

Xtatuz DMCC

XTATUZ asset tokenization

Smart Contract Audit Report



Date Issued: 17 Mar 2023

Version: Final v1.0

Public

Valix
Consulting

Table of Contents

Executive Summary	3
Overview	3
About XTATUZ asset tokenization	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	14
Appendix	173
About Us	173
Contact Information	173
References	174

Executive Summary

Overview

Valix conducted a smart contract audit to evaluate potential security issues of the **XTATUZ asset tokenization feature**. This audit report was published on *17 Mar 2023*. The audit scope is limited to the **XTATUZ asset tokenization feature**. Our security best practices strongly recommend that the **Xtatuz DMCC 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 XTATUZ asset tokenization

The **XTATUZ asset tokenization** feature is a tool for the tokenization of assets with the objective to unlock the possibility of co-ownership or the fractionalization of ownership of traditional assets that already exist in this current world. Each individual token, which represents an ownership or a partial ownership of a particular asset, should possess the quality of transferability and can be traced and tracked on a decentralized ledger. This allows ownership to be traded with ease in a decentralized manner, enhancing transparency, security and accessibility.

Scope of Work

The security audit conducted does not replace the full security audit of the overall Xtatuz DMCC protocol. The scope is limited to the **XTATUZ asset tokenization feature** and their related smart contracts.

The security audit covered the components at this specific state:

Item	Description
Components	<ul style="list-style-type: none"> <i>XTATUZ asset tokenization smart contract</i> <i>Imported associated smart contracts and libraries</i>
Git Repository	<ul style="list-style-type: none"> <i>https://github.com/XTATUZ/xtatuz-contracts-v1</i>
Audit Commit	<ul style="list-style-type: none"> <i>ffb2d2388cb5d9446aa3cb4105b4156dd1362677 (branch: main)</i>
Reassessment Commit	<ul style="list-style-type: none"> <i>ebf430277b06e355a2199392e926d923235355f6 (branch: main)</i>

Audited Files	<ul style="list-style-type: none"> ▪ <code>./contracts/Presaled.sol</code> ▪ <code>./contracts/PresaledFactory.sol</code> ▪ <code>./contracts/ProjectFactory.sol</code> ▪ <code>./contracts/Property.sol</code> ▪ <code>./contracts/PropertyFactory.sol</code> ▪ <code>./contracts/XtatuzFactory.sol</code> ▪ <code>./contracts/XtatuzProject.sol</code> ▪ <code>./contracts/XtatuzReferral.sol</code> ▪ <code>./contracts/XtatuzReroll.sol</code> ▪ <code>./contracts/XtatuzRouter.sol</code> ▪ <i>Other imported associated Solidity files</i>
Excluded Files/Contracts	<ul style="list-style-type: none"> ▪ <code>./contract/Token.sol</code> ▪ <code>./contract/XTAToken.sol</code>

Remark: Our security best practices strongly recommend that the Xtatuz DMCC 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 (Lead Auditor) Kritsada Dechawattana Parichaya Thanawuthikrai
Authors	Anak Mirasing Kritsada Dechawattana Parichaya Thanawuthikrai
Reviewers	Phuwanai Thummavet (Technical Advisor) 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 **XTATUZ asset tokenization feature** has sufficient security protections to be safe for use.



Initially, Valix was able to identify **50 issues** that were categorized from the “Critical” to “Informational” risk level in the given timeframe of the assessment.

For the reassessment, the *Xtatuz* team acknowledged a total of 50 issues, including 8 critical issues, 11 high issues, 17 medium issues, 4 low issues, and 10 informational issues. Of these, the team was able to completely fix 34 issues, partially fix 1 issue, and acknowledge 15 issues.

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		
XTATUZ asset tokenization	8	11	17	4	10	1	5	7	1	2		

Note: Risk Rating

C

Critical,

H

High,

M

Medium,

L

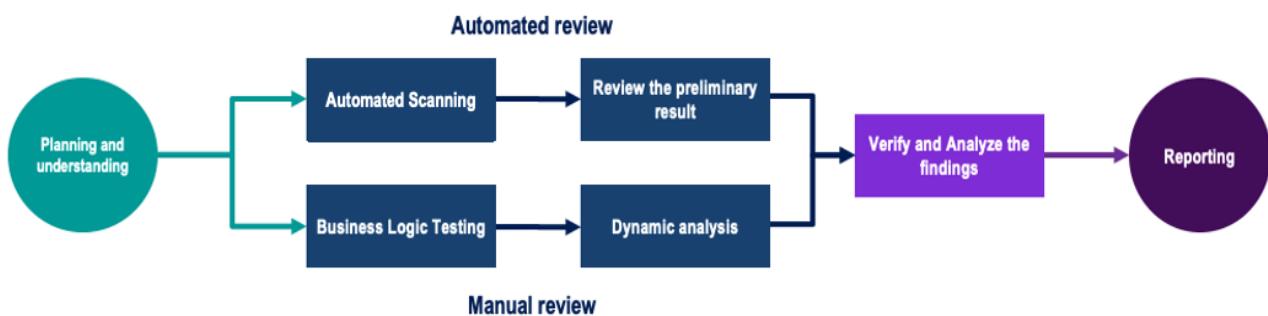
Low,

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

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

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.

Impact \ Likelihood	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	Inconsistent State After Presaled NFT Owners Transfer Tokens Directly	Critical	Fixed	In use
2	Unable To Pull Back Property NFTs	Critical	Fixed	In use
3	Lack Of Validating Master Token ID Leads To Unable To Minting Property NFTs	Critical	Fixed	In use
4	Lack Of Validating The Project Status	Critical	Fixed	In use
5	Function Calls Ordering Issues On Project Finalization Process	Critical	Fixed	In use
6	Out-Of-Sync Operator Transfer	Critical	Fixed	In use
7	Improper Project's Statuses	Critical	Fixed	In use
8	Potential Denial-Of-Service On Adding Project Member	Critical	Acknowledged	In use
9	Potential Denial-Of-Service On Retrieving Member NFTs	High	Acknowledged	In use
10	Potential Denial-Of-Service On Defragmentation	High	Acknowledged	In use
11	Potential Denial-Of-Service On Retrieving Token List	High	Acknowledged	In use
12	Use Of Incorrect Approval Function	High	Fixed	In use
13	Design Flaw In Directly Minting Presale NFTs	High	Fixed	In use
14	Design Flaw In Directly Burning Presale NFTs	High	Fixed	In use
15	Design Flaw In Directly Minting Property NFTs	High	Fixed	In use
16	State Inconsistency Upon Directly Claiming Property NFT Or Directly Refunding Presale NFT	High	Fixed	In use
17	Possibly Insufficient Referral Fee	High	Fixed	In use
18	Lack Of Updating State Variables Upon Claiming Property NFT Or Refunding Presale NFT	High	Acknowledged	In use
19	Possibly Unable To Finish Project	High	Acknowledged	In use

20	Arbitrarily Setting Presale Period	Medium	Fixed	In use
21	Possibly Setting Inconsistent Token URI	Medium	Acknowledged	In use
22	Lack Of Time Delay In Pullback Process	Medium	Fixed	In use
23	Potential Denial-Of-Service On Getting Member Projects	Medium	Acknowledged	In use
24	Potential Denial-Of-Service On Getting All Collections	Medium	Fixed	In use
25	Permanent Lock Of Ether Funds In Contracts	Medium	Fixed	In use
26	Double Registering Project Member	Medium	Fixed	In use
27	Usage Of Unsafe Token Transfer Functions	Medium	Fixed	In use
28	Double Spending On The finishProject Function	Medium	Fixed	In use
29	Unchecking Bounds Of Presale Token ID	Medium	Acknowledged	In use
30	Lack Of Resetting States On Operator And Trustee Transfers	Medium	Acknowledged	In use
31	Unsafe Privileged Role Transfer	Medium	Acknowledged	In use
32	Lack Of Proper Validation On The underwriteCount Parameter	Medium	Acknowledged	In use
33	Lack Of Upper Bound On Referral Percentage	Medium	Fixed	In use
34	Incorrectly Updating Referral Amount	Medium	Fixed	In use
35	Recommended Improving Transparency And Trustworthiness	Medium	Acknowledged	In use
36	Permanent Lock Of Referral Fee	Medium	Fixed	In use
37	Incorrect Time Verification On Extending Presale Period	Low	Fixed	In use
38	Potential Denial-Of-Service On Setting Reroll Data	Low	Acknowledged	In use
39	Compiler Is Not Locked To Specific Version	Low	Fixed	In use
40	Lack Of Validating Reroll Data	Low	Fixed	In use
41	Missing Validation Of Zero Address	Informational	Fixed	In use
42	Predictable Randomness	Informational	Acknowledged	In use
43	Recommended Improving Event Emissions	Informational	Fixed	In use
44	Recommended Improving Transparency And Traceability Of Crucial Variables	Informational	Partially Fixed	In use

