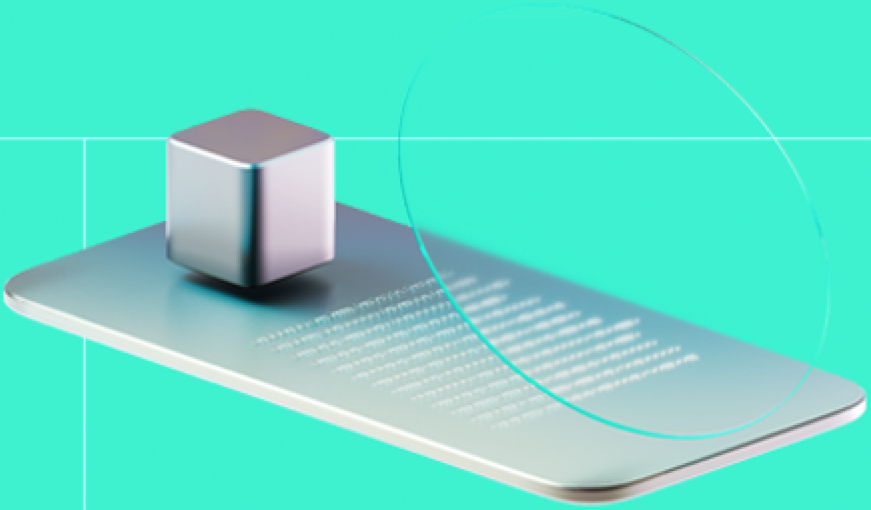
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Kryptomon

Date: 26/02/2024

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

Kryptomon is a groundbreaking game, that adopts Chainlink CCIP to manage multiple chain assets from a single platform.

Platform: EVM

Language: Solidity

Tags: ERC721, ERC1155, ERC20, Chainlink CCIP

Timeline: 06/02/2024 - 26/02/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/KryptomonDAO/chain-migration-contracts
Commit	a36e6a1

Audit Summary

10/10	10/10	95%	6/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.4/10

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

1	1	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	1

Vulnerability	Status
F-2024-0828 - Unchecked Transfer Operations for ERC20 Tokens	Fixed

Document

Name	Smart Contract Code Review and Security Analysis Report for Kryptomon
Audited By	Turgay Arda Usman
Approved By	Grzegorz Trawinski
Website	https://www.kryptomon.co
Changelog	12/02/2024 - Preliminary Report && 26/02/2024 - Final Report

Table of Contents

System Overview

Kryptomon is a groundbreaking game, that adopts Chainlink CCIP to manage multiple chain assets from a single platform. It has the following contracts:

KmonMinter — Minter contract that users interact with.

StakKmonMigrate — The contract that actually mints the NFTs and handles funds transfers to users' wallets.

Privileged roles

- The owner of the `KmonMinter.sol` contract can modify the source chain, allowed addresses, withdraw funds, trigger migrations,
- The owner of the `KmonMigrate.sol` can validate and modify tokens for the system, modify the router and its arguments, modify allowed tokens, withdraw funds, pause the contract
- Allowed addresses can receive ccip messages via `KmonMinter.sol`

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 **6** out of **10**.

- Functional requirements are partially provided.
- Technical description are provided.

Code quality

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

- The code follows best practices and style guides.
- The development environment is configured.

Test coverage

Code coverage of the project is **95.00%** (branch coverage),

- Deployment and basic user interactions are covered with tests.
- Negative case coverage is taken into consideration.
- Interactions by several users are tested .

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **1** low severity issues, 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 **9.4**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The `KmonMigrate.sol` and `KmonMinter.sol` owners can withdraw funds from the contract any time without notifying anyone.
- As per the Chainlink documentation the CCIP version 1.0.0 has been deprecated, on mainnet. The system should take that into consideration.
- This audit report focuses exclusively on the security assessment of the contracts within the specified review scope. Interactions with **out-of-scope contracts are presumed to be correct and are not examined in this audit**. We want to highlight that Interactions with contracts outside the specified scope, such as:
    - DAOFacet.sol
    - LibItems.sol
    - LibAppStorage.sol
    - LibERC1155.sol
    - DAOFacet.sol
    - ItemsFacet.sol
    - KmonMigrationFacet.sol
    - LibKmonMigration.sol
    - LibKryptomon.sol
    - KryptomonFacet.sol
    - LibStrings.sol
    - ItemsTransferFacet.sol
    - LibMeta.sol
    - LibERC721.sol

  have not been verified or assessed as part of this report.

While we have diligently identified and mitigated potential security risks within the defined scope, it is important to note that our assessment is confined to the isolated contracts within this scope. The overall security of the entire system, including external contracts and integrations beyond our audit scope, cannot be guaranteed.

Users and stakeholders are urged to exercise caution when assessing the security of the broader ecosystem and interactions with external contracts. For a comprehensive evaluation of the entire system, additional audits and assessments outside the scope of this report are necessary.

In other words, HACKEN hereby disclaims any responsibility for security issues arising from interactions with out-of-scope contracts, including but not limited to

- DAOFacet.sol
- LibItems.sol
- LibAppStorage.sol
- LibERC1155.sol
- DAOFacet.sol
- ItemsFacet.sol
- KmonMigrationFacet.sol
- LibKmonMigration.sol
- LibKryptomon.sol

- KryptomonFacet.sol
- LibStrings.sol
- ItemsTransferFacet.sol
- LibMeta.sol
- LibERC721.sol

Despite HACKEN's differing opinion on the matter, it is explicitly stated that security checks were not conducted on these out-of-scope interactions. HACKEN cannot be held liable for any security issues that may have occurred in connection with these out-of-scope contracts or any issues arising from the interactions with them, as HACKEN was not granted permission to assess their logic.

This report serves as a snapshot of the security status of the audited contracts within the specified scope at the time of the audit. We strongly recommend ongoing security evaluations and continuous monitoring to maintain and enhance the overall system's security.

Findings

Vulnerability Details

F-2024-0828 - Unchecked Transfer Operations for ERC20 Tokens - Low

Description:

The analysis identified that there are omitted verifications for the return values of ERC20 transfer functions. This oversight can lead to vulnerabilities since certain tokens might deviate from the ERC20 standards, either by returning `false` upon a transfer failure or by not issuing any return value whatsoever.

```
function withdrawERC20(
address _tokenAddress,
address _beneficiary,
uint256 _amount
) public onlyOwner {
IERC20 token = IERC20(_tokenAddress);
uint256 contractTokenBalance = token.balanceOf(address(this));
if (_amount > contractTokenBalance) {
revert FailedToWithdrawErc20(
_tokenAddress,
_msgSender(),
_beneficiary,
_amount,
contractTokenBalance
);
}

bool success = token.transfer(_beneficiary, _amount);

if (!success)
revert FailedToWithdrawErc20(
_tokenAddress,
_msgSender(),
_beneficiary,
_amount,
contractTokenBalance
);
}
```

Functions that transfer do not use SafeERC20 and do not check return value of transfers:

- `withdrawERC20()`

Assets:

- KmonMinter.sol [https://github.com/KryptomonDAO/chain-migration-contracts]

Status: Fixed

Classification

Severity:	Low
Impact:	Likelihood [1-5]: 3 Impact [1-5]: 3 Exploitability [0-2]: 2 Complexity [0-2]: 1 **Final Score:** 2.0 (Low)

Recommendations

Recommendation: Implement the SafeERC20 library to check the `return` value of the calls to ERC20 `transfer`and `transferFrom`, as well as interacting safely with tokens that do not return anything at all

Remediation (Commit:89402c8da38be7d7c6f8ba7e6786dd89c7f7e940): The SafeERC20 library has adopted.

Observation Details

F-2024-0827 - Floating Pragma - Info

Description:	The project uses floating pragmas `0.8.20` and `0.8.17` This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively. Additional assets affected: • contract-eab6af91ae.sol: https://etherscan.io/address/0xc4170fd71eced3c80badca77f4e12e8aac1e3436#code
Assets:	• KmonMigrate.sol [https://github.com/KryptomonDAO/chain-migration-contracts] • KmonMinter.sol [https://github.com/KryptomonDAO/chain-migration-contracts] • ItemTypes.sol [https://github.com/KryptomonDAO/chain-migration-contracts]
Status:	Fixed

Recommendations

Recommendation:	Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen. **Remediation (Commit: 2325f0431ebe9676800e0a4aa146d6d2da4976d8):** The pragma version is locked.

F-2024-0830 - Out-of-Gas Error Due to Excessive Loop Iterations

Edit Write comment Edit - Info

Description:

Removing and setting allowed tokens are handled in batches. While doing so, the related functions, `setAllowedTokens()` and `removeAllowedTokens()`, accept array parameters, and then iterate through them. This extensive looping may exceed the maximum block Gas limit, leading to a revert with out-of-gas errors and rendering the `setAllowedTokens()` and `removeAllowedTokens()` functions unusable.

```
function setAllowedTokens(
address[] calldata _tokenAddress,
uint8[] calldata _assetType,
address[] calldata _receiver,
address[] calldata _targetToken,
uint64[] calldata _targetChain
) external onlyOwner {
if (_tokenAddress.length != _assetType.length)
revert LengthMismatch("tokenAddress-assetType");
if (_assetType.length != _receiver.length)
revert LengthMismatch("assetType-receiver");
if (_receiver.length != _targetToken.length)
revert LengthMismatch("receiver-targetToken");
if (_targetToken.length != _targetChain.length)
revert LengthMismatch("targetToken-targetChain");

bytes32 combinedHash;
for (uint256 i; i < _tokenAddress.length; ) {
require(_assetType[i] <= 2, "assetType > 2 not allowed");

require(allowedTargetChains[_targetChain[i]], "targetChain not allow
ed");

// Calculate the hash of the concatenated address and uint8 value
combinedHash = getCombinedHash(_tokenAddress[i], _assetType[i]);

emit AllowedTokenAdded(
_tokenAddress[i],
_assetType[i],
_receiver[i],
_targetToken[i],
_targetChain[i],
combinedHash
);

tokenInfo[combinedHash] = Token({
allowed: true,
tokenAddress: _tokenAddress[i],
assetType: _assetType[i],
receiver: _receiver[i],
targetToken: _targetToken[i],
targetChain: _targetChain[i]
});

unchecked {
i++;
}
}
}
```

Assets:

- KmonMigrate.sol [https://github.com/KryptomonDAO/chain-migration-contracts]

Status: Fixed

Recommendations

Recommendation: Set a reasonable upper limit for the maximum number of transactions processed within these functions.

Remediation (Commit: 2325f0431ebe9676800e0a4aa146d6d2da4976d8): The following check has been implemented:

```
if (_tokenAddress.length > 100)
revert ArrayTooBig("Total allowed tokens must be equal or less than
100");
```

F-2024-0831 - Missing Zero Address Validation - Info

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, consider a contract that includes a function to change its owner. This function is crucial, as it determines who has administrative access. However, if this function lacks proper validation checks, it might inadvertently permit the setting of the owner to the zero address. Consequently, the administrative functions will become unusable.

The constructors in `withdrawNative()` and `withdrawERC20()` functions are lack of missing zero address validation.

Assets:

- KmonMigrate.sol [https://github.com/KryptomonDAO/chain-migration-contracts]

Status: Fixed

Recommendations

Recommendation: Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Remediation (Commit: 2325f0431ebe9676800e0a4aa146d6d2da4976d8): Zero address has been implemented.

F-2024-0832 - Redundant Receive() Function - Info

Description: The Kryptomon project benefits from the CCIP architecture of Chainlink. This allows its users to send messages between chains. The implemented architecture states that the designed flow is from `KryptomonMigrate.sol` to `KryptomonMinter.sol` contract. This means that no funds will be transferred from minter to migrator contract, this makes the `receive()` function implemented in the `KryptomonMigrate.sol` contract redundant.

Assets:

- KmonMigrate.sol [https://github.com/KryptomonDAO/chain-migration-contracts]

Status: Fixed

Recommendations

Recommendation: It is recommended to remove this function to prevent any accidental fund transfers to the contract.

Remediation (Commit: 2325f0431ebe9676800e0a4aa146d6d2da4976d8): The redundant function has been removed.

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.

Risk Disclaimer

This audit report focuses exclusively on the security assessment of the contracts within the specified review scope. Interactions with **out-of-scope contracts are presumed to be correct and are not examined in this audit**. We want to highlight that Interactions with contracts outside the specified scope, such as:

- DAOFacet.sol
- LibItems.sol
- LibAppStorage.sol
- LibERC1155.sol
- DAOFacet.sol
- ItemsFacet.sol
- KmonMigrationFacet.sol
- LibKmonMigration.sol
- LibKryptomon.sol
- KryptomonFacet.sol
- LibStrings.sol
- ItemsTransferFacet.sol
- LibMeta.sol
- LibERC721.sol

have not been verified or assessed as part of this report.

While we have diligently identified and mitigated potential security risks within the defined scope, it is important to note that our assessment is confined to the isolated contracts within this scope. The overall security of the entire system, including external contracts and integrations beyond our audit scope, cannot be guaranteed.

Users and stakeholders are urged to exercise caution when assessing the security of the broader ecosystem and interactions with external contracts. For a comprehensive evaluation of the entire system, additional audits and assessments outside the scope of this report are necessary.

In other words, HACKEN hereby disclaims any responsibility for security issues arising from interactions with out-of-scope contracts, including but not limited to

- DAOFacet.sol
- LibItems.sol
- LibAppStorage.sol
- LibERC1155.sol
- DAOFacet.sol
- ItemsFacet.sol
- KmonMigrationFacet.sol
- LibKmonMigration.sol
- LibKryptomon.sol
- KryptomonFacet.sol
- LibStrings.sol
- ItemsTransferFacet.sol
- LibMeta.sol
- LibERC721.sol

Despite HACKEN's differing opinion on the matter, it is explicitly stated that security checks were not conducted on these out-of-scope interactions. HACKEN cannot be held liable for any security issues that may have occurred in connection with these out-of-scope contracts or any issues arising from the interactions with them, as HACKEN was not granted permission to assess their logic.

This report serves as a snapshot of the security status of the audited contracts within the specified scope at the time of the audit. We strongly recommend ongoing security evaluations and continuous monitoring to maintain and enhance the overall system's security.

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/KryptomonDAO/chain-migration-contracts
Commit	2325f0431ebe9676800e0a4aa146d6d2da4976d8
Whitepaper	-
Requirements	provided as files
Technical Requirements	provided as files

Contracts in Scope

./contracts/KmonMigrate.sol

./contracts/KmonMinter.sol

./contracts/interfaces/IDiamond.sol

./contracts/interfaces/IERC20.sol

./contracts/interfaces/IERC721.sol

./contracts/utils/ItemTypes.sol

contract-eab6af91ae.sol:

https://etherscan.io/address/0xc4170fd71eced3c80badca77f4e12e8aac1e3436#code