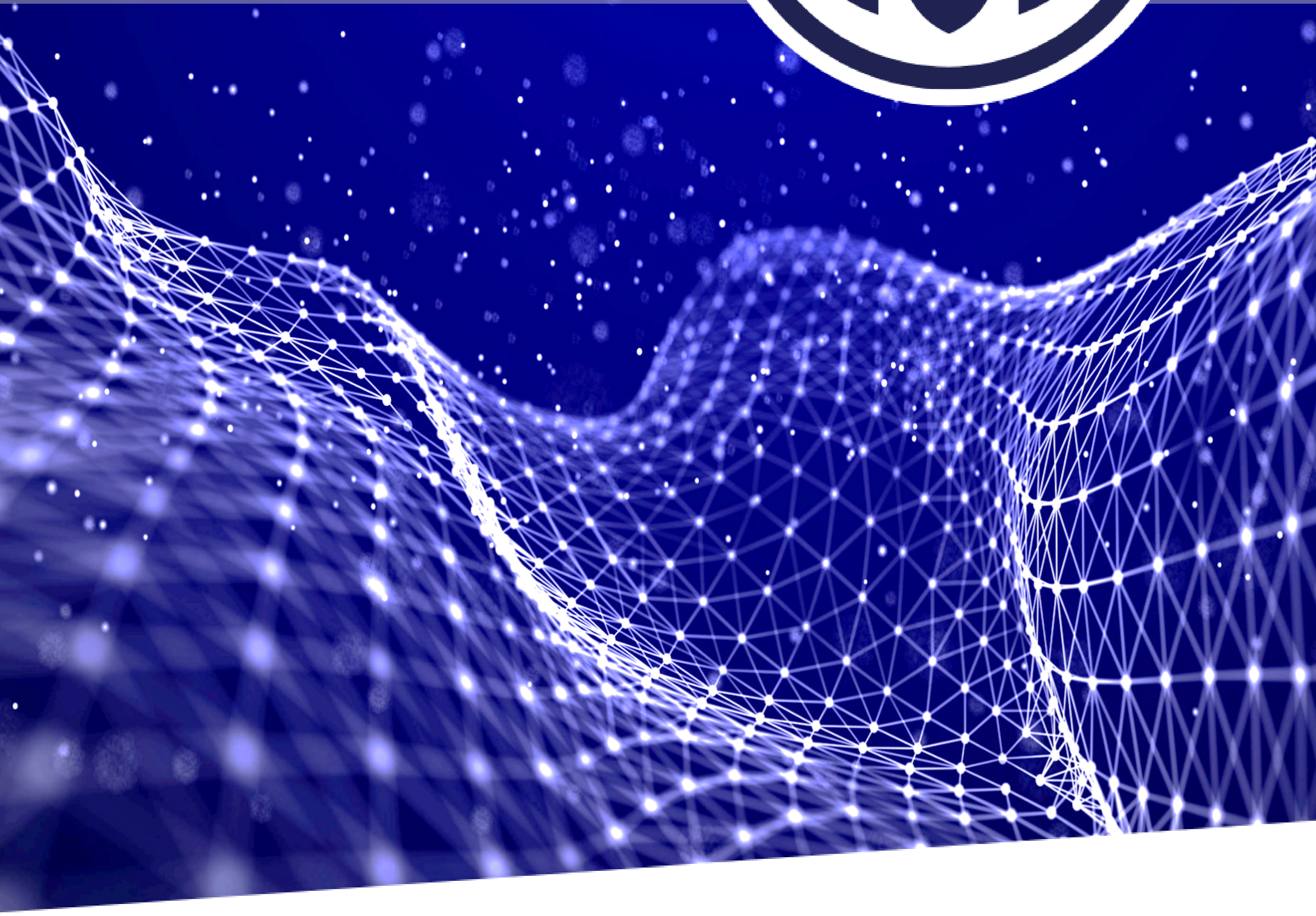
Warden Finance

Wondrous-X

Smart Contract Audit Report



Date Issued: 21 Sep 2022

Version: Final v1.0

Public

Table of Contents

Executive Summary **3**

Overview 3

About Wondrous-X 3

Scope of Work 3

Auditors 5

Disclaimer 5

Audit Result Summary 6

Methodology **7**

Audit Items 8

Risk Rating 10

Findings **11**

Review Findings Summary 11

Detailed Result 12

Appendix **35**

About Us 35

Contact Information 35

References 36

Executive Summary

Overview

Valix conducted a smart contract audit to evaluate potential security issues of the **Wondrous-X feature**. This audit report was published on *21 Sep 2022*. The audit scope is limited to the **Wondrous-X feature.** Our security best practices strongly recommend that the **Warden Finance team** conduct a full security audit for both on-chain and off-chain components of its infrastructure and their interaction. A comprehensive examination has been performed during the audit process utilizing Valix's Formal Verification, Static Analysis, and Manual Review techniques.

