#HashEx
BLOCKCHAIN SECURITY

Nox Beyond

smart contracts final audit report

November 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Nox Beyond team to perform an audit of their smart contract. The audit was conducted between 30/10/2022 and 02/11/2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

This project also had an audit from the Macro team, which can be found at thirdweb's website.

The code is available at 0x1cCb1486f2087809427a6ea52C00AD00e591A02d contract address in AVAX testnet.

2.1 Summary

Project name	Nox Beyond
URL	https://www.noxbeyond.com
Platform	Avalanche Network
Language	Solidity

2.2 Contracts

Name	Address
SignatureDrop.sol	
Royalty.sol	
ERC2771ContextUpgradeable.sol	
DropSinglePhase.sol	
Others contracts	

3. Found issues

High	1 (11%)
Medium	3 (33%)
Low	3 (33%)
Info	2 (23%)

C1. SignatureDrop.sol

ID	Severity	Title	Status
C1-01	High	Exaggerated owner's right	Open
C1-02	Medium	Royalties are not set correctly	Open
C1-03	Medium	Locked AVAX	Open
C1-04	Low	Gas optimization	Open
C1-05	Info	_disableInitializers() function	Open
C1-06	Info	Minted tokens may not have URI	Open

C2. Royalty.sol

ID	Severity	Title	Status
C2-01	Medium	DEFAULT_ADMIN_ROLE can change platform percent	Open

C3. ERC2771ContextUpgradeable.sol

ID	Severity	Title	Status
C3-01	Low	Non-compliance with OZ contract	Open

C4. DropSinglePhase.sol

ID	Severity	Title	Status
C4-01	Low	Only EOA can claim	Open

4. Contracts

C1. SignatureDrop.sol

Overview

ERC-721 standard implementation built on ERC721A from @chiru-labs. The token is modified with the ERC-2981 royalty standard and gas-optimized base URI setting processed in token batches. URI for the token range could be passed encrypted and revealed later by the address authorized with the minter role. To decrypt URI the same key is required that was used for encryption. If the key was lost or corrupted encrypted data was passed, the real token URI may be impossible to know. The main contract in the project inheritance scheme. The token also supports ERC-2771, under the hood it uses a trusted Forwarder address that can forge the value of `_msgSender()` and send transactions from any address. If the owner's account is compromised an attacker acquires unlimited access to the contract and can simulate all admin's and user's actions. A more detailed description of the risks can be found in the Security Considerations section of EIP-2771.

Issues

C1-01 Exaggerated owner's right

● High ⓘ Open

The token transfers can be stopped for users who are not granted `transferRole`. The pausable mechanism has custom implementation and is managed via the presence of `transferRole` on `address(0)`. If `transferRole` is revoked from `address(0)` NFT transfers halt for users without the privilege except from and to `address(0)` which means mint and burn operations respectively.

```
function _beforeTokenTransfers(
    address from,
    address to,
    uint256 startTokenId,
    uint256 quantity
```

```
) internal virtual override {
    super._beforeTokenTransfers(from, to, startTokenId, quantity);

    // if transfer is restricted on the contract, we still want to allow burning and
minting
    if (!hasRole(transferRole, address(0)) && from != address(0) && to != address(0)) {
        if (!hasRole(transferRole, from) && !hasRole(transferRole, to)) {
            revert("!Transfer-Role");
        }
    }
}
```

Recommendation

We recommend removing the `Pausable` functionality or resetting `DEFAULT_ADMIN_ROLE` to zero after the sale ends. Before the end of the sale, `DEFAULT_ADMIN_ROLE` must be a `Timelock` contract with a minimum delay of at least 24 hours and `MultiSig` as admin. This won't stop the owner from possible rights abuse during a sale period, but it will help users to be informed about upcoming changes.

C1-02 Royalties are not set correctly

Medium Open

Lazy minted tokens can be issued with the `mintWithSignature()` method, that takes a `_req` parameter which also includes royalties information. However, the royalties percentage and receiver are set only for the first token to be minted, while the rest of issued tokens will have royalties info absent and their author won't be cherished unless the royalty info is set manually by admin.

```
function mintWithSignature(MintRequest calldata _req, bytes calldata _signature)
    external
    payable
    returns (address signer)
{
    uint256 tokenIdToMint = _currentIndex;
    ...
    if (_req.royaltyRecipient != address(0) && _req.royaltyBps != 0) {
        _setupRoyaltyInfoForToken(tokenIdToMint, _req.royaltyRecipient, _req.royaltyBps);
```

```
    }

    // Mint tokens.
    _safeMint(receiver, _req.quantity);

    emit TokensMintedWithSignature(signer, receiver, tokenIdToMint, _req);
}
```

Recommendation

We recommend making a similar system as an encrypted data setting in the Royalty contract, royalty information should be set for a batch of id tokens, it will solve the problem gas-efficiently and without a lot of code rewriting.

C1-03 Locked AVAX

Medium Open

The `mintWithSignature()` function can accept native EVM currency, i.e. AVAX, as payment. In a situation where AVAX was attached, but the `_currency` parameter not equal to `NATIVE_TOKEN` was passed to the `collectPriceOnClaim()` function, the attached AVAX can not be returned and will be locked on the contract.

```
function mintWithSignature(MintRequest calldata _req, bytes calldata _signature)
        external
        payable
        returns (address signer)
    {
        uint256 tokenIdToMint = _currentIndex;
        if (tokenIdToMint + _req.quantity > nextTokenIdToLazyMint) {
            revert("Not enough tokens");
        }

        // Verify and process payload.
        signer = _processRequest(_req, _signature);

        address receiver = _req.to;

        // Collect price
        collectPriceOnClaim(_req.primarySaleRecipient, _req.quantity, _req.currency,
_req.pricePerToken);
```

```
        // Set royalties, if applicable.
        if (_req.royaltyRecipient != address(0) && _req.royaltyBps != 0) {
            _setupRoyaltyInfoForToken(tokenIdToMint, _req.royaltyRecipient,
_req.royaltyBps);
        }

        // Mint tokens.
        _safeMint(receiver, _req.quantity);

        emit TokensMintedWithSignature(signer, receiver, tokenIdToMint, _req);
    }

function collectPriceOnClaim(
        address _primarySaleRecipient,
        uint256 _quantityToClaim,
        address _currency,
        uint256 _pricePerToken
    ) internal override {
        if (_pricePerToken == 0) {
            return;
        }

        (address platformFeeRecipient, uint16 platformFeeBps) = getPlatformFeeInfo();

        address saleRecipient = _primarySaleRecipient == address(0) ?
primarySaleRecipient() : _primarySaleRecipient;

        uint256 totalPrice = _quantityToClaim * _pricePerToken;
        uint256 platformFees = (totalPrice * platformFeeBps) / MAX_BPS;

        if (_currency == CurrencyTransferLib.NATIVE_TOKEN) {     // !!!
            if (msg.value != totalPrice) {
                revert("Must send total price");
            }
        }

        CurrencyTransferLib.transferCurrency(_currency, _msgSender(),
platformFeeRecipient, platformFees);
        CurrencyTransferLib.transferCurrency(_currency, _msgSender(), saleRecipient,
totalPrice - platformFees);
    }
```

Recommendation

At the end of the `mintWithSignature()` function, you need to return all the funds that were not spent on the payment or add `require(msg.value == 0)` to the function if `_currency` is not a native currency.

C1-04 Gas optimization

Low Open

`transferRole` and `minterRole` state variables should be declared as constants.

C1-05 _disableInitializers() function

Info Open

The contract that inherits the Initializable contract should implement a constructor that calls the `_disableInitializers()` function. An uninitialized contract can be taken over by an attacker and maliciously initialized. To prevent this, you should invoke the `_disableInitializers()` function in the constructor to automatically lock it when it is deployed.

C1-06 Minted tokens may not have URI

Info Open

The number of available tokens for issuing during claims is not limited by the number of tokens with a set URI. If `_currentIndex` exceeds `nextTokenIdToLazyMint` token ids between `nextTokenIdToLazyMint` and `_currentIndex` won't have URI until they are lazy-minted.

```
function lazyMint(
    uint256 _amount,
    string calldata _baseURIForTokens,
    bytes calldata _data
) public override returns (uint256 batchId) {
    if (_data.length > 0) {
        (bytes memory encryptedURI, bytes32 provenanceHash) = abi.decode(_data, (bytes,
bytes32));
        if (encryptedURI.length != 0 && provenanceHash != "") {
            _setEncryptedData(nextTokenIdToLazyMint + _amount, _data);
        }
    }
```

```
    return super.lazyMint(_amount, _baseURIForTokens, _data);
}

function transferTokensOnClaim(address _to, uint256 _quantityBeingClaimed)
    internal
    override
    returns (uint256 startTokenId)
{
    startTokenId = _currentIndex;
    _safeMint(_to, _quantityBeingClaimed);
}
```

C2. Royalty.sol

Overview

This contract provides functions for setting and reading the recipient of the royalty fee and the royalty fee basis points.

Issues

C2-01 DEFAULT_ADMIN_ROLE can change platform percent

Medium Open

Account granted with `DEFAULT_ADMIN_ROLE` can change the platform percent fee to zero or change the fee receiver to an arbitrary address. This may left platform creators without fees. However, It is not clear whether this is an issue or not since nowhere in the documentation is it indicated who has `DEFAULT_ADMIN_ROLE`.

```
function setDefaultRoyaltyInfo(address _royaltyRecipient, uint256 _royaltyBps) external
override {
    if (!_canSetRoyaltyInfo()) {
        revert("Not authorized");
    }
```

```
    _setupDefaultRoyaltyInfo(_royaltyRecipient, _royaltyBps);
}

function _setupDefaultRoyaltyInfo(address _royaltyRecipient, uint256 _royaltyBps) internal
{
    if (_royaltyBps > 10_000) {
        revert("Exceeds max bps");
    }

    royaltyRecipient = _royaltyRecipient;
    royaltyBps = uint16(_royaltyBps);

    emit DefaultRoyalty(_royaltyRecipient, _royaltyBps);
}
```

Recommendation

Provided proper documentation for the `DEFAULT_ADMIN_ROLE` authorization model.

C3. ERC2771ContextUpgradeable.sol

Overview

Context variant with ERC2771 support.

Issues

C3-01 Non-compliance with OZ contract

Low Open

Original OpenZeppelen ERC2771ContextUpgradeable has only one `trustedForwarder` address that is usually easy to monitor. The project's version supports many such addresses that are stored in mapping but can still be set only during initialization and can't be modified. Users may experience difficulties obtaining the full list of `trustedForwarder` addresses as the mapping has private visibility and no getter function is provided.

```
abstract contract ERC2771ContextUpgradeable is Initializable, ContextUpgradeable {
```

```
    mapping(address => bool) private _trustedForwarder;

    function __ERC2771Context_init(address[] memory trustedForwarder) internal
onlyInitializing {
        __Context_init_unchained();
        __ERC2771Context_init_unchained(trustedForwarder);
    }

    function __ERC2771Context_init_unchained(address[] memory trustedForwarder) internal
onlyInitializing {
        for (uint256 i = 0; i < trustedForwarder.length; i++) {
            _trustedForwarder[trustedForwarder[i]] = true;
        }
    }
    ...
    uint256[49] private __gap;
}
```

C4. DropSinglePhase.sol

Overview

The contract adds airdrop functionality to the token. An address can't have more than one claim per drop round.

Issues

C4-01 Only EOA can claim

Low Open

Claims are available only for EOA, claims called by contracts are considered bots.

C5. Others contracts

Overview

This section relates to IERC165.sol, IERC20.sol, IERC2981.sol,BatchMintMetadata.sol, ContractMetadata.sol, DelayedReveal.sol, LazyMint.sol, Ownable.sol, Permissions.sol, PermissionsEnumerable.sol, PlatformFee.sol, PrimarySale.sol, SignatureMintERC721Upgradeable.sol, IClaimCondition.sol, IContractMetadata.sol, IDelayedReveal.sol, IDropSinglePhase.sol, ILazyMint.sol, IOwnable.sol, IPermissions.sol, IPermissionsEnumerable.sol, IPlatformFee.sol, IPrimarySale.sol, IRoyalty.sol, ISignatureMintERC721.sol, IWETH.sol, CurrencyTransferLib.sol, MerkleProof.sol, TWAddress.sol, TWBitMaps.sol, TWStrings.sol, SafeERC20.sol, IERC2981Upgradeable.sol, Initializable.sol, IERC721ReceiverUpgradeable.sol, IERC721Upgradeable.sol, IERC721MetadataUpgradeable.sol, AddressUpgradeable.sol, ContextUpgradeable.sol, MulticallUpgradeable.sol, StringsUpgradeable.sol, ECDSAUpgradeable.sol, draft-EIP712Upgradeable.sol, ERC165Upgradeable.sol, IERC165Upgradeable.sol, ERC721AUpgradeable.sol, IERC721AUpgradeable.sol contracts. No issues were found.

5. Conclusion

1 high, 3 medium, 3 low severity issues were found during the audit. No issues were resolved in the update.

The reviewed contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured. Note also that it is possible that the tokens will be with an empty URI.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY