CERTIK

Security Assessment

Kyoko - V

CertiK Verified on Sept 28th, 2022

CertiK Verified on Sept 28th, 2022

Kyoko - V

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Ethereum	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 09/28/2022	N/A

CODEBASE
https://github.com/kyoko-finance/kyoko-ccal-contract
...View All

COMMITS
0b23657b352a4afbb2a9bc5d1851c74127191ab3
24fab137c99cd0489e9a5f8ddf3f63a51b7c1337
...View All

Vulnerability Summary

20	12	0	2	6	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
3	Major	1 Resolved, 2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.

8	Minor	6 Resolved, 2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
8	Informational	4 Resolved, 2 Partially Resolved, 2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | KYOKO - V

Disclaimer

CODEBASE | KYOKO - V

Repository

https://github.com/kyoko-finance/kyoko-ccal-contract

Commit

0b23657b352a4afbb2a9bc5d1851c74127191ab3

24fab137c99cd0489e9a5f8ddf3f63a51b7c1337

AUDIT SCOPE | KYOKO - V

15 files audited ● 3 files with Acknowledged findings ● 1 file with Resolved findings ● 11 files without findings

ID	File	SHA256 Checksum
BCB	contracts/BaseContract.sol	4ac9455526f54f59610afa3357d8294502bedfb47111a112952b83a5c3c80a86
CCA	contracts/CCALMainChain.sol	136fe731446a52a7b77502f2cd845aa51e02a2475c23d616260ab9a134b49194
CCL	contracts/CCALSubChain.sol	d4b5db6656ce257a54cb1765ca4d56f5a3f1b3ba342c74d65715aa9b48f6789d
VLB	contracts/libs/ValidateLogic.sol	e6058c6a86dc4e88edc5c70ecaec50e3d7a3dab08c3590b7f62d45e25732b220
ILZ	contracts/LayerZero/ILayerZeroEndpoint.sol	eba8eb43d410e92ca9501c5d7f3edc2e255a9c677c0ea47c5aeac931901c7ea3
ILM	contracts/LayerZero/ILayerZeroMessagingLibrary.sol	f247cb8c2c6a96e9e66aca41832fd34bd5ab361e1b3c4b170b2e612d230d7393
ILO	contracts/LayerZero/ILayerZeroOracle.sol	41638769693f40bb2f770f32c57303577fb36928979108b458573c71dc7c8caf
ILR	contracts/LayerZero/ILayerZeroReceiver.sol	0f44c756eeece9519458fe054d9cd554acbeb585e661e6915f17089be9ffc7f7
ILL	contracts/LayerZero/ILayerZeroRelayer.sol	409c54477fbe90ff18263c265729da215e01273e558c6edfe0e59da196cfe9ce
ILU	contracts/LayerZero/ILayerZeroUserApplicationConfig.sol	a2dc353187811421389aaabd35a56b6359873b1bc3f01f9ce22b858ac085335a
ERR	contracts/libs/Errors.sol	48605900bf2d8cfb97917553b90ac2c951e93455608607722305bb7b013c9ba8
HEL	contracts/libs/Help.sol	dddd1f5ba1456d0bfeea76b758c472e0e1ca41e119e700a99ccb56d000590252
PCB	contracts/ProjectConfig.sol	ef1235c2d78b970c2afdfdfc2ed2085194efe3b88c02e02ddbeb3a049fc15988

ID	File	SHA256 Checksum
SLB	contracts/StorageLayer.sol	405e27bd49024b06880220a96dbf5413357a1dab335025a8cf3c9bf67d3f7796
INT	contracts/interface.sol	52a3f7dee798b090479d133e38359dd6554385b3023b4d433dfe4a6de0b41f25

APPROACH & METHODS | KYOKO - V

This report has been prepared for Kyoko to discover issues and vulnerabilities in the source code of the Kyoko - V project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | KYOKO - V

