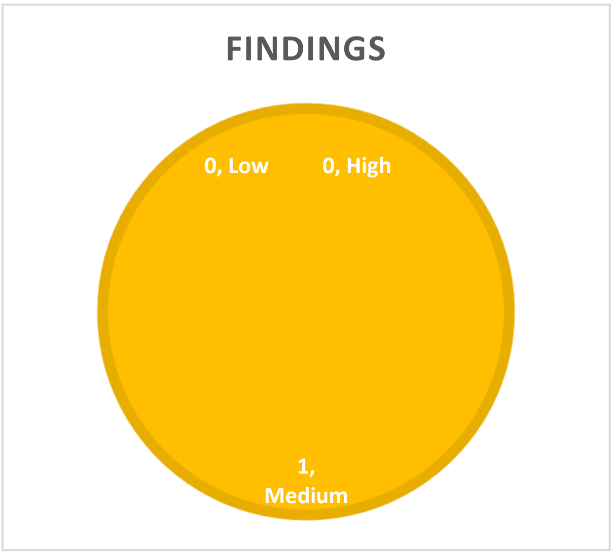
Trust Security

Smart Contract Audit

Mozaic HopliteNFT

01/02/24

Executive summary

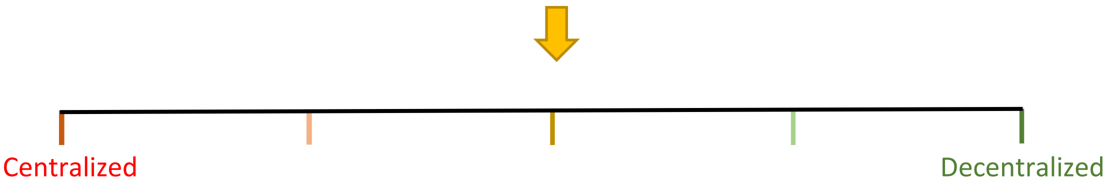


Category	NFT
Audited file count	1
Lines of Code	82
Auditor	Trust
Time period	27-30/01/24

Findings

Severity	Total	Fixed	Acknowledged
High	0	-	-
Medium	1	1	-
Low	0	-	-

Centralization score



Signature

Digitally signed by Trust
Date: 2024.02.01 22:34:57 +02'00'

Document properties

Versioning

Version	Date	Description
0.1	30/01/24	Client report
0.2	01/02/24	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- HopliteNFT.sol

Repository details

- **Repository URL:** https://github.com/Mozaic-fi/Hoplite-NFT
- **Commit hash:** b6d3dc2b90974846415493f06064eb5f3c56af53
- **Mitigation commit hash:** a8599dbf7955c7d72c7c741d55f5bda43a58851e

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

About the Auditors

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys assessing bounty contests in C4 as a Supreme Court judge.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Excellent**	Project kept code as simple as possible, reducing attack risks
Documentation	**Good**	Project is mostly very well documented.
Best practices	**Good**	Project mostly adheres to industry standards.
Centralization risks	**Moderate**	The owner is able to affect significant functionality of the protocol.

Findings

Medium severity findings

TRST-M-1 HopliteNFTs do not support per-token royalties

- **Category:** Logical flaws
- **Source:** HopliteNFT.sol
- **Status:** Fixed

Description

HopliteNFTs conform to the ERC2981 and inherit from Open Zeppelin's ERC2981 contract. By design, it allows NFTs to set the default royalty fee through *_setDefaultRoyalty()* and specific tokenID royalties through *_setTokenRoyalty()*.

Both functions are internal, the first one is exposed by Hoplite to the owner's control through the function below:

```
function adjustRoyalty(uint96 newRoyalty) public onlyOwner {
    require(newRoyalty <= MAX_ROYALTY, "Too high royalty");
    require(royaltyHandler != address(0), "Set the royaltyHandler");
    _setDefaultRoyalty(royaltyHandler, newRoyalty);
    emit NewRoyalty(newRoyalty);
}
```

The issue is that per-token royalty is never exposed. As a result, all tokens will have the same fee. Through discussions with the client this was highlighted to be a requirement of the NFT.

Recommended mitigation

Introduce an *onlyOwner* function which will perform the *_setTokenRoyalty()* call.

Team response

Fixed.

Mitigation review

Additional functions like *setTokenRoyalty()* from ERC2981.sol are now exposed correctly.

Additional recommendations

TRST-R-1 Refactor unchecked blocks

The code uses unchecked blocks when there are no arithmetic operations. In these instances, there is no effect to wrapping a line in such blocks.

```
unchecked {
    whiteList[_whiteListUsers[i]] = false;
}
```

TRST-R-2 Remove unused code

The code snippets below were observed to never be used. It is recommended to remove them.

```
error TooHigh();
```

```
modifier onlyRoyaltyHandler() {
    require(msg.sender == royaltyHandler, "caller must be
royaltyHandler.");
    _;
}
```

TRST-R-3 Improve synchronization of data

The owner can change the **royaltyHandler**, which will receive royalties for the NFT.

```
function setRoyaltyHandler(address _handler) public onlyOwner {
    require(_handler != address(0), "Invalid handler address");
    royaltyHandler = _handler;
}
```

Despite the handler being updated, it will not actually have an effect until *adjustRoyalty()* is called, which triggers the underlying *_setDefaultRoyalty()* function. It would be easy for developers to assume the second call is unnecessary when the royalty fee percentage remains unchanged.

TRST-R-4 Improve validation of input

The whitelist is managed by the two functions below.

```
function updateWhiteList(address[] memory _whiteListUsers) external
onlyOwner {
    require(_whiteListUsers.length > 0, "Invalid Param");
    for(uint i=0; i<_whiteListUsers.length; i++) {
        require(_whiteListUsers[i] != address(0), "Invalid Address");
        unchecked {
            whiteList[_whiteListUsers[i]] = true;
        }
    }
}

function removeWhiteList(address[] memory _whiteListUsers) external
onlyOwner {
    require(_whiteListUsers.length > 0, "Invalid Param");
    for(uint i=0; i<_whiteListUsers.length; i++) {
        require(_whiteListUsers[i] != address(0), "Invalid Address");
        unchecked {
            whiteList[_whiteListUsers[i]] = false;
        }
    }
}
```

It is observed that it never validates that each operation actually flips the status of the target user. Presumably it would be an error to do so, so it is recommended to check for it to detect configuration mistakes.

TRST-R-5 Improve protocol visibility

Consider emitting additional events to improve the transparency of the protocol, for the following functions:

```
function setBaseURI(string memory baseURI) public onlyOwner {
    baseTokenURI = baseURI;
}

function setRoyaltyHandler(address _handler) public onlyOwner {
    require(_handler != address(0), "Invalid handler address");
    royaltyHandler = _handler;
}

function updateGoLiveDate(uint256 _newLiveDate) external onlyOwner {
    goLiveDate = _newLiveDate;
}
```

Centralization risks

TRST-CR-1 Censorship-related risks

The transfer of NFTs can be restricted by the owner through two vectors:

- Adding and removing of users from the whitelist
- Setting the **goLiveDate** timestamp

A possible improvement would be to only be able to set the **goLiveDate** *before* the protocol is live.

TRST-CR-2 LayerZero-related risks

The ONFT owner has significant permissions relating to the bridging-layer properties. In case of a compromised account, many issues can manifest, not limited to:

- Shutdown of the cross-chain functionality.
- Rogue minting of NFTs on the local chain by forging of incoming cross-chain messages.

Systemic risks

TRST-SR-1 Royalty mechanism assumes marketplace honors it

The NFT contract does not actually enforce the payment of royalties. This property is similar to most ERC2981-conforming contracts, but it should still be noted that the operation relies on the good faith of marketplaces and possible peer-to-peer exchanges.

TRST-SR-2 Layer Zero integration risks

The NFT contract relies on the underlying Layer Zero network to relay messages between chains. In the event a Layer Zero compromise or hack occurs, it could enable malicious minting of tokens.