InterFi NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

SAMURAI VERSUS

INTERFI SMART CONTRACT AUDIT

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	Samurai Versus
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
GenesisFireSamurai	0x285c7E1eac419dEAb8791e600140aCf4708C5607
GenesisWaterSamurai	0xA3a92ABa1Cc7De81eF5Eb356Fef3EF2701E0E9e4
RoyaltyBalancer	0xcd71FceaA387098ce07DEB851486D77f415fc89f
	0x5D0bD2528D426270a63E352Ad0FCACEE5C227cbF
Blockchain	Binance Smart Chain
Centralization	Active ownership
Commit	93a34390cda52e6f888eb0ec2b3d38f79bdcb384
Website	https://samurai-versus.io/
Report Date	June 10, 2023

ℹ️ Verify the authenticity of this report on our website: https://www.github.com/interfinetwork

EXECUTIVE SUMMARY

InterFi has performed the automated and manual analysis of solidity codes. Solidity codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical 🔴	Major 🟠	Medium 🟡	Minor 🟢	Unknown 🟤
Open	0	1	0	2	0
Acknowledged	0	0	0	3	1
Resolved	0	1	3	5	0
Noteworthy Privileges	Check PAGE 22 for important centralized privileges				

ℹ️ Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

ℹ️ Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

TABLE OF CONTENTS

SCOPE OF WORK

InterFi was consulted by Samurai Versus to conduct the smart contract audit of their solidity source codes. **The audit scope of work is strictly limited to mentioned solidity file(s) only:**

- GenesisFireSamurai.sol
- IGenesisWaterSamurai.sol
- GenesisWaterSamurai.sol
- IGenesisFireSamurai.sol
- RoyaltyBalancer.sol
- IRoyaltyBalancer.sol

ℹ️ If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Verify the contract's deployment status below:

Verified Code Links
https://bscscan.com/address/0x285c7e1eac419deab8791e600140acf4708c5607#code
https://bscscan.com/address/0xa3a92aba1cc7de81ef5eb356fef3ef2701e0e9e4#code
https://bscscan.com/address/0xcd71fceaa387098ce07deb851486d77f415fc89f#code
https://bscscan.com/address/0x5d0bd2528d426270a63e352ad0fcacee5c227cbf#code

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of InterFi's auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
  - Remix IDE Developer Tool
  - Open Zeppelin Code Analyzer
  - SWC Vulnerabilities Registry
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	○ Token Supply Manipulation ○ Access Control and Authorization ○ Assets Manipulation ○ Ownership Control ○ Liquidity Access ○ Stop and Pause Trading ○ Ownable Library Verification

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to solidity codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

ℹ️ It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical 🔴	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major 🟠	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium 🟡	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- The client can lower centralization-related risks by implementing below mentioned practices:
- Privileged role's private key must be carefully secured to avoid any potential hack.
- Privileged role should be shared by multi-signature (multi-sig) wallets.
- Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.
- Renouncing the contract ownership, and privileged roles.
- Remove functions with elevated centralization risk.

ℹ️ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💵	Function is payable
🔒	Function is internal
🔐	Function is private
❗	Function is important

GenesisFireSamurai

| **ERC1155** | Implementation | Context, ERC165, IERC1155, IERC1155MetadataURI |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | supportsInterface | Public ❗ | |NO❗ |

| └ | uri | Public ❗ | |NO❗ |

| └ | balanceOf | Public ❗ | |NO❗ |

| └ | balanceOfBatch | Public ❗ | |NO❗ |

| └ | setApprovalForAll | Public ❗ | 🛑 |NO❗ |

| └ | isApprovedForAll | Public ❗ | |NO❗ |

| └ | safeTransferFrom | Public ❗ | 🛑 |NO❗ |

| └ | safeBatchTransferFrom | Public ❗ | 🛑 |NO❗ |

| └ | _update | Internal 🔒 | 🛑 | |

| └ | _safeTransferFrom | Internal 🔒 | 🛑 | |

| └ | _safeBatchTransferFrom | Internal 🔒 | 🛑 | |

| └ | _setURI | Internal 🔒 | 🛑 | |

| └ | _mint | Internal 🔒 | 🛑 | |

| └ | _mintBatch | Internal 🔒 | 🛑 | |

| └ | _burn | Internal 🔒 | 🛑 | |

| └ | _burnBatch | Internal 🔒 | 🛑 | |

| └ | _setApprovalForAll | Internal 🔒 | 🛑 | |

| └ | _doSafeTransferAcceptanceCheck | Private 🔐 | 🛑 | |

| └ | _doSafeBatchTransferAcceptanceCheck | Private 🔐 | 🛑 | |

| └ | _asSingletonArrays | Private 🔐 | | |

||||||

| **ERC2981** | Implementation | IERC2981, ERC165 |||

| └ | supportsInterface | Public ❗ | |NO❗ |

| └ | royaltyInfo | Public ❗ | |NO❗ |

| └ | _feeDenominator | Internal 🔒 | | |

| └ | _setDefaultRoyalty | Internal 🔒 | 🛑 | |

| └ | _deleteDefaultRoyalty | Internal 🔒 | 🛑 | |

| └ | _setTokenRoyalty | Internal 🔒 | 🛑 | |

| └ | _resetTokenRoyalty | Internal 🔒 | 🛑 | |

||||||

| **IERC2981** | Interface | IERC165 |||

| └ | royaltyInfo | External ❗ | |NO❗ |

||||||

| **IERC20** | Interface | |||

| └ | totalSupply | External ❗ | |NO❗ |

| └ | balanceOf | External ❗ | |NO❗ |

| └ | transfer | External ❗ | 🛑 |NO❗ |

| └ | allowance | External ❗ | |NO❗ |

| └ | approve | External ❗ | 🛑 |NO❗ |

| └ | transferFrom | External ❗ | 🛑 |NO❗ |

||||||

| **SafeERC20** | Library | |||

| └ | safeTransfer | Internal 🔒 | 🛑 | |

| └ | safeTransferFrom | Internal 🔒 | 🛑 | |

| └ | safeIncreaseAllowance | Internal 🔒 | 🛑 | |

| └ | safeDecreaseAllowance | Internal 🔒 | 🛑 | |

| └ | forceApprove | Internal 🔒 | 🛑 | |

| └ | safePermit | Internal 🔒 | 🛑 | |

| └ | _callOptionalReturn | Private 🔐 | 🛑 | |

| └ | _callOptionalReturnBool | Private 🔐 | 🛑 | |

||||||

| **Ownable** | Implementation | Context |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | owner | Public ❗ | |NO❗ |

| └ | _checkOwner | Internal 🔒 | | |

| └ | renounceOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | transferOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | _transferOwnership | Internal 🔒 | 🛑 | |

||||||

| **ReentrancyGuard** | Implementation | |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | _nonReentrantBefore | Private 🔐 | 🛑 | |

| └ | _nonReentrantAfter | Private 🔐 | 🛑 | |

| └ | _reentrancyGuardEntered | Internal 🔒 | | |

||||||

| **Pausable** | Implementation | Context |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | paused | Public ❗ | |NO❗ |

| └ | _requireNotPaused | Internal 🔒 | | |

| └ | _requirePaused | Internal 🔒 | | |

| └ | _pause | Internal 🔒 | 🛑 | whenNotPaused |

| └ | _unpause | Internal 🔒 | 🛑 | whenPaused |

||||||

| **IRoyaltyBalancer** | Interface | |||

| └ | addMinterShare | External ❗ | 🛑 |NO❗ |

| └ | pendingReward | External ❗ | |NO❗ |

| └ | claimReward | External ❗ | 🛑 |NO❗ |

| └ | userInfo | External ❗ | |NO❗ |

||||||

| **IGenesisWaterSamurai** | Interface | |||

| └ | getMintedAmount | External ❗ | |NO❗ |

||||||

| **GenesisFireSamurai** | Implementation | ERC1155, ERC2981, IGenesisWaterSamurai, Ownable, ReentrancyGuard, Pausable |||

| └ | <Constructor> | Public ❗ | 🛑 | ERC1155 |

| └ | mintSamurai | Public ❗ | 💵 | nonReentrant whenNotPaused |

| └ | claimFreeTokens | Public ❗ | 💵 | nonReentrant whenNotPaused |

| └ | setMintPrice | Public ❗ | 🛑 | onlyOwner |

| └ | setPublicMintPrice | Public ❗ | 🛑 | onlyOwner |

| └ | setWhitelistMintStage | Public ❗ | 🛑 | onlyOwner |

| └ | setPublicMintStage | Public ❗ | 🛑 | onlyOwner |

| └ | releaseReservedTokens | Public ❗ | 🛑 | onlyOwner |

| └ | reserveFreeMintTokens | Public ❗ | 🛑 | onlyOwner |

| └ | setContractAddress | Public ❗ | 🛑 | onlyOwner |

| └ | addToFreeMintList | Public ❗ | 🛑 | onlyOwner |

| └ | addToWhitelist | Public ❗ | 🛑 | onlyOwner |

| └ | removeFromWhitelist | Public ❗ | 🛑 | onlyOwner |

| └ | _addToFreeMintList | Internal 🔒 | 🛑 | |

| └ | _addToWhitelist | Internal 🔒 | 🛑 | |

| └ | _removeFromWhitelist | Internal 🔒 | 🛑 | |

| └ | checkWhitelistMintAvailable | Public ❗ | |NO❗ |

| └ | checkPublicMintAvailable | Public ❗ | |NO❗ |

| └ | checkFreeMintTokensReserved | Public ❗ | |NO❗ |

| └ | isFreeMintEligibled | External ❗ | |NO❗ |

| └ | isMinterWhitelisted | External ❗ | |NO❗ |

| └ | getMintedAmount | Public ❗ | |NO❗ |

| └ | getWaterSamuraiMintedAmount | Public ❗ | |NO❗ |

| └ | checkRemainingTokens | External ❗ | |NO❗ |

| └ | checkFreeMintedTokens | External ❗ | |NO❗ |

| └ | checkFreeMintAmountAvailable | External ❗ | |NO❗ |

| └ | isApprovedForAll | Public ❗ | |NO❗ |

| └ | supportsInterface | Public ❗ | |NO❗ |

| └ | uri | Public ❗ | |NO❗ |

| └ | contractURI | Public ❗ | |NO❗ |

| └ | setDefaultRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | deleteDefaultRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | setTokenRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | resetTokenRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | pause | Public ❗ | 🛑 | onlyOwner |

| └ | unpause | Public ❗ | 🛑 | onlyOwner |

| └ | withdrawFunds | Public ❗ | 🛑 | onlyOwner |

| └ | removeTokensFromContract | Public ❗ | 🛑 | onlyOwner |

| └ | <Receive Ether> | External ❗ | 💵 | nonReentrant |

GenesisWaterSamurai

| **ERC1155** | Implementation | Context, ERC165, IERC1155, IERC1155MetadataURI |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | supportsInterface | Public ❗ | |NO❗ |

| └ | uri | Public ❗ | |NO❗ |

| └ | balanceOf | Public ❗ | |NO❗ |

| └ | balanceOfBatch | Public ❗ | |NO❗ |

| └ | setApprovalForAll | Public ❗ | 🛑 |NO❗ |

| └ | isApprovedForAll | Public ❗ | |NO❗ |

| └ | safeTransferFrom | Public ❗ | 🛑 |NO❗ |

| └ | safeBatchTransferFrom | Public ❗ | 🛑 |NO❗ |

| └ | _update | Internal 🔒 | 🛑 | |

| └ | _safeTransferFrom | Internal 🔒 | 🛑 | |

| └ | _safeBatchTransferFrom | Internal 🔒 | 🛑 | |

| └ | _setURI | Internal 🔒 | 🛑 | |

| └ | _mint | Internal 🔒 | 🛑 | |

| └ | _mintBatch | Internal 🔒 | 🛑 | |

| └ | _burn | Internal 🔒 | 🛑 | |

| └ | _burnBatch | Internal 🔒 | 🛑 | |

| └ | _setApprovalForAll | Internal 🔒 | 🛑 | |

| └ | _doSafeTransferAcceptanceCheck | Private 🔐 | 🛑 | |

| └ | _doSafeBatchTransferAcceptanceCheck | Private 🔐 | 🛑 | |

| └ | _asSingletonArrays | Private 🔐 | | |

||||||

| **ERC2981** | Implementation | IERC2981, ERC165 |||

| └ | supportsInterface | Public ❗ | |NO❗ |

| └ | royaltyInfo | Public ❗ | |NO❗ |

| └ | _feeDenominator | Internal 🔒 | | |

| └ | _setDefaultRoyalty | Internal 🔒 | 🛑 | |

| └ | _deleteDefaultRoyalty | Internal 🔒 | 🛑 | |

| └ | _setTokenRoyalty | Internal 🔒 | 🛑 | |

| └ | _resetTokenRoyalty | Internal 🔒 | 🛑 | |

|||||||

| **IERC2981** | Interface | IERC165 |||

| └ | royaltyInfo | External ❗ | |NO❗ |

|||||||

| **IERC20** | Interface | |||

| └ | totalSupply | External ❗ | |NO❗ |

| └ | balanceOf | External ❗ | |NO❗ |

| └ | transfer | External ❗ | 🛑 |NO❗ |

| └ | allowance | External ❗ | |NO❗ |

| └ | approve | External ❗ | 🛑 |NO❗ |

| └ | transferFrom | External ❗ | 🛑 |NO❗ |

|||||||

| **SafeERC20** | Library | |||

| └ | safeTransfer | Internal 🔒 | 🛑 | |

| └ | safeTransferFrom | Internal 🔒 | 🛑 | |

| └ | safeIncreaseAllowance | Internal 🔒 | 🛑 | |

| └ | safeDecreaseAllowance | Internal 🔒 | 🛑 | |

| └ | forceApprove | Internal 🔒 | 🛑 | |

| └ | safePermit | Internal 🔒 | 🛑 | |

| └ | _callOptionalReturn | Private 🔐 | 🛑 | |

| └ | _callOptionalReturnBool | Private 🔐 | 🛑 | |

|||||||

| **Ownable** | Implementation | Context |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | owner | Public ❗ | |NO❗ |

| └ | _checkOwner | Internal 🔒 | | |

| └ | renounceOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | transferOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | _transferOwnership | Internal 🔒 | 🛑 | |

||||||

| **ReentrancyGuard** | Implementation | |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | _nonReentrantBefore | Private 🔐 | 🛑 | |

| └ | _nonReentrantAfter | Private 🔐 | 🛑 | |

| └ | _reentrancyGuardEntered | Internal 🔒 | | |

||||||

| **Pausable** | Implementation | Context |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | paused | Public ❗ | |NO❗ |

| └ | _requireNotPaused | Internal 🔒 | | |

| └ | _requirePaused | Internal 🔒 | | |

| └ | _pause | Internal 🔒 | 🛑 | whenNotPaused |

| └ | _unpause | Internal 🔒 | 🛑 | whenPaused |

||||||

| **IRoyaltyBalancer** | Interface | |||

| └ | addMinterShare | External ❗ | 🛑 |NO❗ |

| └ | pendingReward | External ❗ | |NO❗ |

| └ | claimReward | External ❗ | 🛑 |NO❗ |

| └ | userInfo | External ❗ | |NO❗ |

||||||

| **IGenesisFireSamurai** | Interface | |||

| └ | getMintedAmount | External ❗ | |NO❗ |

||||||

| **GenesisWaterSamurai** | Implementation | ERC1155, ERC2981, IGenesisFireSamurai, Ownable, ReentrancyGuard, Pausable |||

| └ | <Constructor> | Public ❗ | 🛑 | ERC1155 |

| └ | mintSamurai | Public ❗ | 💵 | nonReentrant whenNotPaused |

| └ | claimFreeTokens | Public ❗ | 💵 | nonReentrant whenNotPaused |

| └ | setMintPrice | Public ❗ | 🛑 | onlyOwner |

| └ | setPublicMintPrice | Public ❗ | 🛑 | onlyOwner |

| └ | setWhitelistMintStage | Public ❗ | 🛑 | onlyOwner |

| └ | setPublicMintStage | Public ❗ | 🛑 | onlyOwner |

| └ | releaseReservedTokens | Public ❗ | 🛑 | onlyOwner |

| └ | reserveFreeMintTokens | Public ❗ | 🛑 | onlyOwner |

| └ | setContractAddress | Public ❗ | 🛑 | onlyOwner |

| └ | addToFreeMintList | Public ❗ | 🛑 | onlyOwner |

| └ | addToWhitelist | Public ❗ | 🛑 | onlyOwner |

| └ | removeFromWhitelist | Public ❗ | 🛑 | onlyOwner |

| └ | _addToFreeMintList | Internal 🔒 | 🛑 | | |

| └ | _addToWhitelist | Internal 🔒 | 🛑 | | |

| └ | _removeFromWhitelist | Internal 🔒 | 🛑 | | |

| └ | checkWhitelistMintAvailable | Public ❗ | |NO❗ |

| └ | checkPublicMintAvailable | Public ❗ | |NO❗ |

| └ | checkFreeMintTokensReserved | Public ❗ | |NO❗ |

| └ | isFreeMintEligibled | External ❗ | |NO❗ |

| └ | isMinterWhitelisted | External ❗ | |NO❗ |

| └ | getMintedAmount | Public ❗ | |NO❗ |

| └ | getFireSamuraiMintedAmount | Public ❗ | |NO❗ |

| └ | checkRemainingTokens | External ❗ | |NO❗ |

| └ | checkFreeMintedTokens | External ❗ | |NO❗ |

| └ | checkFreeMintAmountAvailable | External ❗ | |NO❗ |

| └ | isApprovedForAll | Public ❗ | |NO❗ |

| └ | supportsInterface | Public ❗ | |NO❗ |

| └ | uri | Public ❗ | |NO❗ |

| └ | contractURI | Public ❗ | |NO❗ |

| └ | setDefaultRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | deleteDefaultRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | setTokenRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | resetTokenRoyalty | Public ❗ | 🛑 | onlyOwner |

| └ | pause | Public ❗ | 🛑 | onlyOwner |

| └ | unpause | Public ❗ | 🛑 | onlyOwner |

| └ | withdrawFunds | Public ❗ | 🛑 | onlyOwner |

| └ | removeTokensFromContract | Public ❗ | 🛑 | onlyOwner |

| └ | <Receive Ether> | External ❗ | 💵 | nonReentrant |

RoyaltyBalancer

| **Ownable** | Implementation | Context |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | owner | Public ❗ | |NO❗ |

| └ | _checkOwner | Internal 🔒 | | |

| └ | renounceOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | transferOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | _transferOwnership | Internal 🔒 | 🛑 | |

||||||

| **IRoyaltyBalancer** | Interface | |||

| └ | addMinterShare | External ❗ | 🛑 |NO❗ |

| └ | pendingReward | External ❗ | |NO❗ |

| └ | claimReward | External ❗ | 🛑 |NO❗ |

| └ | userInfo | External ❗ | |NO❗ |

||||||

| **RoyaltyBalancer** | Implementation | IRoyaltyBalancer, Ownable |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | setCollectionAddress | Public ❗ | 🛑 | onlyOwner |

| └ | addMinterShare | External ❗ | 🛑 | onlyCollection |

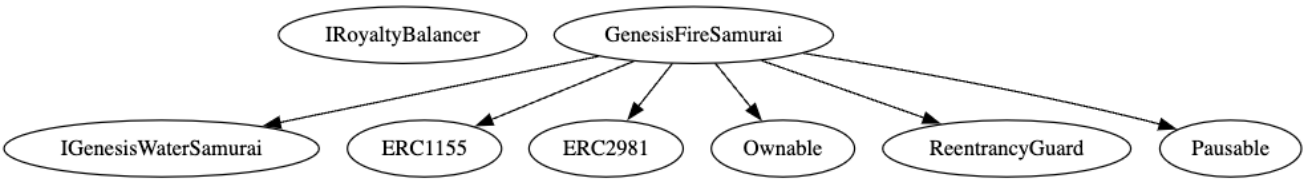
| └ | pendingReward | Public ❗ | |NO❗ |

| └ | claimReward | External ❗ | 🛑 |NO❗ |

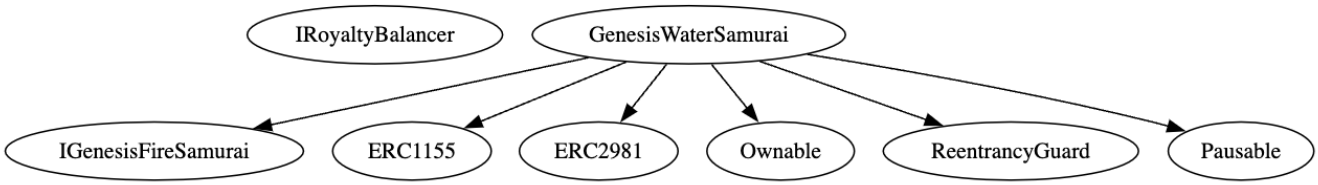
| └ | <Receive Ether> | External ❗ | 💵 |NO❗ |

INHERITANCE GRAPH

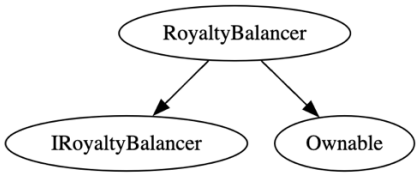
GenesisFireSamurai



GenesisWaterSamurai



RoyaltyBalancer



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized privileges	Major 🟠
CEN-07	Access control	
SAV-01	`withdrawFunds()` and `removeTokensFromContract()` clears contract balance	
CEN-05	Privileged role using contract `pause()` as a circuit breaker	

`GenesisFireSamurai`

`onlyOwner` centralized privileges are listed below:

```
setMintPrice()
setPublicMintPrice()
setWhitelistMintStage()
setPublicMintStage()
releaseReservedTokens()
reserveFreeMintTokens()
setContractAddress()
addToFreeMintList()
addToWhitelist()
removeFromWhitelist()
setDefaultRoyalty()
deleteDefaultRoyalty()
setTokenRoyalty()
resetTokenRoyalty()
pause()
unpause()
withdrawFunds()
removeTokensFromContract()
```

`GenesisWaterSamurai`

`onlyOwner` centralized privileges are listed below:

```
setMintPrice()
setPublicMintPrice()
```

```
setWhitelistMintStage()
setPublicMintStage()
releaseReservedTokens()
reserveFreeMintTokens()
setContractAddress()
addToFreeMintList()
addToWhitelist()
removeFromWhitelist()
setDefaultRoyalty()
deleteDefaultRoyalty()
setTokenRoyalty()
resetTokenRoyalty()
pause()
unpause()
withdrawFunds()
removeTokensFromContract()
```

RoyaltyBalancer

`onlyOwner` centralized privileges are listed below:

```
setCollectionAddress()
```

`onlyCollection` access control is provided to functions listed below:

```
addMinterShare()
```

RECOMMENDATION

Deployers, contract owners, administrators, access controlled, and all other privileged roles' private-keys/access-keys/admin-keys should be secured carefully. These entities can have a single point of failure that compromises the security of the project. Manage centralized and privileged roles carefully, review PAGE 09 for more information.

Identifier	Definition	Severity
LOG-01	Lack of appropriate arbitrary boundaries	Minor

Below mentioned functions are set without any arbitrary boundaries.

`addMinterShare()` – **`RoyaltyBalancer`**

`setDefaultRoyalty()` – **`GenesisFireSamurai`** and **`GenesisWaterSamurai`**

`setTokenRoyalty()` – **`GenesisFireSamurai`** and **`GenesisWaterSamurai`**

RECOMMENDATION

These functions should be provided appropriate upper input boundary.

ACKNOWLEDGEMENT

Samurai Versus team acknowledged this finding and kept the code as-is.

Identifier	Definition	Severity
LOG-02	Potential front-running	Minor

Potential front-running happens when an attacker observes a transaction minting tokens, swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by front-running a transaction to purchase assets and make profits by back-running a transaction to sell assets.

RECOMMENDATION

Use tools like commit-reveal scheme, randomizing transaction order, and/or gas-price limit to deter potential front-runners.

Identifier	Definition	Severity
LOG-03	Re-entrancy	Major 🟠
SAV-03	Checks Effects Interactions	Minor 🟢

Below mentioned function is used without a re-entrancy guard:

`claimReward()` – **`RoyaltyBalancer`**

Below mentioned functions may not be following Checks Effects Interactions pattern:

`mintSamurai()` – **`GenesisFireSamurai`** and **`GenesisWaterSamurai`**

`claimFreeTokens()` – **`GenesisFireSamurai`** and **`GenesisWaterSamurai`**

In both of these functions, external calls should be made at the end of the function to avoid re-entrancy attacks. `nonReentrant` modifier should protect against re-entrant calls.

RECOMMENDATION

Use Checks Effects Interactions pattern when handing over the flow to an external entity and guard functions against re-entrancy attacks. Re-entrancy guard is used to prevent re-entrant calls.

RESOLUTION

Samurai Versus team has added `nonReentrant` modifier check to `claimReward()`.

Re-entrancy guard in all contracts is derived from Open Zeppelin commit `49c0e43` pushed on November 08, 2022.

Identifier	Definition	Severity
LOG-04	Use of low-level calls	Minor 🟢
SAV-02	DoS with unexpected `revert`	Medium 🟡

Use of low-level `.call` for transferring ether in:

`mintSamurai()` – **`GenesisFireSamurai`** and **`GenesisWaterSamurai`** **(RESOLVED)**

`withdrawFunds()` – **`GenesisFireSamurai`** and **`GenesisWaterSamurai`**

`claimReward()` – **`RoyaltyBalancer`**

`.call` may bypass type checking, function existence check, and argument packing.

Malicious contract may cause a denial of service by making the `.call` in `if (msg.value > (amount * MINT_PRICE))`.

RECOMMENDATION

`.call` should be replaced with solidity's high-level transfer function. Check amount before-hand to mitigate denial of service attack.

SAV-02 RESOLUTION

Samurai Versus team has removed `if (msg.value > (amount * MINT_PRICE))` lines.

Identifier	Definition	Severity
COD-04	Missing or inaccurate error messages	Minor

In **`RoyaltyBalancer`** contract, custom error `OnlyCollection` is defined but it does not provide any error information.

RECOMMENDATION

Provide accurate information strings for custom errors.

RESOLUTION

Samurai Versus team has added information strings for custom errors.

Identifier	Definition	Severity
COD-05	Recommendations and comments	Minor

`GenesisFireSamurai` and **`GenesisWaterSamurai`**

- `uri()` and `contractURI()` always return the same URI regardless of the `tokenId` passed to it. IPFS hash is hardcoded into both, which means it can't be changed later if needed. **(RESOLVED)**
- `mintSamurai()` can be simplified. It has repetitive logic in different condition branches. **(RESOLVED)**
- Instead of using `IRoyaltyBalancer` contract for handling payments, you may use widely audited OpenZeppelin `PaymentSplitter`.
- `claimFreeTokens()` is marked `payable` but does not handle ether sent to it. It should either handle sent ether or should not be marked as payable to avoid accidentally receiving ether.

RECOMMENDATION

Implement recommended changes to optimize smart contract functionalities.

PARTIAL RESOLUTION

Samurai Versus team has removed `contractURI()` and updated `mintSamurai()` implementation.

Identifier	Definition	Severity
COD-06	Hardcoded operator approval	Medium 🟡

GenesisFireSamurai and **GenesisWaterSamurai**

`isApprovedForAll()` has hardcoded approval for a specific address.

```
    function isApprovedForAll(address _owner, address _operator)
        public
        view
        override
        returns (bool isOperator)
    {
        /* @dev OpenSea whitelisting. This feature will allow users to list tokens on the
marketplace without paying gas for an additional approval
        If OpenSea's ERC1155 Proxy Address is detected, auto-return true */
        if (_operator == address(0x207Fa8Df3a17D96Ca7EA4f2893fcdCb78a304101)) {
            return true;
        }
        // otherwise, use the default ERC1155.isApprovedForAll()
        return ERC1155.isApprovedForAll(_owner, _operator);
    }
```

RECOMMENDATION

Hardcoded approval must be carefully reviewed and only used if it's absolutely necessary. Hardcoded operator approval can be a security risk.

RESOLUTION

Samurai Versus team has removed hardcoded operator approval implementation from `isApprovedForAll()`.

Identifier	Definition	Severity
COD-07	Non-conforming use of `unchecked` block	Medium 🟡

`GenesisFireSamurai` and **`GenesisWaterSamurai`**

`unchecked` block is used to avoid reverting on overflow/underflow. However, using it in these functions may introduce logical non-conformities and potential bugs:

```
mintSamurai()
claimFreeTokens()
```


RECOMMENDATION

Use of `unchecked` block in this context must be carefully reviewed and only applied if it's absolutely necessary.

RESOLUTION

Samurai Versus team has removed `unchecked` blocks from aforementioned functions.

Identifier	Definition	Severity
COD-09	Unclear logic	Minor
COD-14	Possible rounding errors	

RoyaltyBalancer

`accumulatedRewardPerShare` is modified in `receive()`, but it's unclear how rewards are accumulated and distributed.

```
receive() external payable {
  accumulatedRewardPerShare = accumulatedRewardPerShare + msg.value / totalShares;
}
```

`msg.value / totalShares` may have rounding errors. If `totalShares` is 0, it will throw an exception due to division by zero.

RECOMMENDATION

Provide documentation of intended behavior for better function understanding.

ACKNOWLEDGEMENT

Samurai Versus team acknowledged this finding and kept the code as-is.

Identifier	Definition	Severity
COD-10	Third Party Dependencies	Unknown

Smart contracts are interacting with third party protocols e.g., Market Makers, Marketplace, Centralized and Decentralized Front-end Applications, Open Zeppelin tools. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary.

ACKNOWLEDGEMENT

Samurai Versus team will inspect external and internal dependencies periodically to deter potential down-time, exploits, vulnerabilities, etc.

Identifier	Definition	Severity
COD-12	Missing events	Minor

Smart contracts use function calls to update state, which can make it difficult to track and analyze changes to the contract over time. Both **`GenesisFireSamurai`** and **`GenesisWaterSamurai`** contracts use events to track state changes. However, function like `setContractAddress()` is missing emit events.

Smart contract **`RoyaltyBalancer`** is missing emit events to track state changes.

RECOMMENDATION

Use events to track state changes. Events improve transparency and provide a more granular view of contract activity.

RESOLUTION

Samurai Versus team has added events to track important state changes.

Identifier	Definition	Severity
COD-15	No initialization	Minor

RoyaltyBalancer

`constructor()` is empty and there is no way to initialize essential state variables like the initial `collection` address. You might want to have the contract initialized with necessary state variables.

RECOMMENDATION

Initialize contract with necessary state variables.

ACKNOWLEDGEMENT

Samurai Versus team acknowledged this finding and kept the code as-is.

Identifier	Definition	Severity
COM-01	Floating compiler status	Minor

`GenesisFireSamurai` and **`GenesisWaterSamurai`**

Compiler is set to `^0.8.17`

`RoyaltyBalancer`

Compiler is set to `^0.8.13`

RECOMMENDATION

Pragma should be fixed to the version that you're indenting to deploy your contracts with.

RESOLUTION

Samurai Versus team has fixed compiler pragma to `0.8.17.`

DISCLAIMERS

InterFi Network provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. 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. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way

to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

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

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, 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, INTERFI NETWORK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'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.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. InterFi Network does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than InterFi Network. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' and social accounts' owners. You agree that InterFi Network is not responsible for the content or operation of such websites and social accounts and that InterFi Network shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT INTERFI NETWORK

InterFi Network provides intelligent blockchain solutions. We provide solidity development, testing, and auditing services. We have developed 150+ solidity codes, audited 1000+ smart contracts, and analyzed 500,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Velas, Oasis, etc.

InterFi Network is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 6+ casual contributors.

Website: https://interfi.network

Email: hello@interfi.network

GitHub: https://github.com/interfinetwork

Telegram (Engineering): https://t.me/interfiaudits

Telegram (Onboarding): https://t.me/interfisupport

interfinetwork

hello@interfi.network

https://interfi.network

SMART CONTRACT AUDITS | SOLIDITY DEVELOPMENT AND TESTING

RELENTLESSLY SECURING PUBLIC AND PRIVATE BLOCKCHAINS