20	0	3	1	8	8
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Kyoko - V. Through this audit, we have uncovered 20 issues ranging from different severity levels. Utilizing Static Analysis techniques to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CCA-01	No Restriction On `_fee` Input In `initialize()`	Inconsistency	Minor	Resolved
CCA-02	Checking The State Variable `fee` Instead Of The Input `_fee`	Logical Issue	Minor	Resolved
CCA-03	Missing Validation Checks That `_dstChainId` Is Not `_selfChainId`	Data Flow	Minor	Resolved
CCL-01	Locked Ether	Language Specific	Minor	Resolved
CON-01	**Centralized Control Of Contract Upgrade**	**Centralization / Privilege**	**Major**	**Acknowledged**
CON-02	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
CON-03	Conflicting Initialization Modifiers	Volatile Code	Major	Resolved
CON-04	Implementation Contract Is Not Initialized Automatically	Control Flow	Medium	Resolved
CON-05	Third Party Dependency	Volatile Code	Minor	Acknowledged
CON-06	Missing Zero Address Validation	Volatile Code	Minor	Acknowledged

ID	Title	Category	Severity	Status
CON-07	Check Effect Interaction Pattern Violated	Logical Issue	Minor	Resolved
VLB-01	Calculation For `totalAmount` Validation May Be Incorrect	Mathematical Operations, Logical Issue	Minor	Resolved
CCA-06	Comparison To Boolean Constant	Coding Style	Informational	Resolved
CCA-07	NFT Borrowing Logic At Risk For Tokens Being Taken	Control Flow	Informational	Partially Resolved
CCA-08	Not All Stable ERC20 Tokens Hold Equivalent Value	Logical Issue	Informational	Resolved
CCL-02	Unused `AccessControl` Utility	Volatile Code	Informational	Acknowledged
CCL-03	`msg.value` Sent Back To `msg.sender`	Logical Issue	Informational	Resolved
CCL-04	Dependency That `layerZeroEndpoint` Is Secure	Control Flow	Informational	Acknowledged
CON-08	Missing Emit Events	Coding Style	Informational	Partially Resolved
CON-09	Shadowing Local Variable	Coding Style	Informational	Resolved

CCA-01 | NO RESTRICTION ON `_fee` INPUT IN `initialize()`

Category	Severity	Location	Status
Inconsistency	Minor	contracts/CCALMainChain.sol: 53~54	Resolved

Description

There is no check in the function `initialize()` that the input for `_fee` does not exceed a chosen value. The function `setFee()` includes a check that indicates the state variable `fee` should not exceed 1000, so that it is worth no more than 10% of the `BASE_FEE` value. However, in `initialize()`, this variable can be set to any value, causing any calculations involving this variable to wildly differ depending on the input.

Recommendation

We recommend including a restriction at the beginning of the `initialize()` function that the value of input `_fee` does not exceed 1000.

Alleviation

`[CertiK]`: The client heeded the recommendation above by passing the `_fee` input in `initialize()` to the `setFee()` function which checks the value against the maximum value of 1000. This change was made in hash 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CCA-02 CHECKING THE STATE VARIABLE `fee` INSTEAD OF THE INPUT `_fee`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/CCALMainChain.sol: 71~72	Resolved

Description

In the function `setFee()`, there is a check that the state variable `fee` does not exceed 1000 before `fee` is updated with the new input `_fee`. This does not prevent `fee` from being set higher than 1000, it only prevents it from being set a second time once the value is set to be larger than 1000. The restriction still allows for `fee` to be set to any value at least once.

Recommendation

We recommend replacing `fee` with the input value `_fee` in the check:

```
require(fee <= 1000, Errors.SET_FEE_TOO_LARGE);
```

Alleviation

`[CertiK]`: The client heeded the advice and made the change outlined in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CCA-03 MISSING VALIDATION CHECKS THAT `_dstChainId` IS NOT `_selfChainId`

Category	Severity	Location	Status
Data Flow	Minor	contracts/CCALMainChain.sol: 197~198	Resolved

Description

If a user inputs the `selfChainId` as the `_dstChainId` in function `borrowOtherChainAsset()`, then when `layerZeroEndpoint` calls `lzReceive()`, the processing of bytecode `_payload` will fail without revert, resulting in the caller of `borrowOtherChainAsset()` paying to borrow an NFT that will not be received.

