Tengoku Senso

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	GameFi NFT
Timeline	2023-07-06 through 2023-07-07
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• AkshaySharma96/TGK-Smart-Contracts-Audit

Documentation quality	High
Test quality	Medium
Total Findings	8 **Fixed: 8**
High severity findings ⓘ	4 **Fixed: 4**
Medium severity findings ⓘ	0
Low severity findings ⓘ	3

	#68f99d3
Auditors	• Cameron Biniamow Auditing Engineer • Shih-Hung Wang Auditing Engineer • Valerian Callens Senior Auditing Engineer

	Fixed: 3
Undetermined severity findings ⓘ	0
Informational findings ⓘ	1 Fixed: 1

Summary of Findings

Tengoku Senso is a GameFi NFT platform built on Ethereum. The audit team found high-severity issues that show that the system is not functioning as intended. For example, NFT minting lacks access control, and the `TGKMainContract` contract, which is supposed to receive NFTs, actually cannot receive them via safe transfers. These high-severity issues could have been detected with basic test suites, and their presence signifies the problem of insufficient test suites. Indeed, when the audit team inspected the project, the test suite was absent, and at the same time, there was almost no documentation. Moreover, one of the contracts does not even compile; hence there is no way of running and testing it.

Regarding the project quality, since there is almost no documentation and tests, the auditors followed a best effort-approach in attempting to identify potential combinations of inputs and outputs that could lead to unexpected behavior. Our team frequently interacted with the Tengoku Senso team to clarify code and expected behavior. Their active engagement in answering our questions was crucial and greatly assisted in completing the audit. Still, the overall confidence is low given that the project does not compile and no tests accompany such an audit. An audit does not replace the need for a high-quality test suite, and we strongly recommend fixing the compilation issue and testing all the happy and unhappy paths of the system.

The system is highly centralized and could suffer from a significant exploit if the owner of the contracts is compromised. Further, the Tengoku Senso team will utilize out-of-scope off-chain mechanisms to determine the receiving address and amount of tokens (native, ERC20, and ERC721) to transfer from the contract `TGKMainContract` . As a result, we want to highlight the importance of following best practices for Key Management of admin addresses, as well as carefully assessing the current level of IT risks associated with the off-chain components of the system and implementing mitigating measures accordingly.

Fix review

The Tengoku Senso team addressed all issues. The Tengoku Senso team clearly documented their system in a `README` file and added a test suite. We strongly suggest making that documentation easily available to the end users in the front end of the application instead of keeping it in the `README` file. Also, we would like to highlight the commitment of the Tengoku Senso team to address all issues.

ID	DESCRIPTION	SEVERITY	STATUS
TENGO-1	Anyone Can Mint an Arbitrary Number of NFTs	● High	Fixed
TENGO-2	The Contract `TGKMainContract` Is Not Able to Receive NFTs via Safe Transfers	● High	Fixed
TENGO-3	The Contract `TGKNFTContract` Does Not Compile	● High	Fixed
TENGO-4	Missing Test Suite and Limited Documentation Makes It Hard to Assess that the System Works as Expected	● High	Fixed
TENGO-5	Privileged Roles and Ownership	● Low	Fixed
TENGO-6	Potential Use of Compiler Version with Known Bugs	● Low	Fixed
TENGO-7	Missing Input Validation	● Low	Fixed
TENGO-8	Using `transfer()` for ETH May Fail in The Future	● Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following

    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

The scope included two contracts: `TGKMainContract.sol` and `TGKNFTContract.sol` .

Findings

TENGO-1 Anyone Can Mint an Arbitrary Number of NFTs

● High ⓘ Fixed

✓ **Update**

The client implemented the recommended actions.

✓ **Update**

Marked as "Fixed" by the client. Addressed in: `97625a33a9dd4f4cbce659c86c2e59359caf1359` .

File(s) affected: `TGKNFTContract.sol`

Description: The function `TGKNFTContract.mintNFT()` does not restrict which addresses can mint NFTs since this function is public and has no access control. Per the client's explanation, only the owner of `TGKNFTContract` should be able to call `mintNFT()` .

Recommendation: Consider using the `onlyOwner` modifier in `TGKNFTContract` and adding the modifier to the `mintNFT()` function.

TENGO-2

The Contract `TGKMainContract` Is Not Able to Receive NFTs via Safe Transfers

● High ⓘ Fixed

Update

The client implemented the recommended actions.

Update

Marked as "Fixed" by the client. Addressed in: `c47765595a3fc73626a0792b2460542e00935913` .

File(s) affected: `TGKMainContract.sol`

Description: The contract `TGKMainContract` is expected to receive ERC721 NFT tokens. These tokens can usually be received via different functions. Some of them, like `safeMint()` or `safeTransferFrom()` first check that the receiver address, if it is a contract, implements the function `IERC721Receiver.onERC721Received()` described here. If not, the NFT can remain stuck in the receiver contract. As the contract `TGKMainContract` does not implement the function `IERC721Receiver.onERC721Received()`, it will not be able to receive NFT tokens via any kind of safe transfer.

Recommendation: Consider implementing the function `IERC721Receiver.onERC721Received()` in the contract `TGKMainContract`.

TENGO-3 The Contract `TGKNFTContract` Does Not Compile

● High ⓘ Fixed

Update
It is now possible to compile the contracts (commit `1f8ba861cc5895ca46920a930967a749abae7678`).

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> In our Hardhat configuration file, we set the Solidity compiler version to 0.8.18. By including this configuration in the Hardhat file, we ensure that the contracts are compiled using the Solidity version 0.8.18. This version will be used when compiling and deploying the contracts using Hardhat. If we make the changes suggested by you, we get the following error: Invalid contract specified in override list: "ERC721URIStorage".

File(s) affected: `TGKNFTContract.sol`

Description: The contract `TGKNFTContract` does not compile due to incorrect inheritance declarations and gives two errors:

```
1. Error (4327): Function needs to specify overridden contract "ERC721URIStorage". => needs to
override ERC721URIStorage (since it is explicitly inherited)
```

```
2. Error (2353): Invalid contract specified in override list: "ERC721". => should not override
ERC721 (since it is not explicitly inherited)
```

In detail, the contract `TGKNFTContract` does not inherit `ERC721`, and the function `supportsInterface()` is missing `ERC721URIStorage` from its override list.

Recommendation:
Consider:
1. explicitly inheriting from `ERC721` by adding `ERC721` in `TGKNFTContract is ERC721, ERC721URIStorage, ERC721Enumerable`.
2. replacing the instruction `override(ERC721, ERC721Enumerable)` with the instruction `override(ERC721, ERC721URIStorage, ERC721Enumerable)`.

TENGO-4

Missing Test Suite and Limited Documentation Makes It Hard to Assess that the System Works as Expected

● High ⓘ Fixed

Update
The Tengoku Senso team documented their system (functions, roles, and privileges) in the file `README.md` and added a test suite (commit `1f8ba861cc5895ca46920a930967a749abae7678`).

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> We are working on the documentation and test suite

File(s) affected: `All files`

Description: The system documentation found in the file `README.md` is insufficient and does not clearly describe how the system is expected to work. Still, the team provided a more complete explanation during the audit.
Also, the codebase does not contain a test suite. This has two main consequences:

- it is not possible to check that the current version of the codebase behaves as expected.
- any breaking change added in the context of a system update could remain undetected by the team.

Recommendation:
Consider:

1. documenting the main user roles of the system and, for each of them, the associated use cases.
2. building a comprehensive test suite to make sure that the use cases defined in step 1 work as expected (happy and unhappy paths).

TENGO-5 Privileged Roles and Ownership

● Low ⓘ Fixed

Update
The Tengoku Senso team documented the roles and privileges in the file `README.md` (commit `1f8ba861cc5895ca46920a930967a749abae7678`).

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> We are working on the documentation and test suite

File(s) affected: `TGKMainContract.sol`

Description: The protocol owner has a privileged role and can execute specific functions. Therefore, users are exposed to the risk of being attacked if the privileged roles are malicious or compromised.

The contract `TGKMainContract` has the following owner privileges:

1. The owner can withdraw any amount owned by the contract to any arbitrary address via the function `withdraw()` .
2. The owner can transfer any amount of any ERC20 token owned by the contract to any arbitrary address via the function `transferERC20()` .
3. The owner can transfer any ERC721 token owned by the contract to any arbitrary address via the function `transferNFT()` .

Note that the owner of the contract is the address who deployed the contract and it cannot be updated.

Recommendation: Consider documenting the risk and impact a compromised privileged role can cause on the protocol and inform the users in detail. As the privileged roles can be the single point of failure of the protocol, consider using a multi-sig or a contract with a timelock feature to mitigate the risk of being compromised or exploited. More generally, consider following best practices for Key Management of admin addresses.

TENGO-6
Potential Use of Compiler Version with Known Bugs

● Low ⓘ Fixed

Update

The team now uses the solidity compiler version `v0.8.19` without the caret (`^`) before the version number (commit `dc102e8f9a4ec4473dff3d120d3f70b83d2ecd87`).

