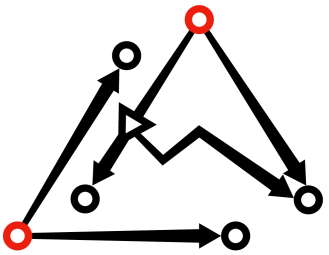

Least Authority

PRIVACY MATTERS

Smart Contracts

Security Audit Report

Cube3

Updated Final Audit Report: 09 October 2023

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - System Design
    - Code Quality
    - Documentation
    - Scope
  - Specific Issues & Suggestions
    - Issue A: _updateIntegrationProtectionStatus Does Not Check Whether Integration Is Pre-registered
    - Issue B: Incorrect Parameter Passed in updateIntegrationProtectionStatus
    - Issue C: Incorrect Cutting of Cube Secure Payload From Message Data
    - Issue D: An Already Protected Integration Can Be Re-Registered
  - Suggestions
    - Suggestion 1: Check EVM Version Before Upgrading to New Solidity Version
    - Suggestion 2: Create Clear Deployment Guidelines for Users
    - Suggestion 3: Restrict getSignatureAuthorityFromInvalidatedSet Function to Only Return Invalidated Signing Authorities
    - Suggestion 4: Minimize or Potentially Remove the Use of _self
    - Suggestion 5: Perform Zero Address Check
    - Suggestion 6: Prevent Variable Shadowing
    - Suggestion 7: Use Correct Error Messages
    - Suggestion 8: Prevent Unnecessary Overflow Check
    - Suggestion 9: Consider Deployer Alternatives
    - Suggestion 10: Use an Array of Struct Instead of Arrays

[About Least Authority]

[Our Methodology]

Overview

Background

Cube3 has requested that Least Authority perform a security audit of their smart contracts. Cube3 provides real-time smart contract security, powered by AI.

Project Dates

- **May 26th - June 20th:** Initial Code Review *(Completed)*
- **June 22:** Delivery of Initial Audit Report *(Completed)*
- **July 5-6:** Verification Review *(Completed)*
- **July 7:** Delivery of Final Audit Report *(Completed)*
- **October 9:** Delivery of updated Final Audit Report *(Completed)*

Review Team

- Mukesh Jaiswal, Security Researcher and Engineer
- Ahmad Jawid Jamiulahmadi, Security Researcher and Engineer
- Steven Jung, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the smart contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Cube3 Protocol:
  https://github.com/cube-web3/cube3-protocol.git
    - Note that after the Initial Audit Report was delivered, the repository was split into two parts:
        - Cube3 Protocol:
          https://github.com/cube-web3/cube3-protocol
        - Cube3-integration:
          https://github.com/cube-web3/cube3-integration

Specifically, we examined the Git revisions for our initial review:

- Initial Commit: `364f7b4aa5693ac9c8cb7a4b70db1bdaaf112805`
- Updated Commit: `76556f50672012ed9eeb5b6286953e3ac3280bd7`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Cube3 Protocol:
  https://github.com/LeastAuthority/cube3_protocol_initial
- Cube3 Protocol:
  https://github.com/LeastAuthority/cube3_protocol

For the verification, we examined the repositories and Git revisions:

- Cube3 Protocol:
  https://github.com/cube-web3/cube3-protocol/tree/develop
  b636ceede25e81190696faa7d50a31c26687c50f

- Cube3 Integration:
  https://github.com/cube-web3/cube3-integration/tree/main
  5a092036f1f441a73eba2df58e195afb3201a739

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- CUBE3.AI:
  https://cube3.ai

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the intended use of the smart contracts or disrupt their execution;
- Vulnerabilities in the smart contracts' code;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a comprehensive review of the design and implementation of the Cube3 smart contracts. The Cube3 Protocol overview in the repository README provides an accurate description of the protocol, and the smart contracts at its core. Our findings from this review consist of implementation issues that could result in unintended behavior.

In addition to the areas of concern listed above, our team reviewed changes proposed by the Cube3 team that were intended to address a potential security Issue. We investigated if an attacker could register a user's integration with the attacker's proxy, and concluded that the Issue being addressed was not a valid security threat. Hence, we recommend that the protective measures implemented to address the Issue be reverted, as our team did not identify any security vulnerabilities resulting from the original

implementation (Suggestion 4). Our team also considered solutions to a deployment Issue preventing the smart contracts from being deployed to the same address on all target chains (Suggestion 9).

System Design

Our team found that security has been taken into consideration in the design of Cube3 as demonstrated by the safeguards implemented to recover a compromised server by invalidating the existing key pairs, and reassigning fresh keys. However, the server performs security-critical functionality, and is a single point of failure in the system. We recommend a comprehensive review of the system that includes all components. In this review, our team investigated the design of the Cube3 smart contracts for security vulnerabilities and found missing checks that could lead to unintended behavior (Issue A, Issue D).

Code Quality

Our team performed a manual review of the Cube3 smart contracts and found the code to be well-organized and adhering to best practice. However, we identified instances of implementation errors that could lead to unintended behavior (Issue B, Issue C). Our team also found opportunities for efficiency improvements (Suggestion 7, Suggestion 8, Suggestion 10).

Tests

Sufficient test coverage has been implemented to test for implementation errors that could lead to security vulnerabilities.

Documentation

Our team found that the documentation provided for this review was sufficient. We recommend that the deployment documentation be improved (Suggestion 2).

Code Comments

Sufficient code comments adhering to NatSpec guidelines describe security-critical functions and components.

Scope

Our team found that the scope of this review included all on-chain security-critical components. During the review, our team was asked to update the scope of the review to include changes to the smart contract. As a result, our team initiated the review with a commit hash of the repository in scope, and subsequently integrated a later commit of the repository into the review. Both commits are referenced above.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: _updateIntegrationProtectionStatus Does Not Check Whether Integration Is Pre-Registered	Resolved
Issue B: Incorrect Parameter Passed in updateIntegrationProtectionStatus	Resolved

Issue C: Incorrect Cutting of Cube Secure Payload From Message Data	Resolved
Issue D: An Already Protected Integration Can Be Re-Registered	Resolved
Suggestion 1: Check EVM Version Before Upgrading to New Solidity Version	Resolved
Suggestion 2: Create Clear Deployment Guidelines for Users	Acknowledged
Suggestion 3: Restrict getSignatureAuthorityFromInvalidatedSet Function to Only Return Invalidated Signing Authorities	Resolved
Suggestion 4: Minimize or Potentially Remove the Use of _self	Acknowledged
Suggestion 5: Perform Zero Address Check	Resolved
Suggestion 6: Prevent Variable Shadowing	Resolved
Suggestion 7: Use Correct Error Messages	Resolved
Suggestion 8: Prevent Unnecessary Overflow Check	Resolved
Suggestion 9: Consider Deployer Alternatives	Resolved
Suggestion 10: Use an Array of Struct Instead of Arrays	Resolved

Issue A: _updateIntegrationProtectionStatus Does Not Check Whether Integration Is Pre-registered

Location

`contracts/Cube3GateKeeper.sol#LL60C14-L60C48`

Synopsis

The `_updateIntegrationProtectionStatus` function does not check if the integration is `preRegistered`.

Impact

`_updateIntegrationProtectionStatus` can update the integration status that is not pre-registered through the protocol.

Remediation

We recommend adding checks to verify that a given integration is pre-registered with the protocol.

Status

The Cube3 team has added two different statuses for the integration state, which are checked by the functions `preRegisterAsIntegration` and `complete2StepIntegrationRegistration` to ensure that the integration is registered with the protocol before its authorized statuses are updated through the function `_updateIngrationProtectionStatus`.

Verification

Resolved.

Issue B: Incorrect Parameter Passed in updateIntegrationProtectionStatus

Location

contracts/Cube3RouterLogic.sol#L275

contracts/Cube3RouterLogic.sol#L286

Synopsis

The `_updateIntegrationProtectionStatus` function expects `proxy` as the first parameter and `implementation` as the second. However, in the functions `bypassIntegrationProtectionStatus` and `revokeIntegrationProtectionStatus`, the first parameter is passed as `implementation` and the second as `proxy`.

Impact

The `mapping _integrationProtectionStatus` function will not be updated as intended. Additionally, it will emit an event with incorrect parameters, which can affect off-chain services.

Remediation

We recommend passing the correct parameter to `_updateIntegrationProtectionStatus`.

Status

The Cube3 team acknowledged the finding but stated that it is no longer relevant. Therefore, we consider this Issue resolved.

Verification

Resolved.

Issue C: Incorrect Cutting of Cube Secure Payload From Message Data

Location

contracts/Cube3Integration.sol#L83-L88

Synopsis

`cube3SecurePayload` is intended to be added as the last parameter in any Cube3-protected function. In order to extract the `calldata` of the function being called, `cube3SecurePayload` should be cut from `msg.data`. In the current implementation, it is assumed that the structure of `cube3SecurePayload` in the `msg.data` is as follows:

```
payload offset <32> | payload length <32> | payload
```

Therefore, to cut `cube3SecurePayload`, it is assumed that 64 additional bytes (`offset <32> + length <32>`) should be cut from `msg.data` as well as `payload`. However, if there are other dynamic parameters preceding `cube3SecurePayload`, the payload offset will not precede the length but may be encoded elsewhere in `msg.data`. Hence, incorrect data will be cut from the end of `msg.data`.

Impact

An incorrectly cut `cube3SecurePayload` would result in unintended behavior.

Preconditions

This Issue is likely if dynamic parameters precede `cube3SecurePayload`.

Remediation

We recommend changing the implementation to take into consideration the dynamic parameters preceding the `cube3SecurePayload`.

Status

The Cube3 team has resolved this Issue by cutting only 32 additional bytes (i.e. the length) from the end of `msg.data`, which will always precede the actual payload. Additionally, the team has modified the generation of payload off-chain to match the implemented fix on-chain.

Verification

Resolved.

Issue D: An Already Protected Integration Can Be Re-Registered

Location

[contracts/Cube3GateKeeper.sol#L122-L125]

Synopsis

The function `preRegisterAsIntegration` can pre-register an already-registered (protected) integration.

Impact

Re-registering a protected integration is unintended behavior, which might result in replacing an already protected integration.

Remediation

We recommend checking if an integration is already registered before pre-registering it to avoid replacing an already protected integration.

Status

The Cube3 team has implemented the remediation as recommended.

Verification

Resolved.

Suggestions

Suggestion 1: Check EVM Version Before Upgrading to New Solidity Version

Synopsis

In the Solidity 0.8.20 release, the compiler automatically upgrades from the EVM version to Shanghai. As a result, the generated bytecode will include the PUSH0 opcode. Hence, if other chains do not support the PUSH0 opcode, the deployment will fail.

Mitigation

Since the contracts are deployed on different EVM-compatible chains, we recommend ensuring that the appropriate EVM version is implemented.

Status

The Cube3 team has added a warning to the README.md file under the #Deployment section.

Verification

Resolved.

Suggestion 2: Create Clear Deployment Guidelines for Users

Synopsis

The `Cube3Integration` smart contract is intended to be used only by singleton contracts. However, it can be extended and utilized by upgradeable contracts since there is no check to disallow it. A user could mistakenly extend the wrong smart contract for the purpose intended, resulting in an unfavorable user experience.

Mitigation

Although it might not be possible to completely prevent it, we recommend reducing this occurrence as much as possible by creating very clear set up documentation, such as a deployment guideline.

Status

The Cube3 team acknowledged that documentation can and should be updated to reflect the final state of the protocol and code, as creating clear documentation and guides is a prerequisite for launching the product.

Verification

The Cube3 acknowledged the suggestion, however at the time of the verification it remained unresolved.

Suggestion 3: Restrict getSignatureAuthorityFromInvalidatedSet Function to Only Return Invalidated Signing Authorities

Location

`contracts/Cube3RegistryLogic.sol#L230-L235`

Synopsis

It is possible to obtain valid signing authorities using the aforementioned function. However, the name of the function and related comments in the interface imply that only invalidated signing authorities can be obtained.

Mitigation

We recommend restricting the `getSignatureAuthorityFromInvalidatedSet` function to only return invalidated signing authorities.

Status

The Cube3 team has updated the function to return whether the integration's signing authority that is returned for a given nonce is active, instead of (incorrectly) returning from an invalidated set.

Verification

Resolved.

Suggestion 4: Minimize or Potentially Remove the Use of _self

Synopsis

The Cube3 team asked our team to investigate a malicious `delegatecall` scenario, which involves an attacker performing a `delegatecall` into a user's integration.

Integration smart contracts use an immutable state variable `_self` to protect against malicious delegate calls. Assuming that an attacker could phish users into calling the attacker's proxy, it can then be delegate-called into the customer's integration. This scenario would result in an attacker successfully being able to bypass the `cube3Protected` modifier.

However, in our review, our team found that implementing the `_self` is unnecessary. Executing a `delegatecall` from an attacker's proxy (i.e. the context) into a customer's integration (implementation) would not impose a security risk, as the implementation does not hold context in this particular scenario, and all state readings and changes would occur in the attacker's proxy. Although external calls from the implementation to other contracts in this chain of calls appear to incur additional risks, a close investigation shows that the calls would result in `msg.sender` becoming the attacker, which, in turn, would invalidate the attack. The malicious actor can then only perform an exploit using `tx.origin`.

In such a case, the attacker would directly call the potentially exploitable function instead of first performing a `delegatecall` into the user's integration since the chain of calls would be shorter.

Therefore, in order to perform an exploit, an attacker would have to compromise the user's context (which could be a proxy) instead of an implementation.

Mitigation

We recommend minimizing or potentially removing the use of `_self` to reduce the complexity and gas cost of the protocol.

Status

The Cube3 team acknowledged the suggestion and noted that the use of `_self` is required to differentiate between proxies (that can be either legitimate or malicious) and different implementation contracts.

Verification

The Cube3 acknowledged the suggestion, however at the time of the verification it remained unresolved..

Suggestion 5: Perform Zero Address Check

Location

`contracts/Cube3SignatureModule.sol#L35`

Synopsis

In the constructor of the `Cube3SignatureModule` smart contract, there is no zero address check validating the correctness of the `cube3RegistryProxy` parameter, thereby preventing an incorrectly set `cube3registry` value.

Mitigation

We recommend checking the referenced parameter against zero address.

Status

The Cube3 team has implemented the mitigation as recommended.

Verification

Resolved.

Suggestion 6: Prevent Variable Shadowing

Location

`contracts/upgradeable/Cube3IntegrationUpgradeable.sol#L61`

`contracts/upgradeable/Cube3IntegrationUpgradeable.sol#L72`

`contracts/upgradeable/SecurityAdmin2StepUpgradeable.sol#L54-L56`

Synopsis

The `securityAdmin` parameter implemented in the `initialize` functions inside the `Cube3IntegrationUpgradeable` smart contract shadows the `securityAdmin getter` function inside the `SecurityAdmin2StepUpgradeable` smart contract. Variable shadowing could lead to unexpected behavior.

Mitigation

We recommend either changing the parameter name `securityAdmin` inside the `initialize` functions in the `Cube3IntegrationUpgradeable` smart contract or changing the `securityAdmin getter` function name inside the `SecurityAdmin2StepUpgradeable` smart contract.

Status

The Cube3 team has implemented the mitigation as recommended.

Verification

Resolved.

Suggestion 7: Use Correct Error Messages

Location

`contracts/Cube3GateKeeper.sol#L113`

Synopsis

This `revert` message is inaccurate. It states that the status is not active, while the error case is unregistered or revoked. Incorrect error messages make it difficult to analyze the cause of the error.

Mitigation

We recommend using correct `revert` error messages.

Status

The Cube3 team has implemented the mitigation as recommended.

Verification

Resolved.

Suggestion 8: Prevent Unnecessary Overflow Check

Location

`contracts/Cube3RegistryLogic.sol#L104`

Synopsis

This calculation can be performed with unchecked math to save gas because the count number cannot overflow.

Mitigation

We recommend using unchecked math operations to save gas on operations that do not require safeguards.

Status

The Cube3 team has implemented the mitigation as recommended.

Verification

Resolved.

Suggestion 9: Consider Deployer Alternatives

Location

Examples (non-exhaustive):

Hardhat Deployer: `create2deployer`

Synopsis

The Cube3 team asked our team to investigate deployment options given that it is a business requirement to deploy the smart contracts to the same address on each of the blockchains they are deployed to.

The currently used Foundry deployer "wolf of wall street" is not compatible with Arbitrum and Canto. Therefore, since the deployer does not support these networks, a custom deployer is needed to deploy the `gatekeeper` and `router` smart contracts on all target chains.

Mitigation

Our team investigated a mitigation offered by the Cube3 team, which includes writing a deployment script to use the appropriate deployer as determined by the chain ID, and confirmed this solution would successfully resolve the suggestion. Additionally, our team can recommend the following two alternatives:

1. Building a custom deployer that has the required capabilities; or
2. Using existing deployment tools available on the Hardhat framework.

Status

The Cube3 team has added the `ProtocolContractsByChain` library for storing deployed protocol addresses, and is conducting further research (as noted here and here).

Verification

Resolved.

Suggestion 10: Use an Array of Struct Instead of Arrays

Location

contracts/Cube3GateKeeper.sol#L173-L185

Synopsis

The function `batchUpdateIntegrationProtectionStatuses` does not check if arrays `integrationOrProxies`, `integrationOrImplementations`, and `statuses` are of the same length. Consequently, a `revert` may occur if all three parameters are not of the same length due to an array index out of bounds exception, which would result in an unnecessary consumption of gas.

Remediation

In the `Cube3RouterLogic` contract, when calling the aforementioned function, the lengths of `integrationOrProxies` and `integrationOrImplementations` are checked, but the length of `statuses` is not checked. Since `batchUpdateIntegrationProtectionStatuses` is an external function, we recommend validating all inputs inside the function itself.

Alternatively, we recommend using an array of `struct` instead of arrays.

Status

The Cube3 team acknowledged the finding but stated that it is no longer relevant. Therefore, we consider this suggestion resolved.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.