Recommendation

We recommend adding a check that the input `_dstChainId` is not `selfChainId` at the beginning of function `borrowOtherChainAsset()`.

Alleviation

`[CertiK]`: The `Kyoko` team heeded the recommendation and made the changes outlined in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CCL-01 | LOCKED ETHER

Category	Severity	Location	Status
Language Specific	Minor	contracts/CCALSubChain.sol: 21~22	Resolved

Description

The contract `CCALSubChain` has a `receive()` and `fallback()` function inherited from `BaseContract`, however, there is no way to withdraw the ETH funds sent directly to the contract.

Recommendation

We recommend removing the ability to send ETH to the contract directly if it is not necessary, or alternatively, adding a withdraw function.

Alleviation

`[CertiK]`: The `Kyoko` team heeded the recommendation by moving the `withdrawEth()` function from `CCALMainChain` to `BaseContract`, allowing the function to be present in both `CCALMainChain` and `CCALSubchain`. This change was made in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CON-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/BaseContract.sol: 35; contracts/CCALMainChain.sol: 28~29; contracts/CCALSubChain.sol: 21~22**	**Acknowledged**

Description

Contracts `CCALSubChain` and `CCALMainChain` are upgradeable contracts; the owner can upgrade the contract without the community's commitment. If an attacker compromises the account, they can change the implementation of the contract and drain tokens from the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

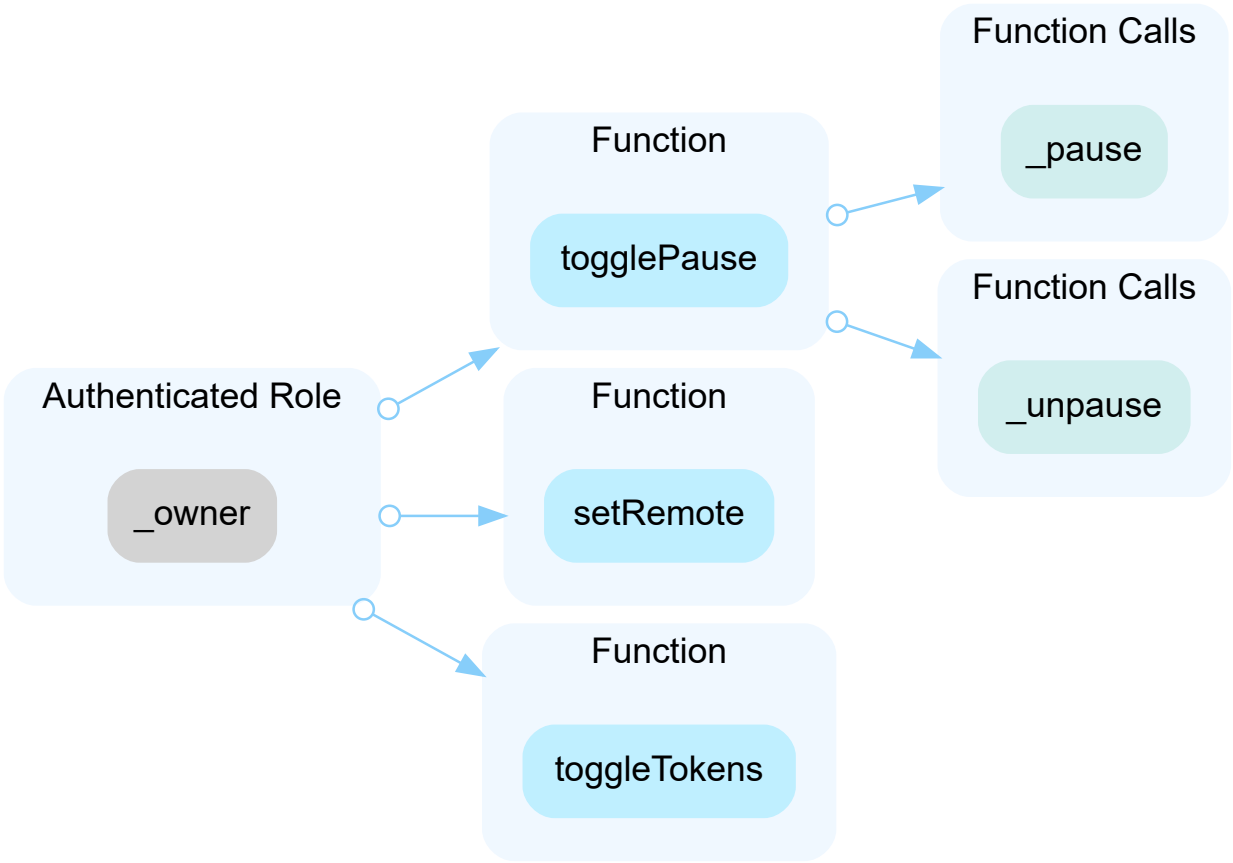
`[CertiK]`: The `Kyoko` team acknowledges the finding and states they are working toward employing multi-signature wallets to mitigate this finding.

