Guild Audits

SEPTEMBER 2022

WEB3BRIDGE SMART CONTRACT AUDIT REPORT

By Guild Audits

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Description:
BlossomingWeb3Bridge is an NFT smart contract that allows for different kinds of minting. This contract integrates ERC721A that permits for minting of more than one token at once. It also inherits Access Control contract from Openzeppelin standard and Chainlink Price Oracle; for the purpose of administration management and for the purpose of price estimation for mint, respectively.

PROJECT AUDIT SCOPE AND FINDINGS

The motive of this audit is to review the codebase of **Web3Bridge** contracts for the purpose of achieving secured, corrected and quality contracts.

Number of Contracts in Scope:

- BlossomingWeb3Bridge Contract (483 SLOC)

Link to Project codebase:
https://etherscan.io/token/0xa2b0d663b1b4e6acb34865f2d24907dc7ae4a9fb

Duration for Audit: September 01, 2022 to September 11, 2021

Audit Methodology:

Issues found:

MODE OF AUDIT AND METHODOLOGIES

The mode of audit carried out in this audit process is as follow:

- **Manual Review**: This is the first and principal step carried out to understand the business logic behind a project. At this point, it involves research, discussion and establishment of familiarity with contracts. Manual review is critical to understand the nitty-gritty of the contracts.
- **Automated Testing:** This is the running of tests with audit tools to further cement discoveries from the manual review. After a manual review of a contract, the audit tools which could be Slither, Echidna, or Mythril are run on the contract to find out issues.
- **Functional Testing:** Functional testing involves manually running unit, static, and dynamic testing. This is done to find out possible exploit scenarios that could be used to steal money from the contracts. This helps understand the functionality of the contracts and find out lapses in the reverts check in contract.

The methodologies for establishing severity issues:

- High Level Severity Issues
- Medium Level Severity Issues
- Low Level Severity Issues
- Informational Level Severity Issues

FUNCTIONAL TESTING

Some functional testing were carried out to help ascertain code security:

- Should get the usdc address assigned to contract. (318ms)
- Should get the cost whitelist assigned to contract.
- Should get the cost PublicSale assigned to contract.
- Should get the baseToken URI assigned to contract.
- Should get the TOTAL_COLLECTION_SUPPLY assigned to contract.
- Should get the whitelistAmount assigned to contract.
- Should get the totalSupply of tokens in contract.
- Should calculate and get the whitelistPriceEth of tokens in contract. (2392ms)
- Should calculate and get the calPublicPriceEth of tokens in contract.
- Should call the setRevealed by moderator of the contract. (64ms)
- Should revert when setRevealed is not called by moderator of the contract.
- Should call the setCostWhitelist by moderator of the contract. (50ms)
- Should call the setcostPublicSale by moderator of the contract. (40ms)
- Should call the setHiddenMetadataUri by moderator of the contract. (55ms)

- Should call the setPaused by moderator of the contract.
- Should call the setWhitelistTrue by moderator of the contract. (72ms)
- Should revert when moderator calls the investorsWhitelistUsers function but incorrect length mismatch.
- Should revert when moderator calls the investorsWhitelistUsers function but with address zero.
- Should revert InvestorWhiteListMint when contract is pasued.
- Should revert InvestorWhiteListMint when user is not aded to investorsWhitelist
- Should revert when whitelist period disabld (72ms)
- Should successfully mint expect quantity assigned to an investor (165ms)
- Should reverts when an investor tries to mint twice (139ms)
- Should revert when quantity passed is zero
- Should revert when quantity passed exceeded whitelistAmount
- Should revert when a caller is not whitelisted address
- Should approve contract and send usdc to mint (4933ms)
- Should get the toknURI after a successful mint. (1335ms)
- Should get ownership URI (1241ms)
- Should revert when the whitelist mints but exceed the max mint amount. (1275ms)

REPORT OF FINDINGS

HIGH SEVERITY ISSUES

- **Issue: wrong transfer made**
  - Context: 173
  - **Description:** The function **WhitlistMint()** and **publicMint** requires ether to be sent into the contract from users who want to mint with Eth but The logic of the contract sent the Ether to the user instead and still mint successfully to the the users which will cause double loss to the contract by sending nft without getting paid and still sending ether to the userAccount. The vulnerability is in the code snippet below:

```
171             );
172         } else {
173             (bool sent, ) = payable(msg.sender).call{
174                 value: calWhitelistPriceEth() * quantity
175             }("");
176             require(sent, "Failed to send Ether");
177         }
```

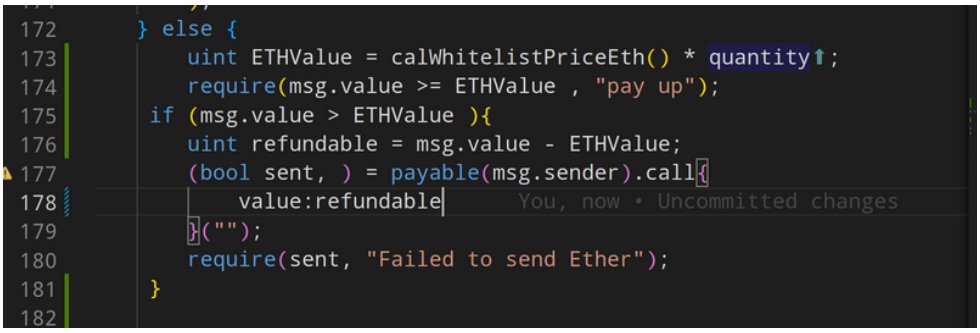
Recommendation: It is recommended to remove the above code snippet from the code and write another logic that ensures Ether is being transferred from the user to the smart contract as intended by the author. The code snippets below are suggestions to pick from in transferring ether successfully into the smart contract.

- Recommendation (a)

```
172         } else {
173             require(msg.value >= calWhitelistPriceEth() * quantity ,
                "pay up");
174
```

After the above logic has been implemented there should be a logic that transfers the excess of the money sent to the contract back to users to avoid users overpaying the contract. The code snippet below handles both the transfer and the refund if excess ether is transferred.

- Recommendation (b)



After the above logic has been implemented there should be a logic that transfers the excess of the money sent to the contract back to users to avoid users overpaying the contract. The code snippet below handles both the transfer and the refund if excess ether is transferred.

Status: Resolved

MEDIUM SEVERITY ISSUES

- **Issue: event emission outside of a loop after a lot of state changes has been made.**
  - **Context**: line 376 and 387.
  - **Description**: The Event emission is done outside of the loop which has no significant effect on the overall state changes made, more than one addresses are added and only one event is emitted to show the state changes made. The event emitted is also not significant to the state changes made as it emits a boolean value.
  - The name of the event emitted is not conventionally written properly.

Recommendation: The emission of the event should be done inside of the loop when each address is whitelisted, the emission of the event should also include the address whitelisted in that loop and the name of the event should be written properly to avoid unreadable code.

Status: Resolved

LOW SEVERITY ISSUES

- **Issue: Invalid return value for some function call.**
  - Context: Line 119; 144

```
function publicMint(uint256 quantity, bool _status)
    external
    payable
    returns (bool status)
{
    supply = totalSupply();
    require(quantity > 0, "mint 1");
    require(!pause, "Is Pause");
    require(whitelistMintEnabled == false , "not public period");
    require(supply + quantity <= TOTAL_COLLECTION_SUPPLY, "max exceded");
    if (_status) {
        require(
            IERC20(usdc).transferFrom(
                msg.sender,
                address(this),
                costPublicSale * quantity
            ),
            "must pay cost"
        );
    } else {
        (bool sent, ) = payable(msg.sender).call{
            value: calPublicPriceEth() * quantity
        }("");
        require(sent, "Failed to send Ether");
    }
    _safeMint(msg.sender, quantity);
    emit pubMunt(quantity, msg.sender);
    return status;
}
```

```
function withdrawETH(address payable account)
    external
    payable
    onlyRole(ADMIN_ROLE)
    returns (bool success)
{
    require(account != address(0), "not zero address");
    (bool sent, ) = account.call{value: address(this).balance}("");
    require(sent, "Failed to send Ether");
    return success;
}
```

  - Description: These are programmed to return the status or success of a mint. However, the variable status and success are not being set within the function to help users know the success or failure of calling the function. When a user successfully mints, the return value will still be falsy. The functions are as follows:

- InvestorWhiteListMint
- publicMint
- WhiteListMint
- withdrawETH
- Scenario: If a user calls any of the functions above, and it was successful, the user gets a false as a returned value. This will make the user question if the call was indeed successful and will therefore create a doubt.

Recommendation:
From the codebase, these functions are intended to make the users know the success of these function calls that is why they all return a boolean value. While this is the principal reason, the returned value according to the codebase will always return a **false** value because these variables are not being set within the functions.

Status: Resolved

- **Issue: Missing Check for Address Zero**
  - Context: Line 368;
  - Description: The **setWhitelistTrue/set** aids in the addition of addresses recognized to be whitelisted afterwards, making it a critical function. If Moderator unknowingly passes in an array of addresses with address zero, it will sets address zero true which could deter the total supply intended for the project.

Recommendation:
Add a check to prevent the addition of address zero so as to prevent any disruption that could deter the motives of adding various addresses to the whitelist.

Status: Resolved

INFORMATIONAL SEVERITY ISSUES

- **Issue: Change Function Visibility**
  - **Context**: Line 368; Line 378.
  - **Description**: This set of functions are created to set the state of whitelisted addresses. This implies that it will make modifications to state variables and not be called within the contract. On a leveling scale of how much gas is paid, external functions are cheaper than public functions visibility. The functions are as follows:
  - setWhitelistTrue
  - setWhitelistFalse

Recommendation:
Since these functions are intended to modify the state with no intention of calling it within the contract, it is recommended that these functions have the external visibilities to minimize the cost of calling these functions.

Status: Resolved

- **Issue: Unused state variable**
  - **Context**:Line 47
  - **Description**:The state variable **maxMintAmountPerAddr** has no use to the code. Although it is allowed, it is best practice to avoid unused variables. Unused variables can lead to a few different problems:
  - Increase in computations (unnecessary gas consumption)
  - Indication of bugs or malformed data structures
  - Decreased code readability

Recommendation:
Remove unused variables from the code to avoid unnecessary hike in gas during deployment.

Status: Resolved

CLOSING SUMMARY

There were discoveries of some high, medium, low and informational issues after the audit. The audit team thereafter suggested some remediation to help remedy the issues found in the contract.

APPENDIX

- *Audit: The review and testing of contracts to find bugs.*
- *Issues: These are possible bugs that could lead exploits or help redefine the contracts better.*
- *Slither: A tool used to automatically find bugs in a contract.*
- *Severity: This explains the status of a bug.*

DISCLAIMER

While the audit report is aimed at achieving a quality codebase with assured security and correctness, it should not be interpreted as a guide or or recommendation for people to invest in **Web3Bridge** contracts.

With smart contract audit being a multifaceted process, we admonish the **Web3Bridge** team to carry out further audit from other audit firms or provide a bug bounty program to ensure that more critical audit is done to the contract.

GUILD AUDITS

Guild Audits is geared towards providing blockchain and smart contract security in the fuming web3 world. The firm is passionate about remedying the constant hacks and exploits that deters the web3 motive.