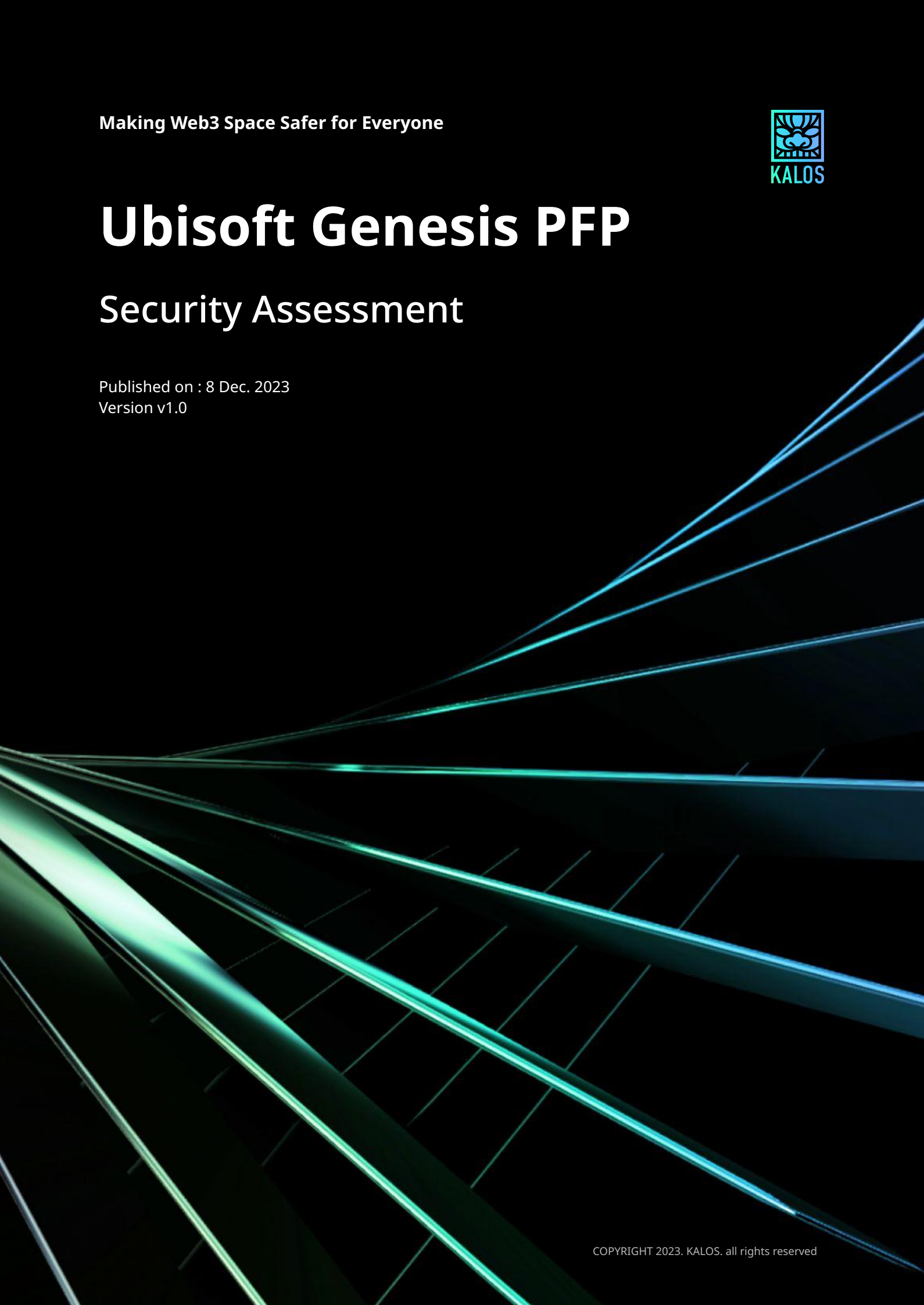
Making Web3 Space Safer for Everyone

KALOS

Ubisoft Genesis PFP

Security Assessment

Published on : 8 Dec. 2023
Version v1.0



Security Report Published by KALOS

v1.0 8 Dec. 2023

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	-	-	-	-
Medium	-	-	-	-
Low	-	-	-	-
Tips	3	-	3	-