About Wondrous-X

Wondrous-X contract is an ERC-721 token deploying on Optimism. The contract is meant to be a reward for the whitelisted wallet addresses to mint their tokens for free during a certain period.

Scope of Work

The security audit conducted does not replace the full security audit of the overall Warden Finance protocol. The scope is limited to the **Wondrous-X feature** and their related smart contracts.

The security audit covered the components at this specific state:

Item	Description
Components	▪ *WondrousX smart contract* ▪ *Imported associated smart contracts and libraries*
Git Repository	▪ *https://github.com/Wardenswap/mwad-eggs-contracts*
Audit Commit	▪ *a304dad2f4174a56526a2e0255003064a4388483 (branch: main)*
Reassessment Commit	▪ *90d1a6db2449d690b99c455825f0189e0aee2dd3 (branch: main)*
Audited Files	▪ *./contracts/WondrousX.sol* ▪ *./contracts/base/SaleSwitch.sol* ▪ *Other imported associated Solidity files*

Excluded Files/Contracts	▪ *./contracts/WonderousXFusion.sol*

Remark: Our security best practices strongly recommend that the Warden Finance team conduct a full security audit for both on-chain and off-chain components of its infrastructure and the interaction between them.

Auditors

Role	Staff List
Auditors	Anak Mirasing Atitawat Pol-in Kritsada Dechawattana Parichaya Thanawuthikrai Phuwanai Thummavet
Authors	Anak Mirasing Atitawat Pol-in Kritsada Dechawattana Parichaya Thanawuthikrai Phuwanai Thummavet
Reviewers	Sumedt Jitpukdebodin

Disclaimer

Our smart contract audit was conducted over a limited period and was performed on the smart contract at a single point in time. As such, the scope was limited to current known risks during the work period. The review does not indicate that the smart contract and blockchain software has no vulnerability exposure.

We reviewed the security of the smart contracts with our best effort, and we do not guarantee a hundred percent coverage of the underlying risk existing in the ecosystem. The audit was scoped only in the provided code repository. The on-chain code is not in the scope of auditing.

This audit report does not provide any warranty or guarantee, nor should it be considered an “approval” or “endorsement” of any particular project. This audit report should also not be used as investment advice nor provide any legal compliance.

Audit Result Summary

From the audit results and the remediation and response from the developer, Valix trusts that the **Wondrous-X feature** has sufficient security protections to be safe for use.

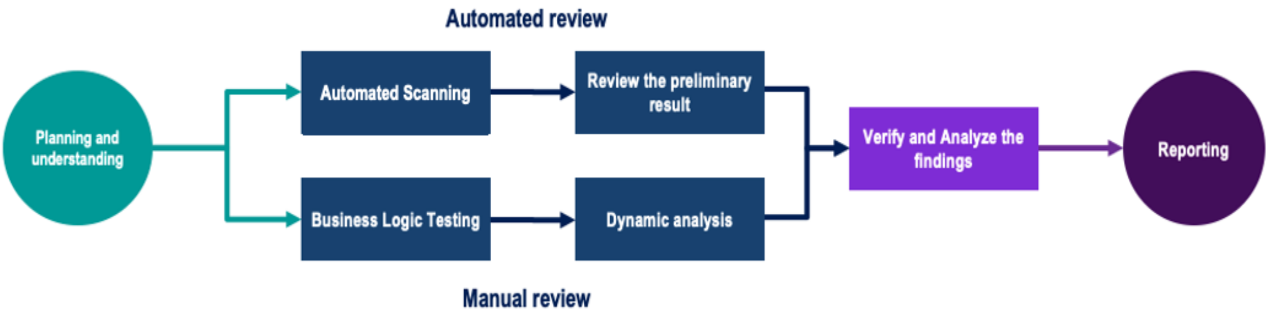
Initially, Valix was able to identify **12 issues** that were categorized from the "Critical" to "Informational" risk level in the given timeframe of the assessment. **For the reassessment, the *Warden* team decided to fix 6 issues and leave 6 issues as acknowledged.** Below is the breakdown of the vulnerabilities found and their associated risk rating for each assessment conducted.

Target	Assessment Result					Reassessment Result				
	C	H	M	L	I	C	H	M	L	I
Wondrous-X	-	2	3	4	3	-	1	1	3	1

Note: *Risk Rating* **C** *Critical,* **H** *High,* **M** *Medium,* **L** *Low,* **I** *Informational*

Methodology

The smart contract security audit methodology is based on Smart Contract Weakness Classification and Test Cases (SWC Registry), CWE, well-known best practices, and smart contract hacking case studies. Manual and automated review approaches can be mixed and matched, including business logic analysis in terms of the malicious doer's perspective. Using automated scanning tools to navigate or find offending software patterns in the codebase along with a purely manual or semi-automated approach, where the analyst primarily relies on one's knowledge, is performed to eliminate the false-positive results.



Planning and Understanding

- Determine the scope of testing and understanding of the application’s purposes and workflows.
- Identify key risk areas, including technical and business risks.
- Determine which sections to review within the resource constraints and review method – automated, manual or mixed.

Automated Review

- Adjust automated source code review tools to inspect the code for known unsafe coding patterns.
- Verify the tool's output to eliminate false-positive results, and adjust and re-run the code review tool if necessary.

Manual Review

- Analyzing the business logic flaws requires thinking in unconventional methods.
- Identify unsafe coding behavior via static code analysis.

Reporting

- Analyze the root cause of the flaws.
- Recommend improvements for secure source code.

Audit Items

We perform the audit according to the following categories and test names.

Category	ID	Test Name
Security Issue	**SEC01**	*Authorization Through tx.origin*
	SEC02	*Business Logic Flaw*
	SEC03	*Delegatecall to Untrusted Callee*
	SEC04	*DoS With Block Gas Limit*
	SEC05	*DoS with Failed Call*
	SEC06	*Function Default Visibility*
	SEC07	*Hash Collisions With Multiple Variable Length Arguments*
	SEC08	*Incorrect Constructor Name*
	SEC09	*Improper Access Control or Authorization*
	SEC10	*Improper Emergency Response Mechanism*
	SEC11	*Insufficient Validation of Address Length*
	SEC12	*Integer Overflow and Underflow*
	SEC13	*Outdated Compiler Version*
	SEC14	*Outdated Library Version*
	SEC15	*Private Data On-Chain*
	SEC16	*Reentrancy*
	SEC17	*Transaction Order Dependence*
	SEC18	*Unchecked Call Return Value*
	SEC19	*Unexpected Token Balance*
	SEC20	*Unprotected Assignment of Ownership*
	SEC21	*Unprotected SELFDESTRUCT Instruction*
	SEC22	*Unprotected Token Withdrawal*
	SEC23	*Unsafe Type Inference*
	SEC24	*Use of Deprecated Solidity Functions*
	SEC25	*Use of Untrusted Code or Libraries*
	SEC26	*Weak Sources of Randomness from Chain Attributes*
	SEC27	*Write to Arbitrary Storage Location*

Category	ID	Test Name
Functional Issue	**FNC01**	*Arithmetic Precision*
	FNC02	*Permanently Locked Fund*
	FNC03	*Redundant Fallback Function*
	FNC04	*Timestamp Dependence*
Operational Issue	**OPT01**	*Code With No Effects*
	OPT02	*Message Call with Hardcoded Gas Amount*
	OPT03	*The Implementation Contract Flow or Value and the Document is Mismatched*
	OPT04	*The Usage of Excessive Byte Array*
	OPT05	*Unenforced Timelock on An Upgradeable Proxy Contract*
Developmental Issue	**DEV01**	*Assert Violation*
	DEV02	*Other Compilation Warnings*
	DEV03	*Presence of Unused Variables*
	DEV04	*Shadowing State Variables*
	DEV05	*State Variable Default Visibility*
	DEV06	*Typographical Error*
	DEV07	*Uninitialized Storage Pointer*
	DEV08	*Violation of Solidity Coding Convention*
	DEV09	*Violation of Token (ERC20) Standard API*

Risk Rating

To prioritize the vulnerabilities, we have adopted the scheme of five distinct levels of risk: **Critical**, **High**, **Medium**, **Low**, and **Informational**, based on OWASP Risk Rating Methodology. The risk level definitions are presented in the table.

Risk Level	Definition
Critical	The code implementation does not match the specification, and it could disrupt the platform.
High	The code implementation does not match the specification, or it could result in losing funds for contract owners or users.
Medium	The code implementation does not match the specification under certain conditions, or it could affect the security standard by losing access control.
Low	The code implementation does not follow best practices or use suboptimal design patterns, which may lead to security vulnerabilities further down the line.
Informational	Findings in this category are informational and may be further improved by following best practices and guidelines.

The **risk value** of each issue was calculated from the product of the **impact** and **likelihood values**, as illustrated in a two-dimensional matrix below.

- **Likelihood** represents how likely a particular vulnerability is exposed and exploited in the wild.
- **Impact** measures the technical loss and business damage of a successful attack.
- **Risk** demonstrates the overall criticality of the risk.

Likelihood Impact	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Informational

The shading of the matrix visualizes the different risk levels. Based on the acceptance criteria, the risk levels "Critical" and "High" are unacceptable. Any issue obtaining the above levels must be resolved to lower the risk to an acceptable level.

Findings

Review Findings Summary

The table below shows the summary of our assessments.

No.	Issue	Risk	Status	Functionality is in use
1	Denial-Of-Service On NFT Minting	High	Acknowledged	In use
2	Unlimited Max Supply For Minting NFTs	High	Fixed	In use
3	Possibly Setting Improper Royalty Percentage	Medium	Fixed	In use
4	Possibly Permanent Ownership Removal	Medium	Fixed	In use
5	Unsafe Ownership Transfer	Medium	Acknowledged	In use
6	Possibly Minting Out-Of-Bound Token ID	Low	Fixed	In use
7	Activating NFT Minting Without Validating Start Time	Low	Acknowledged	In use
8	Configuring Start Time On NFT Minting Is Active	Low	Acknowledged	In use
9	Compiler May Be Susceptible To Publicly Disclosed Bugs	Low	Acknowledged	In use
10	Recommended Improving Transparency And Trustworthiness Of Privileged Operations	Informational	Acknowledged	In use
11	Recommended Removing Unused Library	Informational	Fixed	In use
12	Inconsistent Contract Name	Informational	Fixed	In use

The statuses of the issues are defined as follows:

Fixed: The issue has been completely resolved and has no further complications.

Partially Fixed: The issue has been partially resolved.

Acknowledged: The issue's risk has been reported and acknowledged.

Detailed Result

This section provides all issues that we found in detail.

No. 1	Denial-Of-Service On NFT Minting		
Risk	High	Likelihood	Medium
		Impact	High
Functionality is in use	In use	Status	Acknowledged
Associated Files	*contracts/WonderousX.sol*		
Locations	*WonderousX.sol L: 60 - 72*		

Detailed Issue

The `WondrousX` contract has the `mintAll` function (L60 - 72 in the code snippet below) for minting the `Wondrous-X` tokens. The `mintAll` function allows the whitelisted users to mint their `Wondrous-X` tokens.

We noticed that the minting process of the `mintAll` function allows minting any arbitrary number of the `Wondrous-X` tokens. The minting process can consume gas beyond the block gas limit, leading to a denial-of-service issue.

To elaborate on the issue, the `mintAll` function uses the `for-loop` statement (L69 - 71) to mint tokens. However, this process does not check the length of the `tokenIds` parameter, which can consume gas beyond the block gas limit if the length of the `tokenIds` parameter is too large.

As a result, the transaction would be reverted, and the affected whitelisted users could not mint their tokens.

WonderousX.sol

```
60 function mintAll(
61         uint256[] calldata tokenIds,
62         bytes32[] calldata merkleProofs
63     ) external whenSaleActive nonReentrant {
64     require(
65         verifyWhitelist(_msgSender(), tokenIds, merkleProofs),
66         "Wondrous-X: not in whitelist"
67     );
68 
69     for (uint256 i = 0; i < tokenIds.length; i++) {
70         _mintWDX(_msgSender(), tokenIds[i]);
71     }
72 }
```

Listing 1.1 The *mintAll* function of the *WondrousX* contract

Recommendations

We recommended checking the length of the *tokenIds* parameter on both the *mintAll* (L64) and *verifyWhitelist* (L98) functions.

Since the whitelist would be created off-chain, the whitelist system must not allow adding the tokenIds for each user beyond the minting limit to be compatible with the smart contract.

WonderousX.sol

```
60 function mintAll(
61         uint256[] calldata tokenIds,
62         bytes32[] calldata merkleProofs
63     ) external whenSaleActive nonReentrant {
64     require(tokenIds.length <= MAX_TOKENIDS_LENGTH, "Wondrous-X: The tokenIds' 
   length exceeds the minting limit");
65     require(
66         verifyWhitelist(_msgSender(), tokenIds, merkleProofs),
67         "Wondrous-X: not in whitelist"
68     );
69 
70     for (uint256 i = 0; i < tokenIds.length; i++) {
71         _mintWDX(_msgSender(), tokenIds[i]);
72     }
73 }

   // (...SNIPPED...)

87 function verifyWhitelist(
88     address receiver,
    uint256[] calldata tokenIds,
    bytes32[] calldata merkleProofs
) public view returns (bool) {
    require(tokenIds.length <= MAX_TOKENIDS_LENGTH, "Wondrous-X: The tokenIds' length exceeds the minting limit");
    require(_merkleRoot != "", "Wondrous-X: merkle root not set");

    bytes32 leaf = keccak256(abi.encodePacked(receiver, tokenIds));
    return MerkleProof.verify(merkleProofs, _merkleRoot, leaf);
}
```

Listing 1.2 The improved *mintAll* and *verifyWhitelist* functions

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Warden* team acknowledged this issue and decided to retain the original code.

No. 2	Unlimited Max Supply For Minting NFTs		
Risk	High	Likelihood	Medium
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/WonderousX.sol*		
Locations	*WonderousX.sol L: 77 - 83*		

Detailed Issue

The *WondrousX* is a contract with an NFT minting function for whitelisted users. The internal *_mintWDX* function (L77 - 83 in the code snippet below) is used to mint the token for the eligible users.

We found that the *WondrousX* contract does not have a specific maximum minting supply. This allows a platform admin to open multiple rounds for whitelisted minting without restriction. Consequently, the unlimited supply can decrease the value and rarity of each NFT in the *WondrousX* collection.

WonderousX.sol

```
77  function _mintWDX(address to, uint256 tokenId) internal {
78      require(!_tokenMinted[tokenId], "Wondrous-X: already minted");
79
80      _tokenMinted[tokenId] = true;
81
82      _safeMint(to, tokenId);
83  }
```

Listing 2.1 The *_mintWDX* function that allows an unlimited minting for the *WDX* collection

Recommendations

We recommend specifying the maximum minting supply for the *WDX* collection as shown in L78 in the code snippet below.

WonderousX.sol

```
77 function _mintWDX(address to, uint256 tokenId) internal {
78     require(totalSupply() < MAX_SUPPLY, "Wondrous-X: max supply");
79     require(!_tokenMinted[tokenId], "Wondrous-X: already minted");
80
81     _tokenMinted[tokenId] = true;
82
83     _safeMint(to, tokenId);
84 }
```

Listing 2.2 The improved *_mintWDX* function
that restricts a minting amount with the MAX_SUPPLY

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Warden* team fixed this issue by verifying the inputted *tokenId* with the maximum token id, *MAX_TOKEN_ID* as shown in L81 in the code snippet below. As the *MAX_TOKEN_ID* is a constant value of 2999, the maximum minting supply for the *WDX* collection is 3000.

WondrousX.sol

```
79 function _mintWDX(address to, uint256 tokenId) internal {
80     require(!_tokenMinted[tokenId], "Wondrous-X: already minted");
81     require(tokenId <= MAX_TOKEN_ID, "Wondrous-X: invalid tokenId");
82
83     _tokenMinted[tokenId] = true;
84
85     _safeMint(to, tokenId);
86 }
```

Listing 2.3 The fixed *_mintWDX* function

No. 3	Possibly Setting Improper Royalty Percentage		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/WonderousX.sol*		
Locations	*WonderousX.sol L: 134 - 142*		

Detailed Issue

The `WondrousX` contract has the `setRoyalty` function, which can set the new `_royaltyPercentage` and `_royaltyReceiver` (L138 - 139) by providing the new royalty percentage and the new receiver address.

However, we noticed that the improper configuration of the `_royaltyPercentage` (L138) can lead to an incorrect calculation result of the `_royaltyAmount` variable (L151) in the `royaltyInfo` function.

The following formula is used to calculate the `_royaltyAmount` variable:

```
_royaltyAmount = (salePrice * _royaltyPercentage) / 10000
```

Let’s say we have:

```
salePrice = 10000 and _royaltyPercentage = 20000
```

Thus:

```
_royaltyAmount = (10000 * 20000) / 10000

_royaltyAmount = 20000
```

The example above shows that if the `_royaltyPercentage` is greater than 10000, the resulting `_royaltyAmount` would be greater than the `salePrice` `which is impractical.`

WonderousX.sol

```
133 /// @dev Set new royalty percentage and receiver address. Emit {SetRoyalty}.
134 function setRoyalty(uint256 royaltyPercentageBps, address receiver)
135     public
136     onlyOwner
137 {
138     _royaltyPercentage = royaltyPercentageBps;
139     _royaltyReceiver = receiver;
140
141     emit SetRoyalty(royaltyPercentageBps, receiver);
142 }
143
144 /// @dev Return royalty amount and receiver. See EIP-2981.
145 function royaltyInfo(uint256 tokenId, uint256 salePrice)
146     external
147     view
148     override
149     returns (address receiver, uint256 royaltyAmount)
150 {
151     uint256 _royaltyAmount = (salePrice * _royaltyPercentage) / 10000;
152     return (_royaltyReceiver, _royaltyAmount);
153 }
```

Listing 3.1 The *setRoyalty* and *royaltyInfo* functions of the *WondrousX* contract

Recommendations

We recommend validating the *royaltyPercentageBps* parameter to ensure that its value is lower than or equal to a proper value. The code snippet below (L138) shows the code example for the remediation.

WonderousX.sol

```
133 /// @dev Set new royalty percentage and receiver address. Emit {SetRoyalty}.
134 function setRoyalty(uint256 royaltyPercentageBps, address receiver)
135     public
136     onlyOwner
137 {
138     require(royaltyPercentageBps <= MAX_ROYALTY_PERCENTAGE, "Wondrous-X:
    royaltyPercentageBps is more than MAX_ROYALTY_PERCENTAGE");
139     _royaltyPercentage = royaltyPercentageBps;
140     _royaltyReceiver = receiver;
141
142     emit SetRoyalty(royaltyPercentageBps, receiver);
143 }
```

Listing 3.2 The improved *setRoyalty* function

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

This issue was fixed according to our suggestion as the `MAX_ROYALTY_PERCENTAGE` is a constant value of 2000 (i.e., 20% basis points).

No. 4	Possibly Permanent Ownership Removal		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	*@openzeppelin/contracts/access/Ownable.sol*		
Locations	*Ownable.sol L: 54 - 56*		

Detailed Issue

The *WondrousX* contract inherits from the *Ownable* abstract contract. The *Ownable* contract implements the *renounceOwnership* function (L54 - 56 in the code snippet below), which can remove the contract's ownership permanently.

If the contract owner mistakenly invokes the *renounceOwnership* function, they will immediately lose ownership of the contract, and this action cannot be undone.

Ownable.sol

```
54  function renounceOwnership() public virtual onlyOwner {
55      _transferOwnership(address(0));
56  }

    // (...SNIPPED...)

71  function _transferOwnership(address newOwner) internal virtual {
72      address oldOwner = _owner;
73      _owner = newOwner;
74      emit OwnershipTransferred(oldOwner, newOwner);
75  }
```

Listing 4.1 The *renounceOwnership* function
that can remove the ownership of the contract permanently

Recommendations

We consider the *renounceOwnership* function risky, and the contract owner should use this function with extra care.

If possible, we recommend removing or disabling this function from the contract. The code snippet below shows an example solution to disabling the associated *renounceOwnership* function.

WonderousX.sol

```solidity
function renounceOwnership() external override onlyOwner {
    revert("Ownable: renounceOwnership function is disabled");
}
```

Listing 4.2 The disabled *renounceOwnership* function

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Warden* team fixed this issue by disabling the *renounceOwnership* function as per our suggestion.

No. 5	Unsafe Ownership Transfer		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Acknowledged
Associated Files	*@openzeppelin/contracts/access/Ownable.sol*		
Locations	*Ownable.sol L: 62 - 65*		

Detailed Issue

The *WondrousX* contract inherits from the *Ownable* abstract contract. The *Ownable* contract implements the *transferOwnership* function (L62 - 65 in the code snippet below), which can transfer the ownership of the contract from the current owner to another owner.

Ownable.sol

```
62  function transferOwnership(address newOwner) public virtual onlyOwner {
63      require(newOwner != address(0), "Ownable: new owner is the zero address");
64      _transferOwnership(newOwner);
65  }

    // (...SNIPPED...)

71  function _transferOwnership(address newOwner) internal virtual {
72      address oldOwner = _owner;
73      _owner = newOwner;
74      emit OwnershipTransferred(oldOwner, newOwner);
75  }
```

Listing 5.1 The *transferOwnership* function that has the unsafe ownership transfer

From the code snippet above, the address variable *newOwner* (L62) may be incorrectly specified by the current owner by mistake; for example, an address that a new owner does not own was inputted. Consequently, the new owner loses ownership of the contract immediately, and this action is unrecoverable.

Recommendations

We recommend applying the two-step ownership transfer mechanism as shown in the code snippet below.

WonderousX.sol

```
192 function transferOwnership(address _candidateOwner) public override onlyOwner {
193     require(_candidateOwner != address(0), "Ownable: candidate owner is the zero address");
194     candidateOwner = _candidateOwner;
195     emit NewCandidateOwner(_candidateOwner);
196 }
197
198 function claimOwnership() external {
199     require(candidateOwner == _msgSender(), "Ownable: caller is not the candidate owner");
200     _transferOwnership(candidateOwner);
201     candidateOwner = address(0);
202 }
```

Listing 5.2 The recommended two-step ownership transfer mechanism

This mechanism works as follows.

1. The current owner invokes the *transferOwnership* function by specifying the candidate owner address *_candidateOwner* (L192).
2. The candidate owner proves access to his account and claims the ownership transfer by invoking the *claimOwnership* function (L198).

The recommended mechanism ensures that the ownership of the contract would be transferred to another owner who can access his account only.

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Warden* team acknowledged this issue and decided to retain the original code.

No. 6	Possibly Minting Out-Of-Bound Token ID		
Risk	Low	Likelihood	Medium
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/WonderousX.sol*		
Locations	*WonderousX.sol L: 77 - 83*		

Detailed Issue

The *WondrousX* contract allows the whitelisted users to mint all their eligible tokens via the *mintAll* function. The *mintAll* function contains the logic to verify the caller and then mint the tokens through the *verifyWhitelist* and *_mintWDX* functions, respectively.

The *_mintWDX* function is the internal function to mint a *Wondrous-X* token to the specified address, *to*. However, we found that there are no bounds checking for the *tokenId* parameter before minting which could allow a user to mint an NFT token with an out-of-bound *tokenId* (if the off-chain whitelist system functions incorrectly).

WonderousX.sol

```
77  function _mintWDX(address to, uint256 tokenId) internal {
78      require(!_tokenMinted[tokenId], "Wondrous-X: already minted");
79
80      _tokenMinted[tokenId] = true;
81
82      _safeMint(to, tokenId);
83  }
```

Listing 6.1 The *_mintWDX* function that lacks of bounds checking for the *tokenId* parameter

Recommendations

We recommend validating that the given *tokenId* does not exceed the max supply (L79 in the below code snippet).

WonderousX.sol

```
77 function _mintWDX(address to, uint256 tokenId) internal {
78     require(!_tokenMinted[tokenId], "Wondrous-X: already minted");
79     require(tokenId < MAX_SUPPLY, "Wondrous-X: invalid tokenId");
80     _tokenMinted[tokenId] = true;
81 
82     _safeMint(to, tokenId);
83 }
```

Listing 6.2 The improved *_mintWDX* function

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Warden* team fixed this issue by verifying the inputted *tokenId* with the maximum token id, *MAX_TOKEN_ID* as shown in L81 in the code snippet below, as the *MAX_TOKEN_ID* is a constant value of 2999.

WondrousX.sol

```
79 function _mintWDX(address to, uint256 tokenId) internal {
80     require(!_tokenMinted[tokenId], "Wondrous-X: already minted");
81     require(tokenId <= MAX_TOKEN_ID, "Wondrous-X: invalid tokenId");
82 
83     _tokenMinted[tokenId] = true;
84 
85     _safeMint(to, tokenId);
86 }
```

Listing 6.3 The fixed *_mintWDX* function

No. 7	Activating NFT Minting Without Validating Start Time		
Risk	Low	Likelihood	Medium
		Impact	Low
Functionality is in use	In use	Status	Acknowledged
Associated Files	*contracts/base/SaleSwitch.sol*		
Locations	*SaleSwitch.sol L: 21 - 25*		

Detailed Issue

We found that the *startSale* function (code snippet below) can be triggered by an owner to activate the NFT minting process without validating whether or not the state variable *saleStartTime* is set.

Considering the case that the *saleStartTime* variable was not set. Upon invoking the *startSale* function, the state variable *saleActive* would be activated (L23). Nonetheless, the NFT minting process still could not actually be active for the minting.

SaleSwitch.sol

```
21  function startSale() external onlyOwner {
22      require(!saleActive, "SaleSwitch: already active");
23      saleActive = true;
24      emit SaleStarted(block.timestamp);
25  }
```

Listing 7.1 The *startSale* function
that does not validate the *saleStartTime* state variable

Recommendations

We recommend validating the state variable `saleStartTime` (L23 in the code snippet below) or some proper state variables before activating the state variable `saleActive`.

SaleSwitch.sol

```
21 function startSale() external onlyOwner {
22     require(!saleActive, "SaleSwitch: already active");
23     require(saleStartTime > 0, "SaleSwitch: saleStartTime not set");
24     saleActive = true;
25     emit SaleStarted(block.timestamp);
26 }
```

Listing 7.2 The improved `startSale` function

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The `Warden` team acknowledged this issue and decided to retain the original code.

No. 8	Configuring Start Time On NFT Minting Is Active		
Risk	Low	Likelihood	Medium
		Impact	Low
Functionality is in use	In use	Status	Acknowledged
Associated Files	*contracts/base/SaleSwitch.sol*		
Locations	*SaleSwitch.sol L: 33 - 36*		

Detailed Issue

We found that the `setSaleStartAt` function can be executed by an owner to set a permitted timestamp `saleStartTime` (L34 in the code snippet below) for the NFT minting process. Nevertheless, the function does not validate whether the state variable `saleActive` is activated or not.

In fact, the `setSaleStartAt` function should never be executable if the `saleActive` state variable is activated (i.e., `saleActive` is set to `true`). In other words, the owner should call the `pauseSale` function to deactivate the `saleActive` variable first. Then, call the `setSaleStartAt` function to configure the new timestamp.

SaleSwitch.sol

```
33  function setSaleStartAt(uint256 _saleStartTime) public onlyOwner {
34      saleStartTime = _saleStartTime;
35      emit SaleStartSet(_saleStartTime);
36  }
```

Listing 8.1 The `setSaleStartAt` function that does not validate the `saleActive` state variable

Recommendations

We recommend validating the state variable *saleActive* (L34 in the code snippet below) before setting the state variable *saleStartTime*.

SaleSwitch.sol

```
33 function setSaleStartAt(uint256 _saleStartTime) public onlyOwner {
34     require(!saleActive, "SaleSwitch: sale is active");
35     saleStartTime = _saleStartTime;
36     emit SaleStartSet(_saleStartTime);
37 }
```

Listing 8.2 The improved *setSaleStartAt* function

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Warden* team acknowledged this issue and decided to retain the original code.

No. 9	Compiler May Be Susceptible To Publicly Disclosed Bugs		
Risk	Low	Likelihood	Low
		Impact	Medium
Functionality is in use	In use	Status	Acknowledged
Associated Files	*contracts/WonderousX.sol* *contracts/base/SaleSwitch.sol*		
Locations	*WonderousX.sol L: 2* *SaleSwitch.sol L: 2*		

Detailed Issue

The *WondrousX* and *SaleSwitch* contracts use an outdated Solidity compiler version which may be susceptible to publicly disclosed vulnerabilities. The current compiler version is 0.8.7, which contains the list of known bugs at the following link:

https://docs.soliditylang.org/en/v0.8.16/bugs.html

The known bugs may not directly lead to the vulnerability, but it may increase an opportunity to trigger some attacks further.

An example code that does not use the latest patch version is shown below.

WonderousX.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.7;
```

Listing 9.1 The current compiler version of the *WondrousX* contract

Recommendations

We recommend using the latest patch version, v0.8.16, which fixes all known bugs.

Reassessment

The *Warden* team acknowledged this issue and decided to retain the original code.

No. 10	Recommended Improving Transparency And Trustworthiness Of Privileged Operations		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Acknowledged
Associated Files	*contracts/WonderousX.sol* *contracts/base/SaleSwitch.sol* *@openzeppelin/contracts/access/Ownable.sol*		
Locations	*WonderousX.sol L: 116 - 120, 123 - 127, and 134 - 142* *SaleSwitch.sol L: 21 - 25, 27 - 31, and 33 - 36* *Ownable.sol L: 54 - 56 and 62 - 65*		

Detailed Issue

Our analysis found that the owner account can perform several privileged operations as follows.

1. ***setBaseURI*** `function (L116 - 120 in` WonderousX.sol)
2. ***setMerkleRoot*** function (L123 - 127 in WonderousX.sol)
3. ***setRoyalty*** function (L134 - 142 in WonderousX.sol)
4. ***startSale*** function (L21 - 25 in SaleSwitch.sol)
5. ***pauseSale*** function (L27 - 31 in SaleSwitch.sol)
6. ***setSaleStartAt*** function (L33 - 36 in SaleSwitch.sol)
7. ***renounceOwnership*** function (L54 - 56 in Ownable.sol)
8. ***transferOwnership*** function (L62 - 65 in Ownable.sol)

Although those privileged functions do not manage significant user assets that could lead to loss of user assets directly. However, we consider that those privileged functions should be improved for transparency and trustworthiness.

Recommendations

We recommend governing the associated functions with the ***Multisig***, ***Timelock,*** and/or ***DAO (Decentralized Autonomous Organization)*** mechanisms to improve the transparency and trustworthiness of the *WDX* collection.

Reassessment

The *Warden* team acknowledged this issue and decided to retain the original code and design.

No. 11	Recommended Removing Unused Library		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/WonderousX.sol*		
Locations	*WonderousX.sol L: 9*		

Detailed Issue

We found that the *WondrousX* contract imported the unused library *Counters* (L9 in the code snippet below). Hence, the library can be removed to improve code readability.

WonderousX.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.7;
3
4  import "@openzeppelin/contracts/interfaces/IERC2981.sol";
5  import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
6  import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
7  import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol";
8  import "@openzeppelin/contracts/access/Ownable.sol";
9  import "@openzeppelin/contracts/utils/Counters.sol";

   // (...SNIPPED...)
```

Listing 11.1 The unused *Counters* library

Recommendations

We recommend removing the unused imported library *Counters* to improve code readability.

Reassessment

The *Warden* team removed the unused library in accordance with our recommendation.

No. 12	Inconsistent Contract Name		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/WonderousX.sol*		
Locations	*WonderousX.sol L: 14*		

Detailed Issue

We found inconsistency between the ***file name*** (***WonderousX***) and the ***contract name*** (***WondrousX***) as presented in the below code snippet, which can confuse the users and developers.

WonderousX.sol

```
14  contract WondrousX is
15     ERC721,
16     ERC721Enumerable,
17     ERC721Burnable,
18     IERC2981,
19     Ownable,
20     SaleSwitch,
21     ReentrancyGuard
22  {
    // (...SNIPPED...)
```

Listing 12.1 The contract name *WondrousX*

Recommendations

We recommend renaming the associated contract and file names to be consistent.

Reassessment

The *Warden* team fixed this issue by renaming the file name from ***WonderousX.sol*** to ***WondrousX.sol*** to be consistent with the contract name.

Appendix

About Us

Founded in 2020, Valix Consulting is a blockchain and smart contract security firm offering a wide range of cybersecurity consulting services such as blockchain and smart contract security consulting, smart contract security review, and smart contract security audit.

Our team members are passionate cybersecurity professionals and researchers in areas of private and public blockchain technology, smart contract, and decentralized application (DApp).

We provide a service for assessing and certifying the security of smart contracts. Our service also includes recommendations on smart contracts' security and gas optimization to bring the most benefit to users and platform creators.

Contact Information

✉ **info@valix.io**

https://www.facebook.com/ValixConsulting

https://twitter.com/ValixConsulting

https://medium.com/valixconsulting

References

Title	Link
OWASP Risk Rating Methodology	https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
Smart Contract Weakness Classification and Test Cases	https://swcregistry.io/

ValiX