SECURITY AUDIT OF

OWNERS INC. SMART CONTRACT

Public Report

Apr 12, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Flow	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Flow Token	Flow's native currency which is key to maintaining and operating the Flow blockchain. It can be integrated into dApps for payments, transactions and earning rewards
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Cadence	A resource-oriented programming, high-level language for implementing smart contracts for the Flow platform.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Apr 12, 2022. We would like to thank the OWNERS inc. for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the OWNERS inc. Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified no vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About OWNERS inc. Smart Contract

OWNERS inc. goal is to create a new celebrity and fan community.

OWNERS inc. will create a marketplace where anyone can buy and sell in an auction-style by converting famous SNS accounts to NFT. Fans will want to purchase an NFT for a celebrity SNS account, which will be the only one in the world.

Celebrities can choose to include additional bonuses when listing NFTs, increasing the value of NFTs.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the OWNERS inc. Smart Contract.

It was conducted on commit 7ad9aea1e90b1dc2fc35366ddfa962267b63a36c from git repository *https://github.com/OwnersCompany/NFT-Blockchain*

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
96b8f0b04eb96fca923236997a85d8cdaea3999073e12cdd23a5cc6b90e2f47c	**contracts/NonFungibleToken.cdc**
b5425a7145eea45a302f7efa858bb4b911b2cedd7e8c3d17905c480dca719e68	**contracts/Owners.cdc**
44d15c2341a608e514bb9c386dfd42c79bdcb360be1a35ea1d144cacc32ea90c	**transactions/owners/create_operator_capability.cdc**
a819e3c952dd5b9ba7c02e1fc4d96fa79ab1e80f6bba5bd0a3425e15ab66a477	**transactions/owners/extend_expiry_time.cdc**
6d29a3cc4cd5c3ac75f571cfa57d03907400f898f09dad9898f873856d4f5252	**transactions/owners/mint_owners.cdc**
c781995de35bb2a8d9e4ca754567adb04018ee61edbdd02dd195a7ff031224e9	**transactions/owners/operator_mint_owners.cdc**
420adb0f25c94c50d9c978345fe7264773d92cbd01d71f7cea7b76f6ac2f6f3f	**transactions/owners/operator_transfer_owners.cdc**
eb3819ae8146e0961c26240a9d4b5a4bb11658b2724213cd46965f9a83b1a880	**transactions/owners/revoke_minter_capability.cdc**

ad062ead1fb264e86b9a9e1e394e257ac86d2d40eff8de69d2a83a0577a51e44	**transactions/owners/revoke_transfer_capability.cdc**
c1303f4fb99ec10ff10ecd9297443b4270a326af1c4eed179a5add00c419e4c6	**transactions/owners/setup_account.cdc**
9d046a95397e7d686aee33a8eef4f1ebe743d377386cb5054c6091f6d4234216	**transactions/owners/setup_operator.cdc**
c080f40f9fa6a19748a979fd21b2ba551ef9de82fc7e98a5a1af6897a734b741	**transactions/owners/transfer_owners.cdc**

1.3. Audit methodology

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Access Modifier
- Explicit visibility of functions state variables
- Capability Access Control
- Unsafe type Inference
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The OWNERS inc. Smart Contract was written in the Cadence language. The contract and transaction source code are referenced from Flow team public source code.

The Owner contract extends the NonFungibleToken interface - the flow NFT interface standard and implements all functions in it. The contract also defines some new resources to create mint and operate logic. One of the new resources is NFTMinter, the contract creates an instance of the NFTMinter resource which is responsible for minting new tokens in the initializer, any operator who wants to create new tokens must borrow its functions. NFTOperator resource is also a new one, the admin of the contract may assign the mint permission of the minter to NFTOperator instance - operator. Otherwise, the operators may be received admin vault transfer permission if they are approved. Admin can revoke all above permissions by unlink capability through revoke transactions at any time.

There are 10 transaction templates that are accompanied by the contract. These templates help the users interact with other instances follow the flow chain ideas - reduce the centralized of the contract.

2.2. Findings

During the audit process, the audit team found no vulnerability in the given version of OWNERS inc. Smart Contract.

2.3. Additional notes and recommendations

2.3.1. Consider to update Minted and add ApproveMint event INFORMATIVE

Currently, the Minted event only emits the ID of the new NFT, it may be useless. If this event also emits the recipient, it may be a convenient for the future development.

```
    pub event ContractInitialized()
    pub event Withdraw(id: UInt64, from: Address?)
    pub event Deposit(id: UInt64, to: Address?)
    pub event Minted(id: UInt64)

    // Named Paths
```

Snippet 1. Owner.cdc - Consider to update Minted event

Besides, the mint approval action is a significant action, a new operator has permission to mint unlimited new NFTs. We suggest adding an ApproveMinting event to notify a new operator have this permission.

RECOMMENDATION

We suggest updating the contract like the below:

```
    pub event ContractInitialized()
    pub event Withdraw(id: UInt64, from: Address?)
    pub event Deposit(id: UInt64, to: Address?)
    pub event Minted(id: UInt64, recipient: Address?)

    // Named Paths
    ...
        pub fun mintNFT(recipient: &{NonFungibleToken.CollectionPublic}, …
  twitterID: UInt64) {
            // deposit it in the recipient's account using their reference
            recipient.deposit(token: <-create Owners.NFT(initID: Owners.t…
  otalSupply, initTwitterID: twitterID))
            emit Minted(id: Owners.totalSupply, recipient: recipient.owne…
  r?.address)
            Owners.totalSupply = Owners.totalSupply + 1
        }
```

Snippet 2. Owner.cdc - Update the Minted event and the function use it

```
    pub event Withdraw(id: UInt64, from: Address?)
    pub event Deposit(id: UInt64, to: Address?)
    pub event Minted(id: UInt64, recipient : Address?)
    pub event ApproveMintting(operator: Address?)
    // Named Paths

        pub fun addMinterCapability(cap: Capability<&NFTMinter>) {
            pre {
                cap.borrow() != nil: "Invalid nft minter capability"
            }
            self.operatorCapability = cap
            emit ApproveMintting(operator: self.owner?.address)
        }
```

Snippet 3. Owner.cdc - Add the ApproveMinting event and update the function use it

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged the **OWNERS inc.**

2.3.2. Best practice in transactions INFORMATIVE

There are some statements that do not interact directly with AuthAccount in the prepare phase, they should be moved to the execute phase.

```
transaction(operatorAddress: Address) {

    prepare(admin: AuthAccount) {

        let operator = getAccount(operatorAddress)
        // Private Path to link minter capacitiy for operator
        // If you need to revoke minter capability from old operator,
        // unlink issued paths and manually hard-code this path and submi…
  t the new transaction
        // The old paths should be added to a blacklist and must not be r…
  eused forever
        let minterPrivatePath = /private/OwnersMinterV1
        let transferPrivatePath = /private/OwnersTransferV1

let capabilityReceiver = operator.getCapability
            <&Owners.NFTOperator{Owners.NFTOperatorPublic}>
            (Owners.OperatorPublicPath)
            .borrow() ?? panic("Could not borrow capability receiver refe…
  rence")

        admin.link<&Owners.NFTMinter>(minterPrivatePath, target: Owners.M…
  interStoragePath)
        admin.link<&{NonFungibleToken.Provider, NonFungibleToken.Collecti…
  onPublic}>(transferPrivatePath, target: Owners.CollectionStoragePath)

        let nftMinterCapability = admin
            .getCapability<&Owners.NFTMinter>(minterPrivatePath)

        let nftTransferCapability = admin
            .getCapability<&{NonFungibleToken.Provider, NonFungibleToken.…
  CollectionPublic}>(transferPrivatePath)

        capabilityReceiver.addMinterCapability(cap: nftMinterCapability)
        capabilityReceiver.addTransferCapability(cap: nftTransferCapabili…
  ty)
    }
```

Snippet 4. create_operator_capability.cdc The statements should be moved to execute phase

```
transaction(recipient: Address, twitterID: String) {

    // local variable for storing the minter reference
    let minter: &Owners.NFTMinter
    var twitterIdNumber: UInt64

    prepare(signer: AuthAccount) {

        // borrow a reference to the NFTMinter resource in storage
        self.minter = signer.borrow<&Owners.NFTMinter>(from: Owners.Minte…
  rStoragePath)
            ?? panic("Could not borrow a reference to the NFT minter")

        // Convert twitterID from String to UInt64
        let utf8s = twitterID.utf8
        var i = utf8s.length
        var number = 0 as UInt64
        while (i > 0) {
            var multiple = 1 as UInt64
            var j = utf8s.length - i
            while (j > 0) {
                multiple = multiple * 10
                j = j -1
            }
            i = i - 1
            number = number + UInt64(utf8s[i] - 48) * multiple
        }
        self.twitterIdNumber = number
    }

    execute {
        // get the public account object for the recipient
        let recipient = getAccount(recipient)

        // borrow the recipient's public NFT collection reference
        let receiver = recipient
            .getCapability(Owners.CollectionPublicPath)!
            .borrow<&{NonFungibleToken.CollectionPublic}>()
            ?? panic("Could not get receiver reference to the NFT Collect…
  ion")

        // mint the NFT and deposit it to the recipient's collection
        self.minter.mintNFT(recipient: receiver, twitterID: self.twitterI…
  dNumber)
    }
}
```

Snippet 5. mint_owners.cdc The statements should be moved to execute phase

```
transaction(recipient: Address, twitterID: String) {

    // local variable for storing the minter reference
    let minter: &Owners.NFTMinter
    var twitterIdNumber: UInt64

    prepare(signer: AuthAccount) {
        // borrow a reference to the NFTOperator resource in storage
        let operator = signer.borrow<&Owners.NFTOperator>(from: Owners.Op…
  eratorStoragePath)
            ?? panic("Could not borrow a reference to the NFT operator")
        // borrow a reference to the NFTMinter resource in storage
        if operator.operatorCapability == nil {
          panic("Operator capability is not set")
        }
        self.minter = operator.operatorCapability!.borrow()
            ?? panic("Could not borrow a reference to the NFT minter")

// Convert twitterID from String to UInt64
        let utf8s = twitterID.utf8
        var i = utf8s.length
        var number = 0 as UInt64
        while (i > 0) {
            var multiple = 1 as UInt64
            var j = utf8s.length - i
            while (j > 0) {
                multiple = multiple * 10
                j = j -1
            }
            i = i - 1
            number = number + UInt64(utf8s[i] - 48) * multiple
        }
        self.twitterIdNumber = number
    }
    execute {
        // get the public account object for the recipient
        let recipient = getAccount(recipient)

        // borrow the recipient's public NFT collection reference
        let receiver = recipient
            .getCapability(Owners.CollectionPublicPath)
            .borrow<&{NonFungibleToken.CollectionPublic}>()
            ?? panic("Could not get receiver reference to the NFT Collect…
  ion")

        // mint the NFT and deposit it to the recipient's collection
        self.minter.mintNFT(recipient: receiver, twitterID: self.twitterI…
  dNumber)
    }
}
```

Snippet 6. operator_transfer_owners.cdc The statements should be moved to execute phase

```
transaction(recipient: Address, withdrawID: UInt64) {
    prepare(signer: AuthAccount) {

        // get the recipients public account object
        let recipient = getAccount(recipient)

        // borrow a reference to the signer's NFT collection
        let collectionRef = signer.borrow<&Owners.Collection>(from: Owner…
  s.CollectionStoragePath)
            ?? panic("Could not borrow a reference to the owner's collect…
  ion")

// borrow a public reference to the receivers collection
let depositRef = recipient.getCapability(Owners.CollectionPublicPath)!.bo…
  rrow < &{NonFungibleToken.CollectionPublic}> ()!

        // withdraw the NFT from the owner's collection
        let nft <- collectionRef.withdraw(withdrawID: withdrawID)
        // Deposit the NFT in the recipient's collection
        depositRef.deposit(token : < - nft)
    }
}
```

Snippet 7. transfer_owners.cdc The statements should be moved to execute phase

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged the **OWNERS inc.**

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 31, 2022*	Public Report	Verichains Lab
1.1	*Apr 12, 2022*	Public Report	Verichains Lab

Table 2. Report versions history