CON-02 | CENTRALIZATION RELATED RISKS

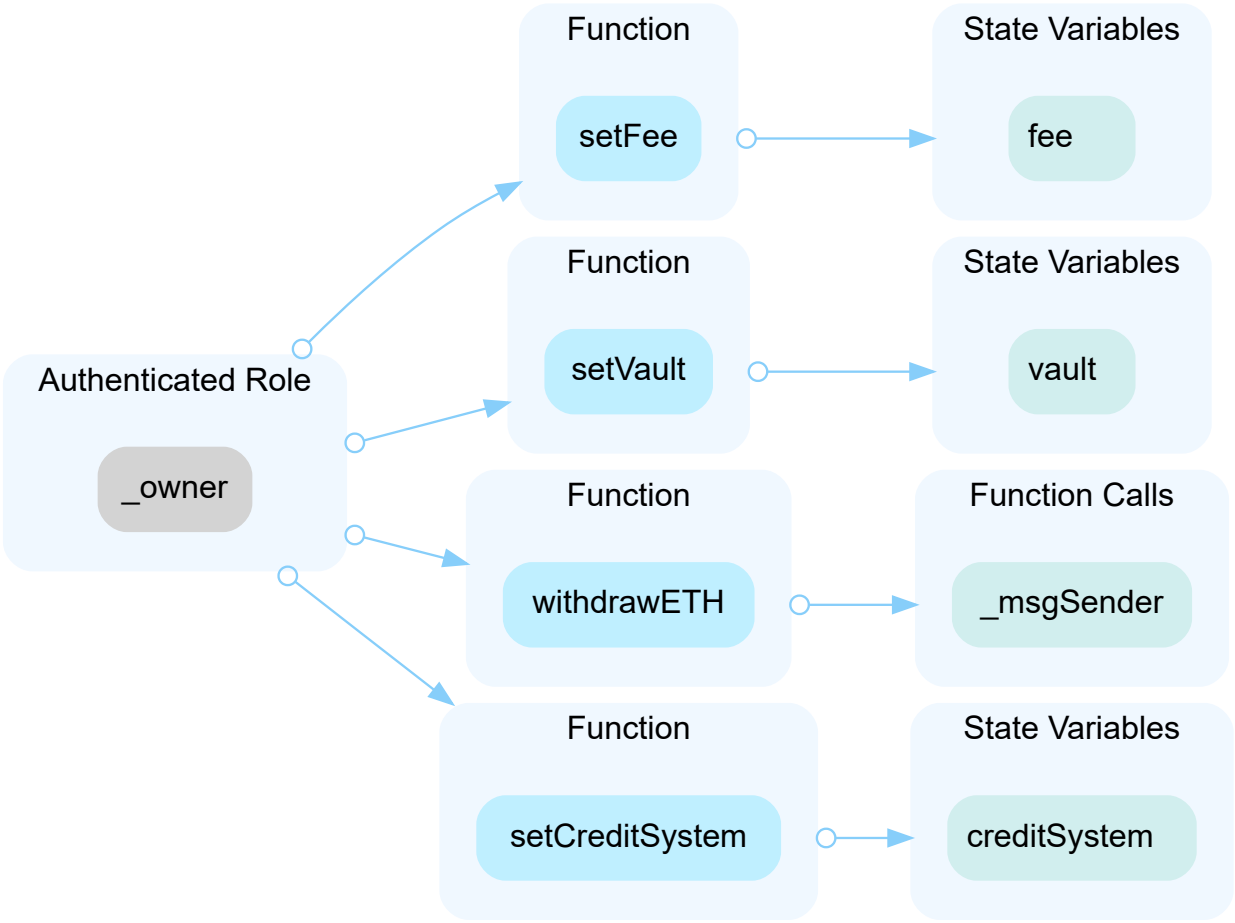
Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/BaseContract.sol: 204, 246, 266; contracts/CCALMainChain.sol: 66, 70, 75, 526~527, 547**	**Acknowledged**

Description

In the contract `BaseContract` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and pause a user's ability to withdraw their tokens from the contract.



In the contract `CCALMainChain` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and withdraw all ETH from the contract.



In the contract `CCALMainChain` the role `AUDITOR_ROLE` has the authority to transfer the amount of ERC20 tokens corresponding to a borrowed NFT at any point. Any compromise to the `AUDITOR_ROLE` account may allow the hacker to take advantage of this authority and delete the `freezeMap` information for all current deposits.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

`[CertiK]`: The `Kyoko` team acknowledges the finding and states they are working toward employing multi-signature wallets to mitigate this finding.

CON-03 | CONFLICTING INITIALIZATION MODIFIERS

Category	Severity	Location	Status
Volatile Code	Major	contracts/BaseContract.sol: 77~78; contracts/CCALMainChain.sol: 52~53; contracts/CCALSubChain.sol: 30~31	Resolved

Description

The initialization function `initialize()` in `CCALMainChain` and `CCALSubChain` call function `initialize()` in parent contract `BaseContract`. All functions use the `initializer` modifier, which leads to a modifier conflict and does not successfully initialize the contracts.

Recommendation

We recommend the client assign non-conflicting modifiers according to the inheritance structure.

Alleviation

`[CertiK]`: The client heeded the recommendation and made the outlined changes in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CON-04 IMPLEMENTATION CONTRACT IS NOT INITIALIZED AUTOMATICALLY

Category	Severity	Location	Status
Control Flow	Medium	contracts/BaseContract.sol: 74~75; contracts/CCALMainChain.sol: 44~45; contracts/CCALSubChain.sol: 24~25	Resolved

Description

When an implementation contract is deployed and the function `initialize()` is called from the proxy contract, it will execute in the context of the proxy contract storage. Hence the state variable `_initialized` and `_initializing` will be 0 and false, respectively. Attackers can directly call the function `initialize()` in the context of the implementation's storage, and feed in malicious inputs which can, in turn, affect the proxy.

Recommendation

We recommend adding the following code to the implementation contract so that the implementation contracts will be initialized automatically.

```
constructor() {
    _disableInitializers();
}
```

Alleviation

`[CertiK]`: The `Kyoko` team heeded the recommendation and made the changes outlined in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CON-05 | THIRD PARTY DEPENDENCY

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BaseContract.sol: 116~117, 247~248; contracts/CCALMainChain.sol: 91~92, 274~275, 320~321; contracts/CCALSubChain.sol: 48~49, 139~140, 169~170	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

`[CertiK]`: The `Kyoko` team acknowledges the finding.

CON-06 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BaseContract.sol: 75~76; contracts/CCALMainChain.sol: 50~51, 54, 55~56, 67~68; contracts/CCALSubChain.sol: 28~29	Acknowledged

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

Recommendation

We recommend adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

`[CertiK]`: The `Kyoko` team acknowledges the finding and opts to make no changes at this time. `[Kyoko]`: Most of the methods that pass in addresses are contract initialization or called by the contract administrator. We will manually control to avoid unexpected errors as much as possible.

CON-07 | CHECK EFFECT INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Logical Issue	Minor	contracts/CCALMainChain.sol: 124~125, 274~275, 369~370, 471~472, 498~499, 517~518; contracts/CCALSubChain.sol: 48~49, 162~163	Resolved

Description

The order of external call/transfer and storage manipulation must follow the check-effect-interaction pattern. This pattern is important for any external calls involving transferring ERC721 tokens from this contract to unknown malicious contracts. In such instances, the `onReceived()` function can be updated to include callback logic that can take advantage of any vulnerable logic.

Recommendation

We recommend the client check that all updates to storage are made before the external call/transfer operation: LINK

Alleviation

`[CertiK]`: The `Kyoko` team has resolved most of the finding in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

The finding is fully resolved as of commit hash 24fab137c99cd0489e9a5f8ddf3f63a51b7c1337.

VLB-01 CALCULATION FOR `totalAmount` VALIDATION MAY BE INCORRECT

Category	Severity	Location	Status
Mathematical Operations, Logical Issue	Minor	contracts/libs/ValidateLogic.sol: 30~31, 40~41	Resolved

Description

The functions `checkDepositPara()` and `checkEditPara()` check the following:

```
totalAmount > (amountPerDay * 1 days / cycle) + minPay
```

From context of the contract logic, it appears that `cycle` is how long the NFTs deposited are to be made available for borrowing. As such, it appears that the check for the `totalAmount` should be :

```
require(totalAmount > (amountPerDay * cycle/ 1 days) + minPay);
```

As an example, if `amountPerDay` is 2 tokens (ignoring decimals), and `cycle` is `10 days`, then the `totalAmount` should be 20 tokens, where the `days` unit has been divided out.

Recommendation

We recommend the client review the information and make adjustments as needed.

Alleviation

`[CertiK]`: The `Kyoko` team confirmed the original calculation was inaccurate and made the changes described above.

The team also made an update to each function that changes the functionality and was not based on the description above. The update is as follows:

- The `totalAmount` is now checked against two outputs:

```
    totalAmount > (amountPerDay * cycle / 1 days) &&
    totalAmount > minPay;
```

instead of the original check that `totalAmount` is greater than the *sum* of the two values.

CCA-06 | COMPARISON TO BOOLEAN CONSTANT

Category	Severity	Location	Status
Coding Style	Informational	contracts/CCALMainChain.sol: 126, 235	Resolved

Description

Boolean constants can be used directly and do not need to be compared to true or false.

```
126             require(tokenInfos[_token].stable == true,
Errors.VL_TOKEN_NOT_MATCH_CREDIT);
```

```
235             require(tokenInfos[_token].stable == true,
Errors.VL_TOKEN_NOT_MATCH_CREDIT);
```

Recommendation

We recommend removing the equality to the boolean constant.

Alleviation

`[CertiK]`: The `Kyoko` team heeded the recommendation and made the changes outlined in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CCA-07 | NFT BORROWING LOGIC AT RISK FOR TOKENS BEING TAKEN

Category	Severity	Location	Status
Control Flow	Informational	contracts/CCALMainChain.sol: 100~101	Partially Resolved

Description

From the context of the contracts, it appears that a user can deposit their NFTs to be loaned out to another user, for a chosen amount of ERC20 tokens, and for a specified number of days. Presumably, the cost for borrowing the NFTs would be less than actually purchasing similar NFTs. Moreover, a borrower may be given a line of credit so they don't actually have to pay ERC20 tokens immediately to borrow. It appears that there is no incentive within the contracts to return the borrowed NFTs. If a user deposits high-value NFTs for loan at a lower price than they are worth, a user can pay the amount of ERC20 tokens to borrow the NFTs (or use their credit line), and never return the NFTs. In such a case, the NFT original holder will only receive the total amount they listed the NFTs to be borrowed for. In this way, borrowers could conceivably steal NFTs for a reduced price (if paying directly), or without paying anything at all (when using credit).

Recommendation

We recommend clarifying if there is a mechanism either within or outside the contracts that would prevent borrowers from taking advantage in the way described above.

Alleviation

`[CertiK]`: The `Kyoko` team notes below that a lender can set their own price in exchange for borrowing a given NFT, and this value can be as high as the lender chooses. It is noted that the team confirms there are conceivable scenarios in which NFTs may not be returned. A lender would account for this possibility by only lending NFTs they are comfortable selling, and to set the borrowing price at an amount for which they would sell the NFT. In this way, the lender can at least recoup the value of the NFT in such a scenario.

`[Kyoko]`:

- The users need to customize the related terms including the deposit amount when conducting the lending process. The setting deposit amount could be more than the NFT value, in which case, the cost for borrowing the NFTs wouldn’ t be less than actually purchasing similar NFTs.
- If the borrower default and fail to return the asset, their initial cash deposit will be liquidated and sent directly to the lender, as compensation.

Therefore, this kind of problem could be avoided to some extent.

CCA-08 | NOT ALL STABLE ERC20 TOKENS HOLD EQUIVALENT VALUE

Category	Severity	Location	Status
Logical Issue	Informational	contracts/CCALMainChain.sol: 126~127, 235~236, 504~505	Resolved

Description

When credit is used, there is a check that the token used in the exchange is a stable token (labeled by the contract owner). Provided this is true, there is a conversion of the value from the original `decimals` of the token used to 18 decimals, so that `creditUsed` values all have the same number of decimals despite the token used in the amount. A hidden assumption is being made here that all stable tokens involved in the exchange will be of approximately equivalent worth (like in the case of 1 USDT being approximately equal to 1 USDC). However, not all stable ERC20 tokens are equivalent in a 1:1 ratio.

Recommendation

We recommend clarifying if the intention is to only use stable coins that have equivalent relative worth. If this is the case, no action needs to be taken. If further changes must be made to accommodate the possibility of non-equivalent stable coins, this finding will be updated to the appropriate severity.

Alleviation

`[Kyoko]`: "We will only use stable coins that have equivalent relative worth in the credit system."

CCL-02 | UNUSED `AccessControl` UTILITY

Category	Severity	Location	Status
Volatile Code	Informational	contracts/CCALSubChain.sol: 21~22	Acknowledged

Description

The contract `CCALSubChain` inherits from `AccessControlUpgradeable`, but the privileged roles are never used in the logic of the contract.

Recommendation

We recommend removing any roles that are never used within the contract.

Alleviation

`[CertiK]` The team acknowledges this finding and opts to make no change at this time.

CCL-03 | `msg.value` SENT BACK TO `msg.sender`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/CCALSubChain.sol: 162~163	Resolved

Description

Function `withdrawAsset()` is labeled `payable`, but if `msg.value` is nonzero, it is sent back to `_msgSender()` without being used. Any calls involving ether make reentrancy possible if `_msgSender()` is a contract. As such, these calls should be avoided if not necessary.

Recommendation

We recommend removing the unnecessary `payable` declaration and the low-level call to `_msgSender()` involving ether.

Alleviation

`[CertiK]`: The `Kyoko` team updated `withdrawAsset()` to exclude the low-level call to `_msgSender()`, removing the immediate risk. Note however, the function is still payable. If a user directly interacts with the contract without the use of the frontend, and they unnecessarily include ether, there is no refund of this payment.

Changes are reflected in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

`[Kyoko]`: The `payable` declaration is necessary for use in the `else` logic. We also provide a front-end interface. When a user calls this function, the front-end program will determine whether to send eth.

CCL-04 | DEPENDENCY THAT `layerZeroEndpoint` IS SECURE

Category	Severity	Location	Status
Control Flow	Informational	contracts/CCALSubChain.sol: 243~244	Acknowledged

Description

Function `lzReceive()` in `CCALSubChain` and in `CCALMainChain` can only be called by `layerZeroEndpoint`. Since the contract logic for `layerZeroEndpoint` is not in scope of the audit, there is an assumption that the functions handling the cross chain interactions take the proper security measures. However, if the functionality of `layerZeroEndpoint` is not secure, these functions could be susceptible to aspects such as replay attacks or falsified information encoded and sent cross-chain.

Recommendation

We recommend the team ensures that falsified information can not be sent through `layerZeroEndpoint` since this is a critical point of security between `CCALMainChain` and `CCALSubChain`.

Alleviation

`[Kyoko]`: We will constantly monitor the safety of this LayerZero endpoint.

