QuillAudits

Audit Report, August, 2024

For

XProtocol

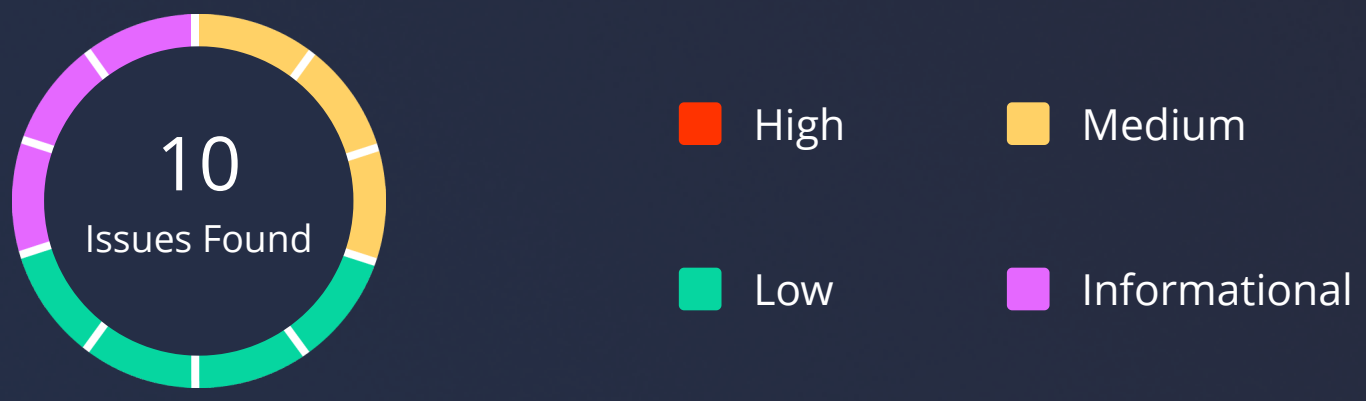
Table of Content

Table of Content

Executive Summary

Project Name	XProtocol
Project URL	https://xprotocol.org/
Overview	XProtocol is an Entertainment-focused L3 Superchain, supported by prominent backers including Saison Capital, Zephyrus Capital, Dragonfly, and Coinfund. Built on the robust Base Layer 2, XProtocol aims to redefine the digital entertainment landscape by leveraging Superchain technology to create a dynamic, value-driven environment for the mass audience.
Audit Scope	https://drive.google.com/drive/folders/1dmge58jgmnRlTOOcOcBFuVe4y25mO4ZN?usp=share_link
Contracts In-Scope	Contracts:- ├─── node | ├─── NodeLicense.sol
Commit Hash	NA
Language	Solidity
Blockchain	Ethereum
Method	Manual Review, Automated Tools.
Review 1	06th August 2024 - 17th August 2024
Updated Code Received	20th August 2024
Review 2	21st August 2024 - 22nd August 2024
Fixed In	https://drive.google.com/drive/folders/12RJUcM8e1suPVzDVnkoWo0k3MIPqAprQ?usp=share_link

Number of Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	2	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	3	2	3

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Staking contract

  Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Hardhat, Solhint, Slither, Mythril, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. Contract - NovaLicenseV1.sol

Medium Severity Issues

A.1 Values in `_promoCodesByRecipient` might get stale

Line	Function -

238-250	function updatePromoCode(
	string calldata _promoCode,
	address _recipient,
	uint256 _referralDiscountPercentage,
	uint256 _referralRewardPercentage
	) external onlyRole(PROMO_ROLE) {
	if (_recipient == address(0)) {
	revert InvalidAddress(ZERO_ADDRESS);
	}
	if (_promoCodes[_promoCode].recipient == address(0)) {
	revert PromoCodeDoesNotExist();
	}
	.....
	emit PromoCodeUpdated(_promoCode, _recipient);
	}

Description

The updatePromoCode function is designed to modify existing promo codes. While it successfully updates the _promoCodes data structure, it does not perform the requisite updates to the _promoCodesByRecipient data structure. This oversight may lead to potential inconsistencies in the system's state. Similar updates should be done in removePromoCode.

Remediation

Correctly, set the value in _promoCodesByRecipient in the function of removePromoCode & removePromoCode.

Status

Resolved

A.2 Missing validation of `referralDiscountPercentage`

Line	Function -

```
function createPromoCode(
    string calldata _promoCode,
    address _recipient,
    uint256 _referralDiscountPercentage,
    uint256 _referralRewardPercentage
) external onlyRole(PROMO_ROLE) {
    if (_recipient == address(0)) {
        revert InvalidAddress(ZERO_ADDRESS);
    }
    if (bytes(_promoCodesByRecipient[_recipient]).length > 0) {
        revert InvalidAddress(RECEIPENT_ALREADY_HAS_PROMO);
    }
    if (_promoCodes[_promoCode].recipient != address(0)) {
        revert PromoCodeAlreadyExists();
    }
    _promoCodes[_promoCode] = PromoCode({
        recipient: _recipient,
        active: true,
        receivedLifetime: 0,
        purchasedNodes: 0,
        referralDiscountPercentage: _referralDiscountPercentage,
        referralRewardPercentage: _referralRewardPercentage
    });
    _promoCodesByRecipient[_recipient] = _promoCode;
    emit PromoCodeCreated(_promoCode, _recipient);
}
```

Description

The PromoCode.referralDiscountPercentage must always be maintained at a value less than 100%. Setting this value to 100% would result in the issuance of an NFT at no cost.

Remediation

Correctly, validates the referralDiscountPercentage value in function createPromoCode & updatePromoCode.

Status

Resolved

A.3 Missing validation in function `setOrAddPricingTier`

Line **Function -**

391-414

```
    function setOrAddPricingTier(uint256 _index, uint256 _price, uint256
_quantity, uint256 _whitelistQuantity)
        external
        onlyRole(DEFAULT_ADMIN_ROLE)
    {

        if (_whitelistQuantity > _quantity) {
            revert InvalidInput(WHITELIST_QUANTITY_TOO_HIGH);
        }
        if (_index < pricingTiers.length) {
            // Subtract the quantity of the old tier from maxSupply
            maxSupply -= pricingTiers[_index].quantity;
            pricingTiers[_index] =
                Tier(_price, _quantity, _whitelistQuantity,
pricingTiers[_index].whitelistSoldQuantity);
        } else {
            pricingTiers.push(Tier(_price, _quantity, _whitelistQuantity, 0));
        }

        // Add the quantity of the new or updated tier to maxSupply
        maxSupply += _quantity;
        emit PricingTierSetOrAdded(_index, _price, _quantity);
    }
```

Description

The Tier.whitelistSoldQuantity should constantly be less than or equal to Tier.whitelistQuantity. However, the setOrAddPricingTier function, which updates price tiers, may not always enforce this constraint. Consequently, this could lead to an inconsistent state within the system.

Remediation

Correctly, assess the new values of the pricing tier and avoid making these operational mistakes by having a validation inside.

Status

Resolved

Low Severity Issues

A.4 Use `nonReentrant` modifier of ReentrancyGuardUpgradeable contract

Description

The contract contains numerous functions that are susceptible to reentrancy attacks, given the extensive use of .send and .call opcodes. To robustly address this issue, it is advisable to implement a non-reentrancy guard across all functions. This approach would effectively mitigate reentrancy vulnerabilities and enhance the contract's security.

Status

Resolved

A.5 Missing events in some functions

Description

Certain functions are not emitting events, which may lead to inadequate indexing and monitoring of the system's state. The list are as follows -

- setWhitelistSalePeriod function
- setWhitelist function
- whitelistMint function

Status

Acknowledged

A.6 Unclear administrative responsibility of function `withdrawFunds`

Description

The withdrawFunds function possesses significant administrative capabilities and should not be managed by an Externally Owned Account (EOA). It is advisable to utilize a multisignature (multisig) wallet for this purpose. Additionally, to mitigate issues when transferring funds to a contract, employ the .send or .call methods with a specified gas limit or the nonReentrant modifier, rather than using the .transfer method.

Status

Resolved

A.7 Lack of Code coverage report

Description

The project currently lacks a code coverage report. Such a report is crucial for tracking the reliability and correctness of the tests that validate the smart contract code. Code coverage reports are instrumental in identifying bugs and issues early in the development cycle and provide a safety net when making changes or adding new features.
Implementing a coverage report will enhance the robustness of the testing process and improve overall project quality.

Status

Acknowledged

Informational Issues

A.8 Inconsistent naming of the function removePromoCode, it's deactivating Promo code but name implies that it removed Promo code.

Status

Resolved

A.9 Remove dead commented code of _getWhitelistByAddrAndTier function.

Status

Resolved

A.10 Conditional check in claimableNodeAmounts[msg.sender] <= 0 of claimNode function can be == instead of <0

Status

Resolved

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the XProtocol. We performed our audit according to the procedure described above.

Some issues Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in XProtocol smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of XProtocol smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services.. It is the responsibility of the XProtocol to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2024

For

XProtocol

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com