking of critical system interactions and detect potential anomalies.
Low-Level Calls	Reviews the use of low-level constructs like inline assembly, raw `call` or `delegatecall`, ensuring they are justified, carefully implemented, and do not compromise contract security.

Category	Description
Testing and Verification	Reviews the implementation of unit tests and integration tests to verify that codebase has comprehensive test coverage and reliable mechanisms to catch potential issues.

4.2.1 Rating Criteria

Rating	Description
Excellent	The system is flawless and surpasses standard industry best practices.
Good	Only minor issues were detected; overall, the system adheres to established best practices.
Fair	Issues were identified that could potentially compromise system integrity.
Poor	Numerous issues were identified that compromise system integrity.
Absent	A critical component is absent, severely compromising system safety.
Not Applicable	This category does not apply to the current evaluation.

4.3 FINDINGS CLASSIFICATION REFERENCE

4.3.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

4.3.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.4 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in smart contracts, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING