VALUART

Smart Contract Audit Report

July 13, 2021

IMMUNE BYTES

Introduction

1. About Valuart

Pending - Waiting for client's response.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

IMMUNE BYTES

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Valuart
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: c33a0e033c60f1713c8de0e2881b5e67ee1a24fa
- Testnet Deployment:
    - ValuartMarket: 0x62f50BD129b4f115F27A1063aC31276059Cb0C47
    - ValuartNFT: 0x416092C0E0aCC4b6bA6dcB367f723A6993d8c8B4
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	2	-	1
Closed	-	-	-

Admin/Owner Privileges

The **owner** of **Valuart** smart contracts has various privileges over the smart contracts. These privileges can be misused either intentionally or unintentionally (in case an admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

1. **In the ValuartNFT_v2 contract the *owner* address can change the *_marketplace* address.**
   The **ValuartNFT_v2** contract contains a **setMarketplace**() function by which the owner account can set any ethereum address as the **_marketplace** address. The new **_marketplace** address is given the approval for all the NFTs held by **ValuartNFT_v2** contract.

2. **In the ValuartNFT_v2 contract the *owner* address can mint NFTs to itself.**
   The **ValuartNFT_v2** contract contains a **mint**() function by which the owner account can mint any number of NFTs with different ids to his address.

3. **Owner has the right to approve or decline selling of NFTs.**
   The **owner** of **ValuartMarket_v2** contract has the right to approve or decline the sale of any NFT using the **approveSelling**() and **declineSelling**() functions respectively.

Recommendation:
Consider hardcoding predefined ranges or validations for input variables in privileged access functions. Also consider adding some governance for admin rights for smart contracts or use a multi-sig wallet as admin/owner address.

High Severity Issues

1. **Denial of Service attack is possible.**
   The **bid**() function of **ValuartMarket_v2** contract is intended to be used for bidding amounts by buyers to purchase any auctioned NFT. During this process the **ValuartMarket_v2** contract repays the bid of the last highest bidder if the new bid is better than the last one.

   This bidding process can be misused by an attacker to make the NFT auction unusable by the users. The attacker can make a bid using a malicious smart contract which always reverts all Ether receiving transactions. In short, the malicious contract does not allow anyone to send Ether to itself. Hence whenever any user makes a higher bid for an NFT, this bid transaction will always fail because the last highest bid repayment will always revert.

   More details about the attack can be found here:
   https://consensys.github.io/smart-contract-best-practices/known_attacks/#dos-with-unexpected-revert

   The unexpected revert can also happen in the **endAuction**() function.

   Recommendation:
   Consider following a pull payment mechanism instead of a push payment mechanism. More details can be found here:
   https://consensys.github.io/smart-contract-best-practices/recommendations/#favor-pull-over-push-for-external-calls

2. **The contract sends zero Ether to zero address.**
   In the **buy**() function of **ValuartMarket_v2** contract, the contract intends to transfer a royalty amount to the **tokenCreator** of the NFT.

```
address payable tokenCreator = creators[tokenAddress][tokenId];
        ...
uint royalty = SafeMath.div(tokenForSale.price, 10);

if (tokenCreator == address(0)) {
royalty = 0;
creators[tokenAddress][tokenId] = tokenSeller;
```

```
}
        ...
tokenCreator.transfer(royalty);
```

However in the case where **tokenCreator** is **zero** address (0x000...) the contract transfers 0 ETH to the zero address. The contract updates the **creators** mapping after storing the **creators** value in the local **tokenCreator** variable. This issue does not cause any loss of funds but still the case where the **tokenCreator** is **zero** address should be handled more appropriately.

Recommendation:
Consider checking the **tokenCreator** value appropriately before making any ETH transfer.

Low severity issues

1. **The contract does not follow the check-effect-interaction pattern.**
   In the **bid()** function of **ValuartMarket_v2** contract, the contract repays the last highest bid before updating its internal ledger. The contract performs an ETH transfer before updating the **Auction** struct. It is always recommended to update the contract's internal start before making any external call.

   More details can be found here:
   https://docs.soliditylang.org/en/latest/security-considerations.html#use-the-checks-effects-interactions-pattern

Unit Test

All unit tests provided by the Valuart team are passing.

```
ValuartMarket
  ✓ create auction (94ms)
  ✓ create auction for token that already excists (174ms)
  ✓ end auction (136ms)
  ✓ end not ended auction (94ms)
  ✓ bidding (79ms)
  ✓ selling (62ms)
  ✓ remove from selling if user is not owner (66ms)
  ✓ buying if user is a seller (65ms)

ValuartNFT_V2
  ✓ get symbol
  ✓ get marketplace
  ✓ set baseURI
  ✓ get tokenURI
  ✓ mint

ValuartNFT
  ✓ get symbol


14 passing (6s)
```

Coverage Report

Test coverage of Valuart Contract Master's smart contract is not 100%.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	58.59	34.38	83.33	66.36	
ValuartMarket.sol	53.64	32.76	80	61.54	... 183,186,196
ValuartNFT.sol	88.89	50	87.5	89.47	41,53
All files	58.59	34.38	83.33	66.36	

Recommendation:
We recommend 100% line and branch coverage for unit test cases.

Automated Auditing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contracts.

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to them in real-time. We performed analysis using contract Library on the Rinkeby address of the Valuart Contracts Master used during manual testing:

- ValuartMarket: 0x62f50BD129b4f115F27A1063aC31276059Cb0C47
- ValuartNFT: 0x416092C0E0aCC4b6bA6dcB367f723A6993d8c8B4

It raises no major concern for the contracts.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit section.

```
INFO:Detectors:
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#74) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
ERC721Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#383) sha
dows:
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgrade
able.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
Reentrancy in ValuartNFT_v2.mint(string) (contracts/ValuartNFT.sol#57-65):
        External calls:
        - _safeMint(msg.sender,_tokenId) (contracts/ValuartNFT.sol#60)
                - IERC721ReceiverUpgradeable(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@ope
nzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#350-361)
        State variables written after the call(s):
        - _nextTokenId ++ (contracts/ValuartNFT.sol#62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgrade
able/token/ERC721/ERC721Upgradeable.sol#346-365) ignores return value by IERC721ReceiverUpgradeable(to).onERC721Recei
ved(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable
.sol#350-361)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
ValuartNFT.initialize(string,string)._name (contracts/ValuartNFT.sol#9) shadows:
        - ERC721Upgradeable._name (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.so
l#24) (state variable)
ValuartNFT.initialize(string,string)._symbol (contracts/ValuartNFT.sol#9) shadows:
        - ERC721Upgradeable._symbol (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.
sol#27) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ValuartNFT_v2.setMarketplace(address).marketplace_ (contracts/ValuartNFT.sol#22) lacks a zero-check on :
                - _marketplace = marketplace_ (contracts/ValuartNFT.sol#23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

Concluding Remarks

While conducting the audits of Company Name smart contract, it was observed that the contracts contain High, Medium, and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High, Medium, Low severity issues should be resolved by Company Name developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Company Name platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.