CON-08 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/BaseContract.sol: 204, 246; contracts/CCALMainChain.sol: 66, 70, 75, 399, 547; contracts/CCALSubChain.sol: 265	Partially Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

`[CertiK]`: The `Kyoko` team updated the contracts `BaseContract` and `CCALMainChain` to emit as many events as the contract size threshold will allow at this time. The changes can be seen in commit hash 24fab137c99cd0489e9a5f8ddf3f63a51b7c1337.

CON-09 | SHADOWING LOCAL VARIABLE

Category	Severity	Location	Status
Coding Style	Informational	contracts/BaseContract.sol: 47; contracts/CCALMainChain.sol : 81, 101, 164, 199, 268, 284, 334, 434, 448, 483	Resolved

Description

A local variable is shadowing another component defined elsewhere.

```
81      function _borrow(address _borrower, uint _internalId, bool _useCredit) internal {
```

- Local variable `_internalId` in `CCALMainChain._borrow` shadows the variable `_internalId` in `BaseContract`.

```
101         uint _internalId,
```

- Local variable `_internalId` in `CCALMainChain.borrowAsset` shadows the variable `_internalId` in `BaseContract`.

```
164         uint _internalId,
```

- Local variable `_internalId` in `CCALMainChain.estimateCrossChainBorrowFees` shadows the variable `_internalId` in `BaseContract`.

```
199         uint _internalId,
```

- Local variable `_internalId` in `CCALMainChain.borrowOtherChainAsset` shadows the variable `_internalId` in `BaseContract`.

```
268     function repayAsset(uint _internalId) external {
```

- Local variable `_internalId` in `CCALMainChain.repayAsset` shadows the variable `_internalId` in `BaseContract`.

```
284     function _afterRepay(uint _internalId) internal {
```

- Local variable `_internalId` in `CCALMainChain._afterRepay` shadows the variable `_internalId` in `BaseContract`.

```
334     function withdrawToken(uint16 _chainId, uint _internalId, uint _borrowIdx)
external {
```

- Local variable `_internalId` in `CCALMainChain.withdrawToken` shadows the variable `_internalId` in `BaseContract`.

```
434             uint _internalId,
```

- Local variable `_internalId` in `CCALMainChain.handleRepayAsset` shadows the variable `_internalId` in `BaseContract`.

```
448         uint _internalId,
```

- Local variable `_internalId` in `CCALMainChain.updateDataAfterRepay` shadows the variable `_internalId` in `BaseContract`.

```
483             uint _internalId,
```

- Local variable `_internalId` in `CCALMainChain.handleLiquidate` shadows the variable `_internalId` in `BaseContract`.

Recommendation

We recommend removing or renaming the local variable that shadows another definition.

Alleviation

`[CertiK]`: The `Kyoko` team heeded the recommendation and made the changes outlined in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

OPTIMIZATIONS | KYOKO - V

ID	Title	Category	Severity	Status
CCA-04	Improper Usage Of `public` And `external` Type	Gas Optimization	Optimization	Resolved
CCA-05	Unnecessary Use Of SafeMath	Language Specific	Optimization	Resolved

CCA-04 | IMPROPER USAGE OF `public` AND `external` TYPE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/CCALMainChain.sol: 44	Resolved

Description

`public` functions that are never called by the contract could be declared as `external`. `external` functions are more efficient than `public` functions.

Recommendation

Consider using the external attribute for public functions that are never called within the contract.

Alleviation

`[CertiK]`: The `Kyoko` team heeded the recommendation and made the changes outlined in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

CCA-05 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Language Specific	Optimization	contracts/CCALMainChain.sol: 15~16	Resolved

Description

The contract `SafeMathUpgradeable` can be removed because SafeMath is no longer needed starting with Solidity 0.8. The compiler version has built-in overflow checking.

Recommendation

We recommend removing this import for gas optimization.

Alleviation

`[CertiK]`: The `Kyoko` team heeded the recommendation and made the changes outlined in commit 83448f0ea7ed5320cce5f615b408b6469fe7a372.

APPENDIX | KYOKO - V

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Data Flow	Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK' s prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK' s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK' s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER' S OR ANY OTHER PERSON' S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY

SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER' S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK' S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER' S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK' S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.