Update

Marked as "Fixed" by the client. Addressed in: `553d4156c8602e53a3a3dd1c2b10a5cbad2f1472` . The client provided the following explanation:

> In our Hardhat configuration file, we set the Solidity compiler version to 0.8.18. By including this configuration in the Hardhat file, we ensure that the contracts are compiled using the Solidity version 0.8.18. This version will be used when compiling and deploying the contracts using Hardhat.

File(s) affected: `All files`

Description: While the Solidity compiler version is not given, the pragma for the contracts is written as `^0.8.9`, allowing the contracts to be compiled using versions below `v0.8.16`, which all have known compiler bugs. A list of known bugs in each specific version can be found here.

Recommendation: Consider using a newer Solidity version without known bugs. However, avoid using a recent release that may have unknown bugs introduced by feature additions. We recommended using `v0.8.18` at the time of writing, without the caret (^) before the version number.

TENGO-7 Missing Input Validation

● Low ⓘ Fixed

> ✓ **Update**
> The client implemented the recommended actions.

> ✓ **Update**
> Marked as "Fixed" by the client. Addressed in: `c47765595a3fc73626a0792b2460542e00935913`.

File(s) affected: `TGKMainContract.sol`

Description: In the `TGKMainContract.withdraw()` function, the `to` address is not validated against the zero address. As a result, an accidental loss of funds is possible.

Recommendation: Consider adding a non-zero check.

TENGO-8 Using `transfer()` for ETH May Fail in The Future

● Informational Fixed

Update

The client implemented the recommended actions.

Update

Marked as "Fixed" by the client. Addressed in: `553d4156c8602e53a3a3dd1c2b10a5cbad2f1472`.

File(s) affected: `TGKMainContract.sol`

Description: The function `address.transfer()` assumes a fixed amount of gas to execute the call. The use of that function protects against reentrancy attacks. The default amount of gas, however, may change in the future. In the worst case, it could lead to failed transfers.

Recommendation: We want you to be aware of this possibility. Whereas we do not recommend taking any action now, if you need more flexibility regarding the forwarded gas, you can use the low-level instruction `call()` to perform transfers.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.

- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

Consider improving the readability of the code documentation by systematically using the NatSpec format (https://docs.soliditylang.org/en/develop/natspec-format.html).

Adherence to Best Practices

1. To facilitate logging, it is recommended to index address parameters within events. Therefore the `indexed` keyword could be added to:
    - the fields `nftContractAddress` and `receiverAddress` of the event `NFTTransferred` in the contract `TGKMainContract`.
    - the field `_nftOwner` of the event `NFTCreated` in the contract `TGKNFTContract`.
2. To reduce the size of a contract and ultimately reduce the gas cost of its deployment, the length of the string messages in the `require` statements can be reduced. In the contract `TGKMainContract`, messages related to ownership checks could be replaced by a shorted message such as "Not owner".
3. As the owner of the contract `TGKMainContract` is only set in the constructor and cannot be updated later, consider using the `immutable` keyword.

4. The parameters `name` and `symbol` of the constructor of the contract `TGKMainContract` shadow the name of the functions `name()` and `symbol()`. It is recommended to use other parameter names like for example `_name` and `_symbol`.
5. For the `receive()` function of `TGKMainContract`, validate that `msg.value` is non-zero to avoid excessive events that may reduce the efficiency of off-chain event monitoring.
6. The visibility of the following functions can be changed to `external`:
    - `TGKMainContract.withdraw()`
    - `TGKMainContract.transferERC20()`
    - `TGKMainContract.getERC20TokenBalance()`
    - `TGKMainContract.transferNFT()`
    - `TGKNFTContract.mintNFT()`
7. As an event `Transfer` with the first field being the `address(0)` is already emitted by the execution of the function `_safeMint()`, consider if the event `NFTCreated` is redundant or not. If it is, consider removing it and, instead, monitoring `Transfer` events with the first field being the `address(0)`.
8. Consider refactoring the event emitted in the function `TGKMainContract.transferERC20()` to include the address of the ERC20 token transferred.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `def...d3d ./TGK-Smart-Contracts-Audit-main/TGKNFTContract.sol`
- `a66...dea ./TGK-Smart-Contracts-Audit-main/TGKMainContract.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Add dependencies to the project.
3. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Relevant issues resulted in dedicated issues and best practices in that report.

Changelog

- 2023-07-07 - Initial report
- 2023-07-28 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Disclaimer