Blaize.Security

BLOCKUS

BLOCKUS
SMART CONTRACT AUDIT

March 11th, 2024 / V. 1.0

Table of Contents

Executive Summary

During the audit, we examined the security of smart contracts for the **Blockus** protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **Blockus** smart contracts conducted between **March 8th, 2024 and March 11th, 2024.**

Blaize Security reviewed the Move (Sui) implementation of Ambrus token and NFT from the Blockus protocol, Auditors checked, that all components correspond to best practices relative to FT and NFT development and to have secure asset management functionality.
During the audit, the security team conducted several checks, including the following activities:

- Auditors reviewed all token manipulation capabilities and checked them during simulation testing as well;
- Review of object types and capabilities (mint/burn rules for FT token; mint, burn, setters, and create new minters for NFT);
- review of transfer capabilities of capability objects;
- review of display implementation for NFT;
- review of returned types and function visibility modifier;
- reviewed token and NFT against best practices and standards accepted within the Sui ecosystem.

The security team concluded that the audited components implement FT and NFT standards correctly and correspond to Sui's best practices. Although the functionality is basic and built via the standard Move (Sui) approaches, auditors ahould note the absence of native tests. Nevertheless, auditors verified the functionality via their own set of Move tests prepared during the testing stage. The protocol implements the necessary security standards.

Auditing strategy and Techniques applied/Procedure

Blaize.Security auditors start the audit by developing an auditing strategy - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

MANUAL AUDIT STAGE

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control review, roles structure, analysis of user and admin capabilities and behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

FOR ADVANCED COMPONENTS:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

TESTING STAGE:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

Audit Rating

Score: **10**/10

	RATING
Security	10
Logic optimization	10
Code quality	10
Testing suite	5
Documentation	N/A

Security: General mark for the security of the protocol.
The main mark for the audit qualification.

Logic optimization: Evaluation of how optimal the implementation is, including presence of extra/unused code, uncovered/extra cases, gas (or its analog) optimization, memory management optimization, etc
Code quality: Evaluation of best practices followed, code readability, structure and convenience of further development
Testing suite: Availability of the native tests suite, level of logic coverage, checks of critical areas being covered.
Documentation: Availability and quality of the documentation, coverage of core functionality and user flows: whitepaper, gitbook, readme, specs, natspec, comments in the code and other possible forms of documentation.

SECURITY RATING CALCULATION

Approximate weight of unresolved issues.
Critical: -3 points
High: -2 points
Medium: -1 points
Low: -0.5 points
Informational: -0.1 point

Note: additional concerns, violated checklist items (including standard vulnerabilities), and verified backdoors may influence the final mark and weight of certain issues.

Starting with a perfect score of 100:

Critical issues: 0 issues (0 resolved): 0 points deducted

High issues: 0 issues (0 resolved): 0 points deducted

Medium issues: 0 issues (0 resolved): 0 points deducted

Low issues: 1 issue (1 resoled): 0 points deducted

Informational issues: 4 issues (3 resolved, 1 verified): 0 points deducted

Other: 0 points deducted

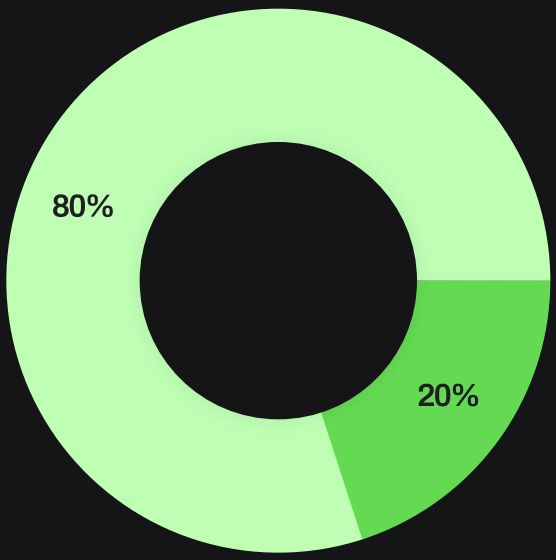
Native tests are absent, thus the testing mark is 5. It indicates that tests for the basic functionality implemented in the protocol are not required but recommended

Security rating = 10 - 0 = 10

Technical Summary

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

- Critical
- High
- Medium
- Low
- Info



The table below shows the number of the detected issues and their severity. A total of 5 problems were found. 5 issues were fixed or verified by the Blockus team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	1	1
Info	4	4

SEVERITY DEFINITION

CRITICAL

The system contains several issues ranked as very seriousand dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

HIGH

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

MEDIUM

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

LOW

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

INFO

The issue has no impact on the contract's ability to operate, yet is relevant for best practices. Or this status can be assigned to the issues related to the suspicious activity or substandard business logic decisions which cannot be classified without the comments from the team (and can be re-classified on the later audit stages).

