SolyzerX
BLOCKCHAIN SECURITY

8sian Main Collection

Security Assesment

DECEMBER 2022

SolyzerX

SolyzerX.com

Prepared for:

Nicole Yap
8sian Founder

Prepared by:

Adrian Mikhail
SolyzerX Engineer

SolyzerX

Introduction

Auditing Firm	SolyzerX
Client Firm	8sian
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0x198478F870d97D62D640368D111b979d7CA3c38F
Blockchain	Ethereum
Centralization	Active Ownership
NFT Collection	8SIAN Main Collection
Website	https://8sian.io
Discord	https://discord.com/invite/8sian
Opensea	https://opensea.io/collection/8sian-main-collection
Twitter	https://twitter.com/8siannft
Report Date	December 3, 2022

Verify the authenticity of this report on our website: https://solyzerx.com/audits

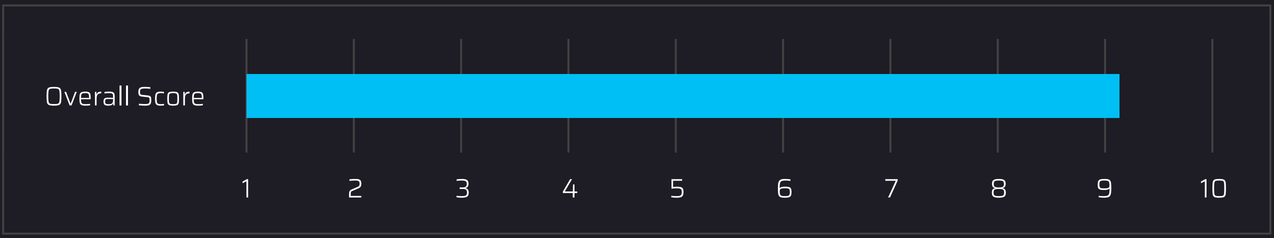
Executive Summary

SolyzerX 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:

Severity	High	Medium	Low	Informational	Undetermined
Count	0	1	3	8	1

Category	Denial of service	Data Validation	Arithmetic	Auditing and Logging	Undefined Behavior
Count	0	2	1	2	8

8sian NFT's smart contract source codes have achieved the following score: 9.2



❗ 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.

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

SolyzerX

Table of Contents

Scope of Work

SolyzerX was consulted by one of 8sian NFT (Main Collection) holders 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:

○ _8SIAN_.sol

❗ 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:

Public Contract Link	
https://etherscan.io/address/0x198478f870d97d62d640368d111b979d7ca3c38f# code	
Contract Name	_8SIAN_
Compiler Version	0.8.7
License	MIT license

Audit Methodology

Smart contract audits are conducted using a set standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of SolyzerX'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
  - Slither-SolyzerX
  - SWC Vulnerabilities Registry

- 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.

❗ 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 view:

Risk Type	Definition
High	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Medium	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.
Low	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.
Informational	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.
Undetermined	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed to mitigate the risk uncertainty.

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

Category Breakdown				
Denial of service	Data Validation	Arithmetic	Auditing and Logging	Undefined Behavior

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.

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

Contract	Function	Visibility	Modifiers
IERC721Receiver	onERC721Received	External	
IERC165	supportsInterface	External	
ERC165	supportsInterface	External	
	supportsInterface	Public	
IERC721	supportsInterface	External	
	balanceOf	External	
	ownerOf	External	
	safeTransferFrom	External	
	transferFrom	External	
	approve	External	
	getApproved	External	
	setApprovalForAll	External	
	isApprovedForAll	External	
	safeTransferFrom	External	

IERC721Metadata	balanceOf	External	
	ownerOf	External	
	safeTransferFrom	External	
	transferFrom	External	
	approve	External	
	getApproved	External	
	setApprovalForAll	External	
	isApprovedForAll	External	
	safeTransferFrom	External	
	supportsInterface	External	
	name	External	
	symbol	External	
	tokenURI	External	
ERC721	name	External	
	symbol	External	
	tokenURI	External	
	balanceOf	External	

	ownerOf	External	
	safeTransferFrom	External	
	transferFrom	External	
	approve	External	
	getApproved	External	
	setApprovalForAll	External	
	isApprovedForAll	External	
	safeTransferFrom	External	
	supportsInterface	External	
	supportsInterface	Public	
	_msgSender	Internal	
	_msgData	Internal	
	constructor	Public	
	supportsInterface	Public	
	balanceOf	Public	
	ownerOf	Public	
	name	Public	
	symbol	Public	
	tokenURI	Public	

	_baseURI	Internal	
	approve	Public	
	getApproved	Public	
	setApprovalForAll	Public	
	isApprovedForAll	Public	
	transferFrom	Public	
	safeTransferFrom	Public	
	safeTransferFrom	Public	
	_safeTransfer	Internal	
	_exists	Internal	
	_isApprovedOrOwner	Internal	
	_safeMint	Internal	
	_safeMint	Internal	
	_mint	Internal	
	_burn	Internal	
	_transfer	Internal	
	_approve	Internal	
	_setApprovalForAll	Internal	
	_checkOnERC721Received	Private	

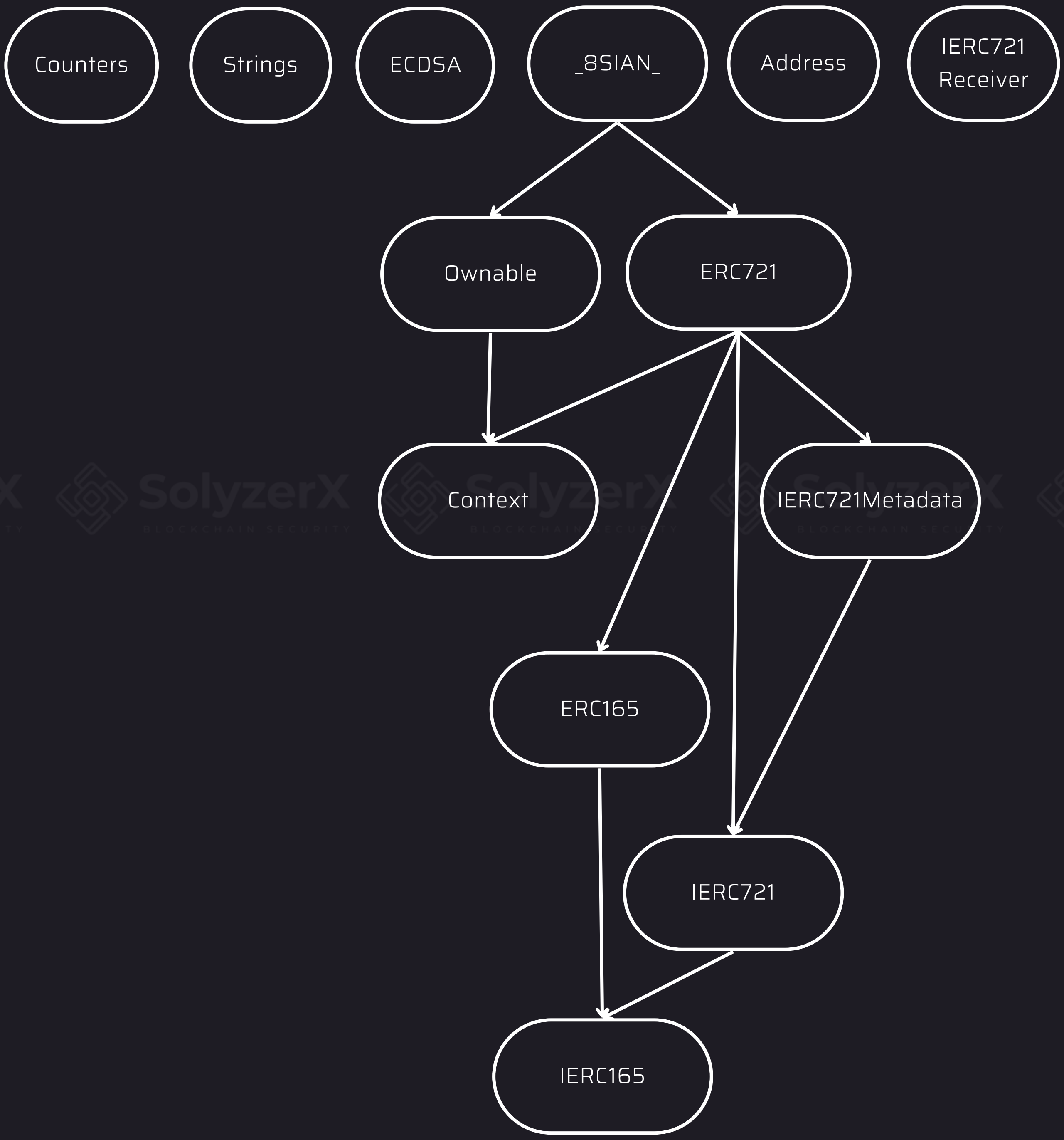
	_beforeTokenTransfer	Internal	
8SIAN	constructor	Internal	
	owner	Public	
	renounceOwnership	Public	onlyOwner
	transferOwnership	Public	onlyOwner
	_transferOwnership	Internal	
	_msgSender	Internal	
	_msgData	Internal	
	constructor	Public	
	supportsInterface	Public	
	balanceOf	Public	
	ownerOf	Public	
	name	Public	
	symbol	Public	
	tokenURI	Public	
	_baseURI	Internal	
	approve	Public	
	getApproved	Public	

	setApprovalForAll	Public	
	isApprovedForAll	Public	
	transferFrom	Public	
	safeTransferFrom	Public	
	safeTransferFrom	Public	
	_safeTransfer	Internal	
	_exists	Internal	
	_isApprovedOrOwner	Internal	
	_safeMint	Internal	
	_safeMint	Internal	
	_mint	Internal	
	_burn	Internal	
	_transfer	Internal	
	_approve	Internal	
	_setApprovalForAll	Internal	
	_checkOnERC721Received	Private	
	_beforeTokenTransfer	Internal	
	name	External	
	symbol	External	

	tokenURI	External	
	balanceOf	External	
	ownerOf	External	
	safeTransferFrom	External	
	transferFrom	External	
	approve	External	
	getApproved	External	
	setApprovalForAll	External	
	isApprovedForAll	External	
	safeTransferFrom	External	
	supportsInterface	External	
	supportsInterface	Public	
	constructor	Public	
	matchAddresSigner	Private	
	gift	External	onlyOwner
	founderMint	External	onlyOwner
	privateMint	External	
	claim	External	
	mint	External	

	withdraw	External	onlyOwner
	withdrawB	External	onlyOwner
	togglePrivateMintStatus	External	onlyOwner
	togglePublicMintStatus	External	onlyOwner
	toggleClaimFreeMintStatus	External	onlyOwner
	setPrice	External	onlyOwner
	setTeamReserve	External	onlyOwner
	setFreeClaimReserve	External	onlyOwner
	setPublic	External	onlyOwner
	setMax	External	onlyOwner
	setBaseURI	External	onlyOwner
	tokenURI	Public	
	totalSupply	Public	
	receive	External	

Inheritance Graph



Findings Summary

#	Title	Type	Severity
1	ERC721. _checkOnERC721Received() ignores return value by IERC721Receiver()	Data Validation	Medium
2	_8SIAN_.setTeamReserve() should emit an event for: - M_8SIAN_TEAM_RESERVE = newCount	Arithmetic	Low
3	ERC721._checkOnERC721Received() has external calls inside a loop: IERC721Receiver(to).onERC721Received()	Data Validation	Low
4	Variable 'ECDSA.tryRecover().r' potentially used before declaration: r = mload()(signature + 0x20)	Undefined Behavior	Low
5	ECDSA.tryRecover(bytes32,bytes) (8sian.sol#179-208) uses assembly	Undefined Behavior	Informational
6	Different versions of Solidity are used: - Version used: ['^0.8.0', '^0.8.7']	Undefined Behavior	Informational
7	_8SIAN_.gift(address[]) has costly operations inside a loop: - teamTokensMinted ++	Auditing and Logging	Informational
8	Address.functionCall(address,bytes) is never used and should be removed	Undefined Behavior	Informational
9	_8SIAN_.M_8SIAN_MAX is set pre-construction with a non-constant function or state variable	Undefined Behavior	Informational
10	solc-0.8.7 is not recommended for deployment	Undefined Behavior	Informational

11	Low level call in Address.sendValue()	Auditing and Logging	Informational
12	Parameter ERC721.safeTransferFrom()._data is not in mixedCase	Undefined Behavior	Informational
13	_8SIAN_.M_8SIAN_PER_MINT should be constant	Undefined Behavior	Undetermined

Detailed Findings

1. ERC721. _checkOnERC721Received() ignores return value by IERC721Receiver()	
Severity: Medium	Difficulty: Medium
Type: Data Validation	Finding ID: _8SIAN_.sol#1279-1289
Target: ERC721.sol	

Description

The return value of an external call is not stored in a local or state variable.

Exploit Scenario:

```
contract MyConc{
    using SafeMath for uint;
    function my_func(uint a, uint b) public{
        a.add(b);
    }
}
```

MyConc calls add of SafeMath, but does not store the result in a. As a result, the computation has no effect.

Recommendation

Ensure that all the return values of the function calls are used.

2. _8SIAN_.setTeamReserve() should emit an event for: - M_8SIAN_TEAM_RESERVE = newCount	
Severity: Low	Difficulty: Medium
Type: Arithmetic	Finding ID: _8SIAN_.sol#1472-1474
Target: _8SIAN_.sol	

Description

Detect missing events for critical arithmetic parameters.

Exploit Scenario:

```
contract C {

    modifier onlyOwner {
        if (msg.sender != owner) throw;
        _;
    }

    function setBuyPrice(uint256 newBuyPrice) onlyOwner public {
        buyPrice = newBuyPrice;
    }

    function buy() external {
     ... // buyPrice is used to determine the number of tokens purchased
    }
}
```

updateOwner() has no event, so it is difficult to track off-chain changes in the buy price.

Recommendation

Emit an event for critical parameter changes.

3. ERC721._checkOnERC721Received() has external calls inside a loop: IERC721Receiver(to).onERC721Received()	
Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: _8SIAN_.sol#1272-1293
Target: ERC721.sol	

Description

Calls inside a loop might lead to a denial-of-service attack.

Exploit Scenario:

```
contract CallsInLoop{

    address[] destinations;

    constructor(address[] newDestinations) public{
        destinations = newDestinations;
    }

    function bad() external{
        for (uint i=0; i < destinations.length; i++){
            destinations[i].transfer(i);
        }
    }

}
```

If one of the destinations has a fallback function that reverts, bad will always revert.

Recommendation

Favor pull over push strategy for external calls.

4. Variable 'ECDSA.tryRecover().r' potentially used before declaration: r = mload()(signature + 0x20)	
Severity: Low	Difficulty: High
Type: Undefined Behavior	Finding ID: _8SIAN_.sol#179-208
Target: _8SIAN_.sol, ECDSA	

Description

Detects the possible usage of a variable before the declaration is stepped over (either because it is later declared, or declared in another scope).

Exploit Scenario:

```
contract C {
    function f(uint z) public returns (uint) {
        uint y = x + 9 + z; // 'z' is used pre-declaration
        uint x = 7;

        if (z % 2 == 0) {
            uint max = 5;
            // ...
        }

        // 'max' was intended to be 5, but it was mistakenly declared in
a scope and not assigned (so it is zero).
        for (uint i = 0; i < max; i++) {
            x += 1;
        }

        return x;
    }
}
```

In the case above, the variable x is used before its declaration, which may result in unintended consequences. Additionally, the for-loop uses the variable max, which is declared in a previous scope that may not always be reached. This could lead to unintended consequences if the user mistakenly uses a variable prior to any intended declaration assignment. It also may indicate that the user intended to reference a different variable.

Recommendation

Move all variable declarations prior to any usage of the variable, and ensure that reaching a variable declaration does not depend on some conditional if it is used unconditionally.

5. ECDSA.tryRecover(bytes32,bytes) uses assembly	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: _8SIAN_.sol#179-208
Target: _8SIAN_.sol, ECDSA	

Description

The use of assembly is error-prone and should be avoided.

Recommendation

Do not use evm assembly.

6. Different versions of Solidity are used: - Version used: ['^0.8.0', '^0.8.7']	
Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: _8SIAN_.sol# 902 & # 1316
Target: _8SIAN_.sol	

Description

Detect whether different Solidity versions are used.

Recommendation

Use one Solidity version.

7. _8SIAN_.gift(address[]) has costly operations inside a loop: - teamTokensMinted ++	
Severity: Informational	Difficulty: Medium
Type: Auditing and Logging	Finding ID: _8SIAN_.sol#1357-1365
Target: _8SIAN_.sol	

Description

Costly operations inside a loop might waste gas, so optimizations are justified.

Exploit Scenario:

```
contract CostlyOperationsInLoop{

    uint loop_count = 100;
    uint state_variable=0;

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
      uint local_variable = state_variable;
      for (uint i=0; i < loop_count; i++){
        local_variable++;
      }
      state_variable = local_variable;
    }
}
```

Incrementing state_variable in a loop incurs a lot of gas because of expensive SSTOREs, which might lead to an out-of-gas.

Recommendation

Use a local variable to hold the loop computation result.

8. Address.functionCall(address,bytes) is never used and should be removed	
Severity: Informational	Difficulty: Medium
Type: Undefined Behavior	Finding ID: _8SIAN_.sol#528-530
Target: _8SIAN_.sol	

Description

Functions that are not sued.

Exploit Scenario:

```
contract Contract{
    function dead_code() internal() {}
}
```

dead_code is not used in the contract, and make the code's review more difficult.

Recommendation

Remove unused functions.

9. _8SIAN_.M_8SIAN_MAX is set pre-construction with a non-constant function or state variable	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: _8SIAN_.sol#1328
Target: _8SIAN_.sol	

Description

Detects the immediate initialization of state variables through function calls that are not pure/constant, or that use non-constant state variable.

Exploit Scenario:

```
contract StateVarInitFromFunction {

    uint public v = set(); // Initialize from function (sets to 77)
    uint public w = 5;
    uint public x = set(); // Initialize from function (sets to 88)
    address public shouldntBeReported = address(8);

    constructor(){
        // The constructor is run after all state variables are
initialized.
    }

    function set() public  returns(uint)  {
        // If this function is being used to initialize a state variable
declared
        // before w, w will be zero. If it is declared after w, w will be
set.
        if(w == 0) {
            return 77;
        }

        return 88;
    }
}
```

In this case, users might intend a function to return a value a state variable can initialize with, without realizing the context for the contract is not fully initialized. In the example above, the same function sets two different values for state variables because it checks a state variable that is not yet initialized in one case, and is initialized in the other. Special care must be taken when initializing state variables from an immediate function call so as not to incorrectly assume the state is initialized.

Recommendation

Remove any initialization of state variables via non-constant state variables or function calls. If variables must be set upon contract deployment, locate initialization in the constructor instead.

10. solc-0.8.7 is not recommended for deployment	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: _8SIAN_.sol#1316
Target: _8SIAN_.sol	

Description

solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statement.

Recommendation

Deploy with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.16

The recommendations take into account:

- Risks related to recent releases
- Risks of complex code generation changes
- Risks of new language features
- Risks of known bugs

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

11. Low level call in Address.sendValue()	
Severity: Informational	Difficulty: High
Type: Auditing and Logging	Finding ID: _8SIAN_.sol#503-508
Target: _8SIAN_.sol	

Description

The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.

Recommendation

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

12. Parameter ERC721.safeTransferFrom()._data is not in mixedCase	
Severity: Informational	Difficulty: High
Type: Undefined Behavior	Finding ID: _8SIAN_.sol#1069
Target: _8SIAN_.sol	

Description

Solidity defines a naming convention that should be followed.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (ERC20).
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

Recommendation

Follow the Solidity naming convention.

13. _8SIAN_.M_8SIAN_PER_MINT should be constant	
Severity: Undetermined	Difficulty: Medium
Type: Undefined Behavior	Finding ID: _8SIAN_.sol#1330 & #1334
Target: _8SIAN_.sol	

Description

Constant state variables should be declared constant to save gas.

Recommendation

Add the constant attributes to state variables that never change.

Disclaimers

SolyzerX 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 level 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 SolyzerX'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 RESULT 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, SOLYZERX HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, SOLYZERX 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, SOLYZERX 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. SolyzerX 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 SolyzerX. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such website's and social account's owners. You agree that SolyzerX is not responsible for the content or operation of such websites and social accounts and that SolyzerX 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.

SolyzerX

About SolyzerX

Founded in 2022 and headquartered in Malaysia, SolyzerX provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code.

We provide solidity development, testing, and auditing services. We work on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Velas, Oasis, etc.

SolyzerX is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 5+ casual contributors.

Website: https://solyzerx.com

Email: support@solyzerx.com

Github: https://github.com/SolyzerX

Telegram (Channel): https://t.me/SolyzerX

Telegram (Foundation): https://t.me/SolyzerXFoundation

solyzerx

solyzerx.com

t.me/solyzerx

t.me/solyzerxfoundation

SMART CONTRACT AUDITS | SOLIDITY DEVELOPMENT AND TESTING

RELENTLESSLY SECURING PUBLIC AND PRIVATE BLOCKCHAINS