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

KALOS is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

Having secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges, KALOS is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the Genesis PFP NFT contracts. KALOS conducted the audit focusing on whether the system created by the Ubisoft team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of all. In detail, we have focused on the followings

- Unintended behavior on the NFT minting and random revealing process
- Manipulation of critical variables due to incorrect access control settings
- Proper implementation of ERC721 specifications contract
- Existence of known smart contract vulnerabilities
- Signature replay.

Codebase Submitted for the Audit

The codes used in this Audit are delivered to the private repository.

The last commit of the code used for this Audit is “d1f1c17a4bd2ae1df1d86f6fe63b174b125b7e0e”.

Audit Timeline

Date	Event
2023/11/24	Audit Initiation
2023/12/8	Delivery of v1.0 report.

Findings

KALOS found no issues. There are 3 Tips issues explained that would improve the code’s usability or efficiency.

Severity	Issue	Status
Tips	[GENESIS-PFP-01] Lack of Validation	(Acknowledged - v1.0)
Tips	[GENESIS-PFP-02] No Event emission	(Acknowledged - v1.0)
Tips	[GENESIS-PFP-03] Code Quality Improvements	(Acknowledged - v1.0)

OVERVIEW

Functional Description

GenesisPFP

This contract is an NFT contract that uses ERC721Psi, which is designed for gas saving, ERC2981 which is royalty payment information for NFT, EIP712, Ownable, and AccessControl. Its maximum supply is 9,999 and has the roles of Admin and Minter, and its default royalty is 5%. The royalty recipient is vault and the rate can be set later by the owner. The mintWithSignature() function allows users to mint based on a given signature and checks (remaining supply, timestamp, chain ID, minting amount, and duplicate mints). Only signers with the minter role can mint. The tokenURI() returns the metadata URI based on the token ID, which is determined by a specific metadata ID derived using the random seed generated by the Chainlink VRF.

Scope

```
src
├── ERC721Psi
│   ├── ERC721Psi.sol
│   └── extension
│       └── ERC721PsiAddressData.sol
├── GenesisPFP.sol
├── abstracts
│   ├── ChainlinkVRFMetadata.sol
│   └── GenesisBase.sol
├── interfaces
│   ├── IGenesisBase.sol
│   └── IGenesisPFP.sol
├── librairies
│   └── Errors.sol
└── types
    └── MintData.sol
```

Access Controls

The admin can grant the Admin, Minter roles. The admins can grant and revoke roles. The Owner can request random seed from Chainlink VRF, transfer remaining LINK tokens, set the base URI for tokens' metadata, and update the default royalty rate. The Minter role as a signer can generate a signature for NFTs minting.

- Admin role

- GenesisBase.grantRole, grantRoles, and revokeRoles: Grant or revoke other roles.

- Owner

- ChainlinkVRFMetadata.requestChainlinkVRF(): Request a random seed from the Chainlink VRF
- ChainlinkVRFMetadata.withdrawRemainingLink(): Transfer the remaining LINK tokens to the owner
- GenesisBase.setBaseURI(): Set the base URI for token’s metadata URI
- GenesisBase.updateDefaultRoyalty(): Update default royalty rate. It allows the owner to change royalty instead of the initial 5%.

- Minter role

- GenesisPFP.mintWithSignature(): Mint NFT with signature. The signer is supposed to have the Minter role.

Notice

NFTs can only be minted using a signature generated off-chain. The data for minting, including the signature validity period, chain ID, mint amount, and user nonce, are determined off-chain. There is no limit on the mint amount, and the user nonce is flexible and does not have constraints. Although the user nonce can prevent duplicate mints, we were not provided with the logic for generating it. The team replied that there would be no issues with their back-end infrastructure that generates the signature. Therefore, the process of generating the signature and mint data is out of the scope of this audit. The team replied they would carefully manage their signer key and the infrastructure.

FINDINGS

Lack of Validation

ID: GENESIS-PFP-01
Severity: Tips
Type: Input Validation
Difficulty: High
File: src/abstracts/ChainlinkVRFMetadata.sol

```
    function requestChainlinkVRF(uint32 _callbackGasLimit, uint16 _requestConfirmations) external
onlyOwner {
        if (chainlinkRequestID != 0) {
            revert Errors.RequestAlreadyInitialized();
        }
        // 150_000 gas should be more than enough for the callback
        // 6 block confirmations or more
        // 1 random number used a a seed for the tokenIDs
        chainlinkRequestID = requestRandomness(_callbackGasLimit, _requestConfirmations, 1);
    }
// ...
    function fulfillRandomWords(uint256 _requestId, uint256[] memory _randomWords) internal override {
        if (_requestId != chainlinkRequestID || _randomWords.length != 1) {
            chainlinkRequestID = 0;
            return;
        }
        // Assign the retrieved random word from Chainlink VRF V2
        // Anyone can re-generate the tokenIds from the seed deterministically
        chainlinkSeed = _randomWords[0];
    }
```

[ChainlinkVRFMetadata.sol]

If chainlinkRequestID is set to zero in fulfillRandomWords, requestChainlinkVRF can retry requesting a random seed from the Chainlink. It allows requestChainlinkVRF to be able to request again, even if chainlinkSeed is already fulfilled.

This is up to the Chainlink and rare scenario that fulfills with a wrong request id after fulfilled.

Recommendation

Revert in requestChainlinkVRF() if the seed is already fulfilled.

Fix Comment

The team acknowledged and will not fix this issue.

No Event emission

ID: GENESIS-PFP-02
Type: Visibility
File: src/abstracts/GenesisBase.sol

Severity: Tips
Difficulty: Low

The default royalty update function does not emit an event. If an event is not emitted, it is difficult to monitor royalty change on-chain when the owner changes value.

```
    function updateDefaultRoyalty(address receiver, uint96 feeNumerator) external onlyOwner {
        _setDefaultRoyalty(receiver, feeNumerator);
    }
```

[GenesisBase.sol]

Recommendation

Emit event including new fee rate.

Fix Comment

The team acknowledged and will not fix this issue.

Code Quality Improvements

ID: GENESIS-PFP-03
Type: N/A
File: src/GenesisPFP.sol

Severity: Tips
Difficulty: N/A

This section can improve code quality and save gas consumption.

Description and Recommendations

```
    function hashTypedDataV4(MintData memory mintData) public view returns (bytes32) {
        return _hashTypedDataV4(hashStruct(mintData));
    }

    function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
        if (!_exists(tokenId)) revert Errors.ERC721UriNonExistent();

        // Return empty tokenURI if metadata CID isn't registered
        if (bytes(_baseURI()).length == 0) {
            return "";
        }

        // Return a fallback URI if reveal isn't called yet
        if (chainlinkSeed == 0) {
            return string(abi.encodePacked(_baseURI(), "default.json"));
        }

        uint256 metadataId = ((tokenId + chainlinkSeed) % 9999) + 1;
        return string(abi.encodePacked(_baseURI(), metadataId.toString(), ".json"));
    }
```

[GenesisPFP.sol]

hashTypedDataV4() external view function is unnecessary and can be removed for gas savings at deployment time.

tokenURI() calls _baseURI() many times. By storing baseURI as a local variable, it can save gas.

Fix Comment

The team acknowledged and will not fix this issue.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure codes, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	Medium	High
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document