Issues reviewed by the team can get the next statuses:
Resolved: issue is resolved by an appropriate patch or changes in the business logic
Verified: the team provided sufficient evidences that the issue describes desired behavior
Unresolved: neither path nor comments provided by the team, or they are not sufficient to resolve the issue
Acknowledged: the team accepts the misbehavior and connected risks

Audit Scope

Language/Technology: **Move**

Blockchain: **Sui**

The scope of the project includes:

- nft\sources\nft_module.move
- ambrus_coin\sources\token.move

Repository: https://github.com/BlockUs0/sui

The source code of the smart contract was taken from the branch: main

Initial commit:

- bc67084b2fcf84755db3ebc38cf8bae8b159b38c

Final commit:

- d1b40a86173439a7717c604fc930406b11b4166b

Protocol overview

DESCRIPTION

The protocol contains two contracts on the Sui blockchain:

- ambrus_in_game_currency (Ambrus coin)
- blockus_nfts (nft).

Ambrus coin is an implementation of the Sui token FALLEN_ARENA_SHARD, under the ticker Ambrus with the symbol SHARD. The contract represents basic functionality for minting and burning tokens for the owner.
The second contract is a basic mint/burn contract for NFTs named BlockusNft, with additional management of the metadata fields. Privileged users appointed by the Admin perform management.

ROLES AND RESPONSIBILITIES

Ambrus coin
1. Publisher (deployer):
- Receives the associated UpgradeCap and TreasuryCap objects when the contract is deployed.
- Can initiate upgrades of the contract since it owns UpgradeCap.

2. Admin:
- represented by game_studio_address defined in Move.toml
- Receives the associated TreasuryCap object when the contract is deployed.
- Can run mint and burn since it owns TreasuryCap.

3. Regular user:
- all regular FT operations (transfer, hold, etc)

ROLES AND RESPONSIBILITIES

NFT

1. Publisher (deployer):

- Receives the associated Display, Publisher, and UpgradeCap objects when the contract is deployed.
- Can initiate upgrades of the contract since it owns UpgradeCap.

2. Admin:

- represented by blockus_shinami_wallet defined in Move.toml
- Receives the associated MintCap and AdminCap objects when the contract is deployed.
- Can create new minters.
- Can manipulate the NFT object - mint, burn, set fields.

3. MintCap owner (minter):

- User who owns the MintCap object
- Can manipulate the supply of the NFT object (mint, burn), set all metadata fields.

4. Regular user:

- all regular NFT operations (transfer, hold, etc)

LIST OF VALUABLE ASSETS

1.Ambrus coin:

Standard Move implementation of the FT token.

2. NFT:

Standard implementation of the NFT

Both contracts are not designed to keep any other assets (or objects in Move/Sui paradigm).

CONFIGURATION AND SETTINGS

Ambrus coin

1. Roles Assignment:

- game_studio_address - address of the coin admin (defined in Move.toml). Holds the TreasuryCap object
- Incorrect assignment can potentially lead to unauthorized access to critical functions such as mint, because the total supply of which is limited to a maximum value of u64, preventing further minting of coins.

2. Coin constants:

- COIN_DECIMALS - set to 0 for the token.
- COIN_SYMBOL - “Shard”
- COIN_NAME - “Ambrus”
- COIN_DESCRIPTION and COIN_IMAGE_URL - associated metadata

NFT

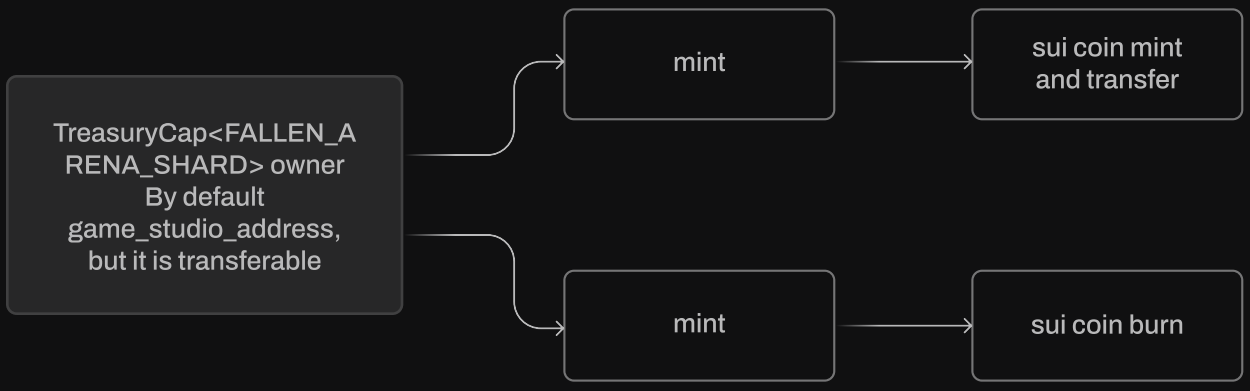
1. Roles Assignment:

- blockus_shinami_wallet - address of the coin admin (defined in Move.toml). Holds the AdminCap and MintCap objects
- Incorrect assignment can lead to unauthorized access to critical functions such as create_minter, allowing MintCap, and consequently the privileges to create and manipulate the nft

DEPLOYMENT

The deploy scripts are the standard deploy command for Sui and have also been duplicated by the Blockus team in the README for each of the projects.

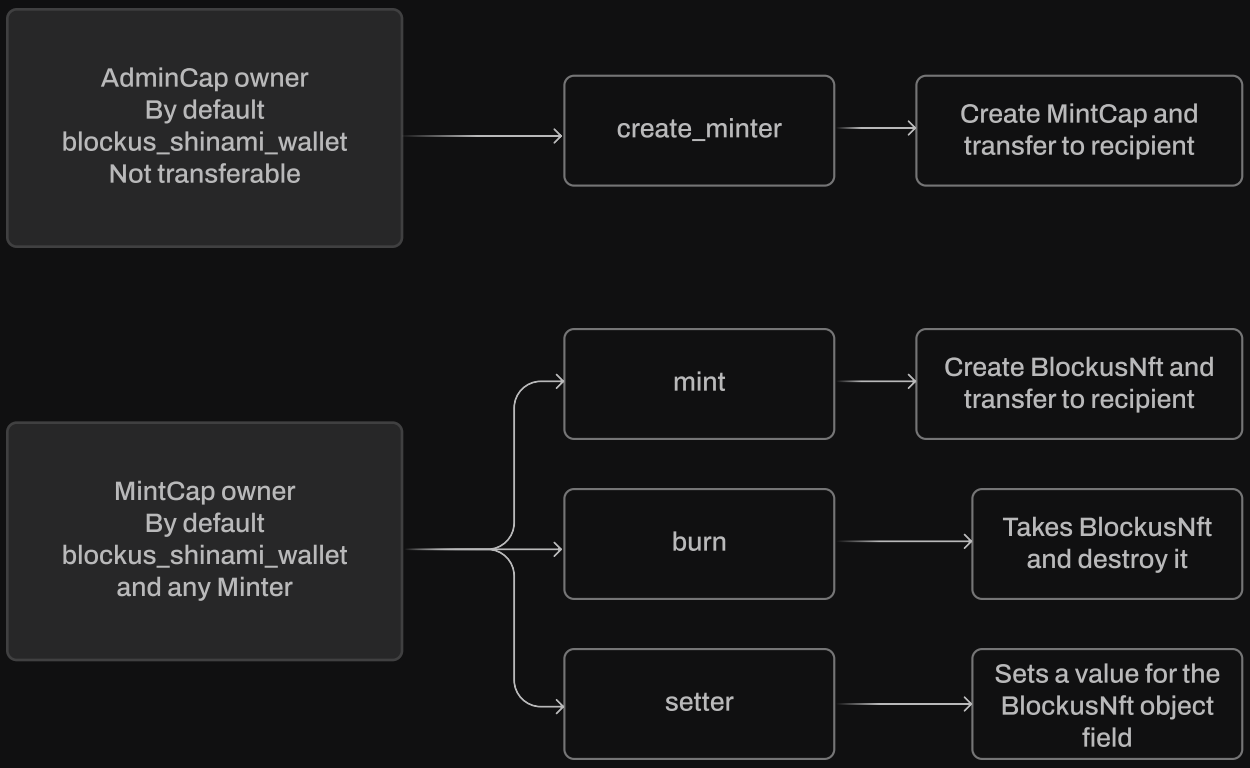
AMBRUS COIN



Main flows with User

Roles		
Deployer	game_studio_address	User
Deploys the package and receives the `UpgradeCap` object	When the contract is deployed, this address receives a `TreasuryCap<FALLEN_ARENA_SHARD>` object. Performs package management such as `mint` and `burn`	Third party who may be the recipient of the token `Coin<FALLEN_ARENA_SHARD>`

NFT



Contract roles

Roles			
Deployer	blockus_shinami_wallet	User	Minter
Deploys the package and receives the `UpgradeCap`, `Publisher`, `Display` objects	When the contract is deployed, this address receives `MintCap` and `AdminCap` objects	Third party who may be the recipient of the `BlockusNft` object	Third party who may be the recipient of the `MintCap` object from `create_minter` function called by `AdminCap` owner

NFT

Protocol setters

Function Name	Function Description
set_name	Sets a new name for the BlockusNft object
set_description	Sets a new description for the BlockusNft object
set_image_url	Sets a new image url for the BlockusNft object
set_link	Sets a new link for the BlockusNft object
set_thumbnail_url	Sets a new thumbnail url for the BlockusNft object
set_project_url	Sets a new project url for the BlockusNft object
set_creator	Sets a new creator for the BlockusNft object
set_custom_attributes	Sets a new custom attributes for the BlockusNft object
set_external_id	Sets a new external id for the BlockusNft object

Complete Analysis

STANDARD CHECKLIST / VULNERABLE AREAS

Storage structure and data modification flow	Pass
Access control structure, roles existing in the system	Pass
Public interface and restrictions based on the roles system	Pass
General Denial Of Service (DOS)	N/A
Entropy Illusion (Lack of Randomness)	N/A
Order-dependency and time-dependency of operations	Pass
Accuracy loss, incorrect math/formulas other violated operations with numbers	Pass
Validation of function parameters, inputs validation	Pass
Asset management, funds flow and assets conversions	N/A
Signatures replay and multisig schemes security	N/A
Asset Security (backdoors connected to underlying assets)	Pass
General Denial Of Service (DOS)	Pass
Global settings mis-using, incorrect default values	Pass
Violated communication between components/modules, broken co-dependencies	N/A
3rd party dependencies, used libraries and packages structure	Pass
General code structure checks and correspondence to best practices	Pass
Language-specific checks	Pass

LOW-1 Resolved

NO VALIDATION FOR THE MINTED AMOUNT

token.move: mint().
Sui allows you to create token objects with zero amount, but owning such a token is meaningless. Thus it is better to restrict the default ability to mint such tokens.

RECOMMENDATION:

Before the mint_and_transfer call, add an assertion that the amount is greater than zero:
assert!(amount > 0, EInsufficientAmount);

INFO-1 Resolved

NO VALIDATION FOR THE MINTED AMOUNT

ambrus_coin/Move.toml, nft/Move.toml.
The Sui dependency currently relies on the "framework/testnet" revision. Over time, the data in the "framework/testnet" may change, and it will be possible to face an on-chain code mismatch error.

RECOMMENDATION:

Replace "framework/testnet" in rev Sui dependency with a specific mainnet commit.

INFO-2 Resolved

LACK OF VERSIONING

ambrus_coin/Move.toml, nft/Move.toml.
The lack of a version of the contract could cause havoc in the future if there are more versions of the contract.

RECOMMENDATION:

Add versioning to Move.toml according to `semver`
version = "0.1.0"

INFO-3 Resolved

LACK OF EVENTS

token.move: mint(), burn()
nft_module.move: create_minter(), mint(), burn().
To keep track of historical changes in token creation or burning, as well as the creation of new gravitated users (as happens in the `create_minter()` function), it is recommended to issue events on each such function call.

RECOMMENDATION:

Add the corresponding `MintEvent`, `BurnEvent`, and `MinterEvent` events.

POST-AUDIT:

In 2 iterations, the Blockus team added event for all main functions and metadata setters

INFO-4 Verified

ADMIN CAN EDIT NFT METADATA

nft_module.move: set_name(), set_image_url(), set_link(), set_thumbnail_url(), set_project_url(), set_creator(), set_custom_attributes(), set_external_id()

The standard practice for working with NFTs does not allow privileged users to modify all metadata fields. Verification from the Blockus team is required to confirm that the metadata change is the desired functionality.

RECOMMENDATION:

Verify that the metadata correction functionality is required for the protocol and/or add the context of this functionality within the acceptable user flows of the protocol

POST-AUDIT:

The Blockus team confirmed the need for this functionality and verified it.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

ambrus_coin

✓ test_workflow

This test simulates the following case:

1) game_studio_address (aka admin) deploys and initializes a contract and creates a currency of the type FALLEN_ARENA_SHARD with preset parameters.

2) The admin mints coins twice (with the amount of `mint_amount`)

3) The admin burns the coins in the amount of mint_amount

Note: all other functionality is implemented via the standard Sui approach

nft

✓ test_workflow

This test simulates the following case:

1) blockus_shinami_wallet (who is also the admin) deploys and initializes the contract, and creates AdminCap and MintCap objects. Objects are sent to the blockus_shinami_wallet address.

2) The AdminCap owner creates a MintCap for the minter and for the authorized user.

3) The minter mints an NFT with empty fields for the user.

4) In turn, after receiving the NFT, the authorized user with MintCap object changes the values of all fields at their discretion.

Note: all other functionality is implemented via the standard Sui approach

NATIVE TESTS OVERVIEW

Project contains no native tests. Despite, since most of the functionality is based on the standard Sui approaches, auditors still recommend having native tests for the extra logic, despite its simplicity.

Disclaimer

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.