SECURITY AUDIT OF

NEXTVISIONCAPITAL SMART CONTRACTS

Public Report

Oct 05, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Oct 05, 2022. We would like to thank the NextVisionCapital 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 NextVisionCapital Smart Contracts. 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 some vulnerable issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About NextVisionCapital Smart Contracts

NextVisionCapital Smart Contracts are smart contracts which run on BSC blockchain.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of NextVisionCapital. It was conducted on the source code provided by the NextVisionCapital team.

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

SHA256 Sum	File
53155224f65acb1e37f1535b697023ed9d078c740a8ca042c324e8ba5ea88e6c	NextVisionCapital.sol
ed429cb500d0293205317174f4fe2c12edd4bb71b447f1f2075390719959964c	WrapTransactions.sol
a9aff3d76df2b08f2a17be3d513c01a1ec91db5792bd22b6201d74c2ce8921e4	access/Ownable.sol
f4c2e3393254e06c3044ca5bc80c03417bbb8584ba1b914068c5ce90e30d2dc6	extensions/ERC721A.sol
86a0fd15fc4a69eb4cb102482b2c416018692dc9d4c0e685fd43231e31b754bc	extensions/ERC721AQueryable.sol
fb8544a92ed77462745319db0ae46193038c26f1d5297e7c64c6b71489ce2618	extensions/IERC721A.sol
ed4fbd16dacaf1ff84db59c032e564335bdd91c7a1a289fc1701aebf3ed3b71f	extensions/IERC721AQueryable.sol
e99a929d5cff350c85680fc598005a3e7dbc82edce39bdd6ad8a9d8f832b1a51	utils/Counters.sol

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.

- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

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

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- 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 NextVisionCapital Smart Contracts was written in `Solidity` language, with the required version to be `^0.8.4`.

2.1.1. NextVisionCapital contract

This table lists some properties of the audited NextVisionCapital Smart Contracts (as of the report writing time).

PROPERTY	VALUE
Name	NextVisionCapital T100
Symbol	NVC-T100
Max Supply	5

Table 2. The NextVisionCapital Smart Contracts properties

NextVisionCapital contract extends `ERC721AQueryable` and `Ownable`. by default, Token Owner is contract deployer, but he can transfer ownership to another address at any time.

The contract implements `safeMintTo` public function which allows `owner` to mint new tokens for any address account in the max supply range.

2.1.2. WrapTransactions contract

A support contract which allows users to scatter ether/ERC20 tokens to a list of another users.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of NextVisionCapital Smart Contracts.

2.2.1. WrapTransaction.sol - The `scatterTokens` function can't work CRITICAL

The `scatterToken` function uses the `transferFrom` method which has a mistake in the return data definition. According to EIP20, the `transferFrom` function only returns one value but in the `IERC20` interface, it defines two. Therefore, the `scatterToken` is always reverted when this method is triggered.

```
function scatterTokens(
        IERC20 token,
        address[] memory recipients,
        uint256[] memory values,
        bool revertOnfail
    ) external onlyOwner {
        uint256 totalSuccess = 0;

        for (uint256 i = 0; i < recipients.length; i++) {
            (bool success, bytes memory returnData) = token.transferFrom(msg.sender,
recipients[i], values[i]); //<- Triggered here
        ...
```

More information about EIP-20:

- *https://eips.ethereum.org/EIPS/eip-20*

RECOMMENDATION

The team should update the interface and rewrite the `scatterToken` function.

UPDATES

- *Aug 31, 2022*: This issue has resolved.

2.2.2. WrapTransaction.sol - Attacker can transfer native tokens from the contract without any limitation CRITICAL

The `scatterEthers` function is used to scatterEthers from caller to multi recipients but the function doesn't check the `msg.value`, the contract balance can be drained by the attacker.

The attacker only calls this function with total values lower than his balance to trigger the contract to transfer its native tokens to the recipients.

```
function scatterEthers(
        address []  memory recipients,
        uint256[] memory values,
        bool revertOnfail)
        external payable {
        uint256 totalSuccess = 0;
        uint256 totalTokens  = 0;
        for (uint256 i = 0; i < values.length; i++){
            totalTokens = totalTokens + values[i];
        }
        require(totalTokens <= (msg.sender).balance,"not enough balance");
        for (uint256 i = 0; i < recipients.length; i++) {
            (bool success,) =  recipients[i].call{value: values[i],gas:300000}("");  //<--
Transfer native token from contract to recipients

            if (!success) {
                require(revertOnfail, "One of the transfers failed");
                emit TransferFailed(recipients[i], values[i]);
            }
            else{
                totalSuccess++;
            }
        }

        require(totalSuccess >= 1, "all transfers failed");
        }
```

The attacked transaction of your contract on testnet:

- *https://testnet.bscscan.com/tx/0x961055ff3811f61beae15aaa6ac5d6db6c77207cab3f8cd23ed3aa680cfeb1d6*

RECOMMENDATION

Update the function like the code below:

```
function scatterEthers(
        address []  memory recipients,
        uint256[] memory values,
        bool revertOnfail)
        external payable {
        uint256 totalSuccess = 0;
        uint256 totalTokens  = 0;
        for (uint256 i = 0; i < values.length; i++){
            totalTokens = totalTokens + values[i];
        }
        require(totalTokens <= msg.value ,"not enough balance");
        for (uint256 i = 0; i < recipients.length; i++) {
            (bool success,) =  recipients[i].call{value: values[i],gas:300000}("");

            if (!success) {
                require(revertOnfail, "One of the transfers failed");
                emit TransferFailed(recipients[i], values[i]);
            }
            else{
                totalSuccess++;
            }
        }

        require(totalSuccess >= 1, "all transfers failed");
        }
```

UPDATES

- *Oct 04, 2022*: This issue has resolved.

2.3. Additional notes and recommendations

2.3.1. NextVisionCapital.sol - Mismatch data constant with comment. INFORMATIVE

In `NextVisionCapital` contract, the `PRICE` constant is declared with `1000 ether` value, it's corresponding to `1000 BUSD` with the logic in the `safeMint` function (if `ERC20` is `BUSD`). But it is commented `1 BUSD` in the declare statement. To avoid a mistake, we note this issue to warn the team.

```
uint256 public constant PRICE = 1000 ether; //1 BUSD  //<- May be a mistake

    function safeMint(uint256 _quantity) external payable {
        require(_quantity > 0, "Quantity must be greater than 0.");

        require(
            totalSupply() + _quantity <= COLLECTION_SIZE,
            "Cannot mint over supply cap"
        );

        require(
            transferERC20(msg.sender, PRICE * _quantity),
            "Fail to transfer token."
        );

        _safeMint(msg.sender, _quantity);
    }
```

- *Aug 31, 2022*: This issue has resolved.

APPENDIX

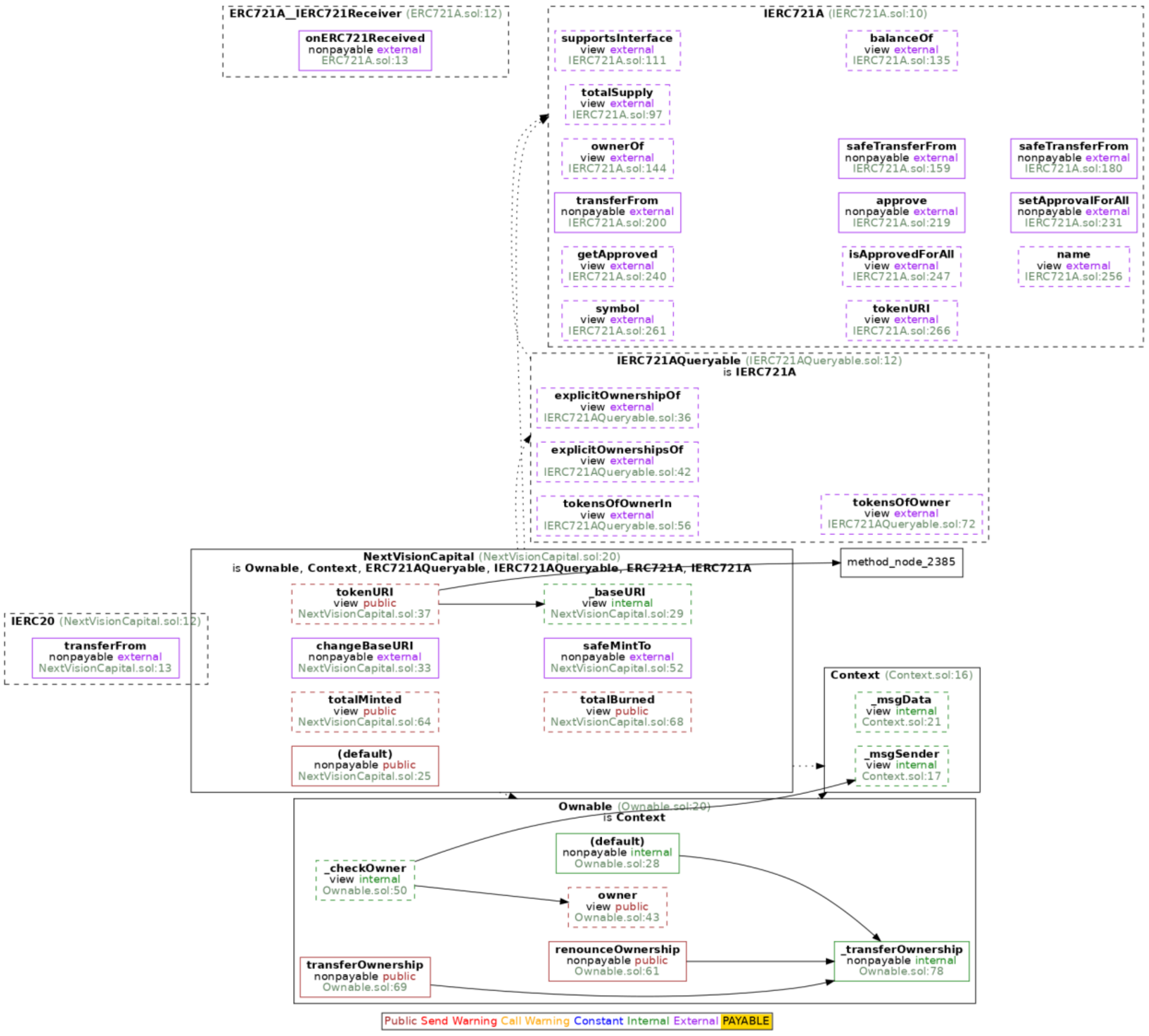
ERC721A__IERC721Receiver (ERC721A.sol:12)
onERC721Received nonpayable external ERC721A.sol:13
IERC721A (IERC721A.sol:10)
supportsInterface view external IERC721A.sol:111
balanceOf view external IERC721A.sol:135
totalSupply view external IERC721A.sol:97
ownerOf view external IERC721A.sol:144
safeTransferFrom nonpayable external IERC721A.sol:159
safeTransferFrom nonpayable external IERC721A.sol:180
transferFrom nonpayable external IERC721A.sol:200
approve nonpayable external IERC721A.sol:219
setApprovalForAll nonpayable external IERC721A.sol:231
getApproved view external IERC721A.sol:240
isApprovedForAll view external IERC721A.sol:247
name view external IERC721A.sol:256
symbol view external IERC721A.sol:261
tokenURI view external IERC721A.sol:266
IERC721AQueryable (IERC721AQueryable.sol:12) is IERC721A
explicitOwnershipOf view external IERC721AQueryable.sol:36
explicitOwnershipsOf view external IERC721AQueryable.sol:42
tokensOfOwnerIn view external IERC721AQueryable.sol:56
tokensOfOwner view external IERC721AQueryable.sol:72
NextVisionCapital (NextVisionCapital.sol:20) is Ownable, Context, ERC721AQueryable, IERC721AQueryable, ERC721A, IERC721A
method_node_2385
IERC20 (NextVisionCapital.sol:12)
transferFrom nonpayable external NextVisionCapital.sol:13
tokenURI view public NextVisionCapital.sol:37
_baseURI view internal NextVisionCapital.sol:29
changeBaseURI nonpayable external NextVisionCapital.sol:33
safeMintTo nonpayable external NextVisionCapital.sol:52
totalMinted view public NextVisionCapital.sol:64
totalBurned view public NextVisionCapital.sol:68
(default) nonpayable public NextVisionCapital.sol:25
Context (Context.sol:16)
_msgData view internal Context.sol:21
_msgSender view internal Context.sol:17
Ownable (Ownable.sol:20) is Context
(default) nonpayable internal Ownable.sol:28
_checkOwner view internal Ownable.sol:50
owner view public Ownable.sol:43
renounceOwnership nonpayable public Ownable.sol:61
transferOwnership nonpayable public Ownable.sol:69
_transferOwnership nonpayable internal Ownable.sol:78
Public Send Warning Call Warning Constant Internal External PAYABLE

Image 1. NextVisionCapital call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 05, 2022*	Public Report	Verichains Lab

Table 3. Report versions history