45	Recommended Enforcing Checks-Effects-Interactions Pattern	Informational	Fixed	In use
46	Recommended Event Emissions For Transparency And Traceability	Informational	Fixed	In use
47	Recommended Removing Unused Code	Informational	Fixed	In use
48	Inconsistent Error Message With The Code	Informational	Fixed	In use
49	Misspelling State Variable	Informational	Fixed	In use
50	Inconsistent Interfaces With The Implementation	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	Inconsistent State After Presaled NFT Owners Transfer Tokens Directly		
Risk	Critical	Likelihood	High
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	<i>contract/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 165 - 174</i>		

Detailed Issue

The `addProjectMember` function enables users to purchase presale tokens. Upon invocation, the function first checks the availability of tokens and subsequently mints the purchased tokens. The `getMemberedNFTList` state is then updated to reflect the changes. (L38, L385 in code snippet 1.2).

However, if a member transfers their token directly to another wallet, the `getMemberedNFTList` state will not be updated, which can cause inconsistencies in the `refund` function. The size of the `getMemberedNFTList` state will not decrease after the transfer, resulting in miscalculations in the payout fund for the refund (L172 in code snippet 1.3).

XtatuzProject.sol

```

38  mapping(address => uint256[]) public getMemberedNFTList;

    // (...SNIPPED...)
120 function addProjectMember(address member_, uint256[] memory nftList_)
121     public
122     isAvailable
123     isLeftReserve
124     whenNotPaused
125     onlyOwner
126     returns (uint256)
127 {
128     uint256 amount = nftList_.length;
129     require(amount > 0, "PROJECT: ZERO_AMOUNT");
130     for(uint i = 0; i < amount; ++i){
131         require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");

```

```

132     }
133     _checkAvailableNFT(nftList_);
134
135     uint256 price = minPrice * amount;
136     projectValue += price;
137
138     _pickupAvailableNFT(nftList_, member_);
139
140     countReserve -= amount;
141
142     projectMember.push(member_);
143
144     IPresaled(_presaledAddress).mint(member_, nftList_);
145     emit AddProjectMember(projectId, member_, price);
146
147     return price;
148 }
```

Listing 1.1 The *addProjectMember* function**XtatuzProject.sol**

```

382 function _pickupAvailableNFT(uint256[] memory nftList_, address member_)
internal {
383     for (uint256 i = 0; i < nftList_.length; i++) {
384         unavailableNFT.push(nftList_[i]);
385         getMemberedNFTList[member_].push(nftList_[i]);
386     }
387 }
```

Listing 1.2 The *_pickupAvailableNFT* functions**XtatuzProject.sol**

```

167 function refund(address member_) public onlyOperator whenNotPaused{
168     uint256[] memory tokenList =
169     IPresaled(_presaledAddress).getPresaledOwner(member_);
170     require(projectStatus() == IXtatuzProject.Status.REFUND, "PROJECT:
171     PROJECT_UNREFUNDED");
172
173     IPresaled(_presaledAddress).burn(tokenList);
174
175     uint256 totalToken = getMemberedNFTList[member_].length * minPrice;
176     IERC20(tokenAddress).transfer(member_, totalToken);
177 }
```

Listing 1.3 The *refund* function

Recommendations

We recommend implementing a hook function that disables the transfer feature for presale tokens. This would help ensure that presale tokens cannot be transferred, thereby preventing potential issues with state consistency.

Presaled.sol

```
100  function _beforeTokenTransfer(address from, address to, uint256 tokenId)
101    internal
102    override(ERC721Enumerable) {
103      super._beforeTokenTransfer(from, to, tokenId);
104      require(from == address(0) || to == address(0), "PRESALED:
105        UNABLE_TO_TRANSFER");
```

Listing 1.4 Improved *Presaled* contract

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 2	Unable To Pull Back Property NFTs		
Risk	Critical	Likelihood	High
Functionality is in use	In use	Impact	High
Associated Files	<i>contract/Property.sol</i> <i>contract/XtatuzRouter.sol</i>		
Locations	<i>Property.sol L: 65 - 76</i> <i>XtatuzRouter.sol L: 288 - 299</i>		

Detailed Issue

The *pullbackInactive* function enables SPV accounts to retrieve property tokens from inactive wallets (L295 in code snippet 2.2). This is made possible because the *mintFragment* function grants approval to the SPV account during the minting process (L74 in code snippet 2.1).

However, if a member transfers their tokens directly to another wallet, the approval for the recipient wallet will not be automatically granted to the SPV account (L144 in code snippet 2.3), which can cause issues when the SPV account attempts to invoke the *pullbackInactive* function.

As a result, the transaction will revert, and the SPV account will be unable to pull back tokens from the inactive wallet (L181 in code snippet 2.4).

Property.sol	
65	function mintFragment(address to, uint256[] memory tokenIdList) public onlyOperator { require(ownerOf(0) == address(this), "PROPERTY: NO_MASTER_NFT"); require(isMintedMaster == true, "PROPERTY: MASTER_NOT_MINTED"); uint256 amount = tokenIdList.length; for (uint256 index = 0; index < amount; index++) { uint256 tokenId = tokenIdList[index]; require(!_exists(tokenId) == false, "PROJECT: ALREADY_EXISTS"); _safeMint(to, tokenId); } _setApprovalForAll(to, _routerAddress, true); emit MintFragment(to, tokenIdList); }

Listing 2.1 The *mintFragment* function

XtatuzRouter.sol

```

288 function pullbackInactive(uint256 projectId_, address inactiveWallet_) public
289   onlySpv {
290     require(_isNotice[projectId_][inactiveWallet_] == true, "ROUTER:
291 NOTICE_BEFORE");
292     address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
293     IProperty property = IProperty(propertyAddress);
294     uint256[] memory nftList = property.getTokenIdList(inactiveWallet_);
295
296     for(uint i = 0; i < nftList.length; i++){
297       IERC721(propertyAddress).safeTransferFrom(inactiveWallet_, msg.sender,
298         nftList[i]);
299     }
299 }
```

Listing 2.2 The *pullbackInactive* function**ERC721.sol**

```

143 function isApprovedForAll(address owner, address operator) public view
144   virtual override returns (bool) {
145   return _operatorApprovals[owner][operator];
145 }
```

Listing 2.3 The *isApprovedForAll* function**ERC721.sol**

```

175 function safeTransferFrom(
176   address from,
177   address to,
178   uint256 tokenId,
179   bytes memory data
180 ) public virtual override {
181   require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not
182   token owner nor approved");
182   _safeTransfer(from, to, tokenId, data);
183 }

// (...SNIPPED...)

232 function _isApprovedOrOwner(address spender, uint256 tokenId) internal view
233   virtual returns (bool) {
234   address owner = ERC721.ownerOf(tokenId);
234   return (spender == owner || isApprovedForAll(owner, spender) ||
234   getApproved(tokenId) == spender);
```

235 }

Listing 2.4 The `safeTransferFrom` and `_isApprovedOrOwner` function

Recommendations

We recommend including a hook function that allows `_routerAddress` to perform tokenId activities after the token has been transferred. This means that whenever a token owner transfers a token to another account, this hook function is invoked, the `_routerAddress` will always be granted approval to perform tokenId activities.

Property.sol

```

146  function _afterTokenTransfer(
147      address from,
148      address to,
149      uint256 tokenId
150  ) internal override(ERC721) {
151      super._afterTokenTransfer(from, to, tokenId);
152      _setApprovalForAll(to, _routerAddress, true);
153  }

```

Listing 2.5 Improved *Property* contract

Please note that although we recommend implementing the hook function as described, there may still be a scenario that cannot be resolved. Specifically, if a token receiver revokes their own approval, the router's approval to operate the token will also be revoked, which could potentially lead to a malfunction of the `pullbackInactive` function. Nevertheless, by implementing the recommended hook function, you can help ensure proper authorization for most scenarios involving token transfers.

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 3	Lack Of Validating Master Token ID Leads To Unable To Minting Property NFTs		
Risk	Critical	Likelihood	High
	Impact	High	
Functionality is in use	In use	Status	Fixed
Associated Files	<code>contracts/XtatzuProject.sol</code> <code>contracts/Presaled.sol</code> <code>contracts/Property.sol</code>		
Locations	<code>XtatzuProject.sol L: 108 - 138</code> <code>Presaled.sol L: 43 - 55</code> <code>Property.sol L: 43 - 55</code>		

Detailed Issue

The `Presaled` contract represents the presale tokens when the project status is available, and the `Property` contract represents the property fragment tokens when the project status is finished.

When the project is finished, the `claim` function of the `XtatzuProject` contract (L162 in code snippet 3.1) allows the operator to claim their fragments tokens by **burning their presale tokens and then minting the fragment tokens with the same token list** (L161 - 162 in code snippet 3.1).

We noticed that the `mintFragment` function of the `Property` contract (L65 - 76 in code snippet 3.2) allows the operator to mint any token IDs except the **token ID 0, which is reserved to represent the master token that is owned by the `Property` contract** (L66 in code snippet 21.2) and must exist before the claim process.

The issue occurs when invoking the `claim` function if the `tokenList` (L151 in code snippet 3.1) contains the token ID 0, the invocation of the `mintFragment` function (L162 in code snippet 3.1) would be reverted because the token ID 0 token already exists (L71 in code snippet 3.2).

The root cause of this issue is due to the `mint` function of the `Presaled` contract (L43 - 55 in code snippet 3.3) allows the operator to mint the token ID 0, which crashes the master token of the `Property` contract later.

XtatuzProject.sol

```

150  function claim(address member_) public onlyOperator whenNotPaused {
151      uint256[] memory tokenList =
152          IPresaled(_presaledAddress).getPresaledOwner(member_);
153      require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
154      require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
155          PROJECT_UNFINISH");
156
157      IProperty property = IProperty(_propertyAddress);
158      IPresaled presaled = IPresaled(_presaledAddress);
159
160      bool isMintedMaster = property.isMintedMaster();
161      require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");
162
163      presaled.burn(tokenList);
164      property.mintFragment(member_, tokenList);
165  }

```

Listing 3.1 The *claim* function that burns the presale token
and mint the property fragment

Property.sol

```

65  function mintFragment(address to, uint256[] memory tokenIdList) public
onlyOperator {
66      require(ownerOf(0) == address(this), "PROPERTY: NO_MASTER_NFT");
67      require(isMintedMaster == true, "PROPERTY: MASTER_NOT_MINTED");
68      uint256 amount = tokenIdList.length;
69      for (uint256 index = 0; index < amount; index++) {
70          uint256 tokenId = tokenIdList[index];
71          require(!_exists(tokenId) == false, "PROJECT: ALREADY_EXISTS");
72          _safeMint(to, tokenId);
73      }
74      _setApprovalForAll(to, _routerAddress, true);
75      emit MintFragment(to, tokenIdList);
76  }

```

Listing 3.2 The *mintFragment* function that
mint the property fragment

Presaled.sol

```

43  function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
44    require(to != address(0), "Presaled: Reciever address is address 0");
45    uint256 amount = tokenIdList_.length;
46    for (uint256 index = 0; index < amount; index++) {
47      uint256 tokenId = tokenIdList_[index];
48      require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
49      _safeMint(to, tokenId);
50      _mintedTimestamp[tokenId] = block.timestamp;
51      _tokenIdCount.increment();
52    }
53    setApprovalForAll(_operator, true);
54    emit Minted(to, tokenIdList_, tokenIdList_.length);
55 }
```

Listing 3.3 The *mint* function
that lack of validated token ID 0

Recommendations

We recommend adding the **require** statement (L48 in the code snippet below) to prevent the operator from mistakenly minting the token ID 0 on the *Presaled* contract.

Presaled.sol

```

43  function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
44    require(to != address(0), "Presaled: Reciever address is address 0");
45    uint256 amount = tokenIdList_.length;
46    for (uint256 index = 0; index < amount; index++) {
47      uint256 tokenId = tokenIdList_[index];
48      require(tokenId > 0, "PROJECT: NOT_ALLOWED_TO_MINT_MASTER");
49      require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
50      _safeMint(to, tokenId);
51      _mintedTimestamp[tokenId] = block.timestamp;
52      _tokenIdCount.increment();
53    }
54    setApprovalForAll(_operator, true);
55    emit Minted(to, tokenIdList_, tokenIdList_.length);
56 }
```

Listing 3.4 The improved *mint* function
that not allow to mint token ID 0

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 4	Lack Of Validating The Project Status		
Risk	Critical	Likelihood	High
Functionality is in use	In use	Impact	High
Associated Files	<code>contracts/XtatuzRouter.sol</code> <code>contracts/XtatuzProject.sol</code>		
Locations	<code>XtatuzRouter.sol L: 108 - 138</code> <code>XtatuzProject.sol L:120 - 148</code>		

Detailed Issue

A user can purchase the specified presale tokens using the `addProjectMember` function of the `XtatuzRouter` contract as shown in the code snippet 4.1.

However, we discovered that **although the specified Xtatuz project was already finished, a user could still buy the presale tokens of that Xtatuz project** since there is no validation of the project status, as shown in the code snippet 4.2, resulting in the inconsistency of the associated contract states.

XtatuzRouter.sol

```

108  function addProjectMember(
109      uint256 projectId_,
110      uint256[] memory nftList_,
111      string memory referral_
112  ) public {
113      uint256 amount = nftList_.length;
114      address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
115      IXtatuzProject project = IXtatuzProject(projectAddress);
116      uint256 price = project.addProjectMember(msg.sender, nftList_);
117
118      uint256 minPrice = project.minPrice();
119      IXtatuzReferral referralContract = IXtatuzReferral(_referralAddress);
120      referralContract.increaseBuyerRef(projectId_, referral_, amount * minPrice);
121
122      address tokenAddress = project.tokenAddress();
123      IERC20(tokenAddress).transferFrom(msg.sender, projectAddress, price);
124
125      uint256[] memory memberedProject = _memberedProject[msg.sender];
126      bool foundedIndex;
127      for (uint256 index = 0; index < memberedProject.length; index++) {
128          if (memberedProject[index] == projectId_) {

```

```

129         foundedIndex = true;
130     }
131   }
132   if (!foundedIndex) {
133     _memberdProject[msg.sender].push(projectId_);
134   }
135
136   _isMemberClaimed[msg.sender][projectId_] = false;
137   emit AddProjectMember(projectId_, msg.sender, referral_, price);
138 }
```

Listing 4.1 The *addProjectMember* function of the *XtatuzRouter* contract**XtatuzProject.sol**

```

120 function addProjectMember(address member_, uint256[] memory nftList_)
121   public
122   isAvailable
123   isLeftReserve
124   whenNotPaused
125   onlyOwner
126   returns (uint256)
127 {
128   uint256 amount = nftList_.length;
129   require(amount > 0, "PROJECT: ZERO_AMOUNT");
130   for(uint i = 0; i < amount; ++i){
131     require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
132   }
133   _checkAvailableNFT(nftList_);
134
135   uint256 price = minPrice * amount;
136   projectValue += price;
137
138   _pickupAvailableNFT(nftList_, member_);
139
140   countReserve -= amount;
141
142   projectMember.push(member_);
143
144   IPresaled(_presaledAddress).mint(member_, nftList_);
145   emit AddProjectMember(projectId, member_, price);
146
147   return price;
148 }
```

Listing 4.2 The *addProjectMember* function of the *XtatuzProject* contract

Recommendations

We recommend **validating the project status** in the `addProjectMember` function of the `XtatuzProject` contract to prevent adding a project member after the project has finished as shown in the code snippet below.

XtatuzProject.sol

```

120  function addProjectMember(address member_, uint256[] memory nftList_)
121    public
122    isAvailable
123    isLeftReserve
124    whenNotPaused
125    onlyOwner
126    returns (uint256)
127  {
128    require(projectStatus() == IXtatuzProject.Status.AVAILABLE, "PROJECT:
129    PROJECT_UNAVAILABLE");
130    uint256 amount = nftList_.length;
131    require(amount > 0, "PROJECT: ZERO_AMOUNT");
132    for(uint i = 0; i < amount; ++i){
133      require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
134    }
135    _checkAvailableNFT(nftList_);
136
137    uint256 price = minPrice * amount;
138    projectValue += price;
139
140    _pickupAvailableNFT(nftList_, member_);
141
142    countReserve -= amount;
143
144    projectMember.push(member_);
145
146    IPresaled(_presaledAddress).mint(member_, nftList_);
147    emit AddProjectMember(projectId, member_, price);
148
149  }

```

Listing 4.3 The improved `addProjectMember` function

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

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 5	Function Calls Ordering Issues On Project Finalization Process		
Risk	Critical	Likelihood	High
Functionality is in use	In use	Impact	High
Associated Files	<i>contract/XtatuzProject.sol</i> <i>contract/Property.sol</i>		
Locations	<i>XtatuzProject.sol L: 175 - 188, L: 197 - 204</i> <i>Property.sol L: 46 - 51</i>		

Detailed Issue

The *finishProject* function enables the only operator to finalize the project by invoking the function, which triggers the distribution of funds to the *project owner*, *referral contract*, and *Xtatuz wallet*.

However, we identified two malfunctions related to the finalization of the project that could occur during the operation.

1. Unable to finish the project.

- In this case, if the *operator* and *trustee* call the *_multiSigMint* function before calling the *finishProject* function, the *finishProject* function will be unable to be called, and the transaction will revert. The root cause of this issue is that the condition on *_multiSigMint[_operatorAddress] && _multiSigMint[_trusteeAddress]* will pass (L200 in code snippet 5.2), causing the master token to be minted to the system. As a result, when the *operator* attempts to invoke the *finishProject* function after that, the transaction will revert because the master token has already been minted (L47 in code snippet 5.3), and the funds will not be distributed to the beneficiaries.

2. Fund transferred without minting master token

- In this case, if the *operator* calls the *finishProject* function before calling the *multiSigMint* function, the funds will be distributed to the beneficiaries immediately without minting the master token. However, this can lead to inconsistencies in the system in the event that the operator and trustee incorrectly call the *multiSigMint* function.

In both of the malfunctions we described, the issue was caused by a problem with the order of function calls.

XtatuzProject.sol

```

175   function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
176     onlyOperator {
177       address referralAddress = IXtatuzRouter(owner()).referralAddress();
178
179       isFinished = true;
180       multiSigMint();
181
182       uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
183       uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
184       IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
185       IERC20(tokenAddress).transfer(referralAddress, referralAmount);
186       IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
187
188       emit FinishProject(projectId, xstatuzWallet_);
189     }

```

Listing 5.1 The *finishProject* function**XtatuzProject.sol**

```

197   function multiSigMint() public isFullReserve spvAndTrustee {
198     _multiSigMint[msg.sender] = true;
199
200     if (_multiSigMint[_operatorAddress] && _multiSigMint[_trusteeAddress]) {
201       IProperty(_propertyAddress).mintMaster();
202       checkCanClaim = true;
203     }
204   }

```

Listing 5.2 The *multiSigMint* function**Property.sol**

```

46   function mintMaster() public onlyOwner {
47     require(isMintedMaster == false, "PROPERTY: MASTER_MINTED");
48     _mint(address(this), 0);
49     isMintedMaster = true;
50     emit MasterMinted(msg.sender);
51   }

```

Listing 5.3 The *mintMaster* function

Recommendations

To address the malfunctions related to the finalization of the project, we recommend enforcing the correct order of function calls. First, the trustee should call the `_multiSigMint` function to set their approval for minting the master token. Then, the operator should call the `finishProject` function to minting the master token and distribute funds to the beneficiaries. This order of function calls would ensure that the master token is minted before the funds are distributed, preventing inconsistencies in the system.

XstatuzProject.sol

```

177  function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
onlyOperator {
178    _multiSigMint[msg.sender] = true;
179    require(_multiSigMint[_operatorAddress] && _multiSigMint[_trusteeAddress],
"PROJECT: NOT_ALLOWS_BY_MULTISIGMINT");
180
181    isFinished = true;
182    checkCanClaim = true;
183    IProperty(_propertyAddress).mintMaster();
184
185    address referralAddress = IXstatuzRouter(owner()).referralAddress();
186    uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
187    uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
188    IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
189    IERC20(tokenAddress).transfer(referralAddress, referralAmount);
190    IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
191
192    emit FinishProject(projectId, xstatuzWallet_);
193 }
```

Listing 5.4 The improved `finishProject` function

XstatuzProject.sol

```

202  function multiSigMint() public isFullReserve spvAndTrustee {
203    _multiSigMint[msg.sender] = true;
204 }
```

Listing 5.5 The improved `multiSigMint` function

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 6	Out-Of-Sync Operator Transfer		
Risk	Critical	Likelihood	High
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 265 - 267 and L: 299 - 303</i>		

Detailed Issue

We noticed that the *Presaled*, *Property*, and *XtatuzProject* contracts introduce an operator role, which enables the management of crucial functionalities (as shown in code snippets 6.2 and 6.3).

However, we considered a potential issue whereby, in the event that the operator transfers their role to a new operator address on the *XtatuzProject* contract via the *transferOperator* function, the operator address in the *Presaled* and *Property* contracts will not be updated accordingly.

As a result, **the old operator will still have access to operate on the *Presaled* and *Property* contracts even after the *XtatuzProject* contract has transferred the operator role to the new operator.**

XtatuzProject.sol	
265	function transferOperator(address newOperator_) public whenNotPaused onlyOperator {
266	_transferOperator(newOperator_);
267	}
	//(...SNIPPED...)
299	function _transferOperator(address newOperator_) internal ProhibitZeroAddress(newOperator_) {
300	address prevOperator = _operatorAddress;
301	_operatorAddress = newOperator_;
302	emit OperatorTransferred(prevOperator, newOperator_);
303	}

Listing 6.1 The *transferOperator* function of the *XtatuzProject* contract

Property.sol

```

14  constructor(
15      string memory name_,
16      string memory symbol_,
17      address operator_,
18      address routerAddress_,
19      uint256 count_
20  ) ERC721(name_, symbol_) {
21      _setOperator(operator_);
22      setPropertyStatus(IProperty.PropertyStatus.INCOMPLETE);
23      _routerAddress = routerAddress_;
24      count = count_;
25  }
26
27  address private _operator;
28
29 //(...SNIPPED...)
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45  function mintFragment(address to, uint256[] memory tokenIdList) public
46  onlyOperator {
47      require(ownerOf(0) == address(this), "PROPERTY: NO_MASTER_NFT");
48      require(isMintedMaster == true, "PROPERTY: MASTER_NOT_MINTED");
49      uint256 amount = tokenIdList.length;
50      for (uint256 index = 0; index < amount; index++) {
51          uint256 tokenId = tokenIdList[index];
52          require(!_exists(tokenId) == false, "PROJECT: ALREADY_EXISTS");
53          _safeMint(to, tokenId);
54      }
55      _setApprovalForAll(to, _routerAddress, true);
56      emit MintFragment(to, tokenIdList);
57  }
58 }
```

Listing 6.2 The example operator privilege of the *Property* contract**Presaled.sol**

```

12  address private _operator;
13
14 //(...SNIPPED...)
15
16
17
18
19  constructor(
20      string memory _name,
21      string memory _symbol,
22      uint256 count_,
23      address operator_,
24      address routerAddress_
25  ) ERC721(_name, _symbol) {
26      _setOperator(operator_);
27      _tokenIdCount.increment();
```

```

28     _routerAddress = routerAddress_;
29     presaleIdList = new uint256[](count_);
30     for(uint256 index = 0; index < count_; index++){
31         presaleIdList[index] = index + 1;
32     }
33 }

//(...SNIPPED...)

57 function burn(uint256[] memory tokenIdList_) public onlyOperator {
58     for (uint256 index = 0; index < tokenIdList_.length; index++) {
59         _burn(tokenIdList_[index]);
60     }
61     emit Burned(tokenIdList_);
62 }
```

Listing 6.3 The example operator privilege of the *Presaled* contract

Recommendations

Since there is the need for careful consideration and management of operator roles across the *Presaled*, *Property*, and *XtatuzProject* contracts.

We recommend **adding the logic to update the operator address** on the *Presaled* and *Property* contracts whenever the *transferOperator* function is invoked on the *XtatuzProject* contract.

XtatuzProject.sol

```

299 function _transferOperator(address newOperator_) internal
300 ProhibitZeroAddress(newOperator_) {
301     address prevOperator = _operatorAddress;
302     _operatorAddress = newOperator_;
303     IPresaled(_presaledAddress).setOperator(newOperator_);
304     IProperty(_propertyAddress).setOperator(newOperator_);
305     emit OperatorTransferred(prevOperator, newOperator_);
306 }
```

Listing 6.4 The improved *_transferOperator* function of the *XtatuzProject* contract

Presaled.sol

```
104 function setOperator(address operator_) external onlyOwner{  
105     _setOperator(operator_);  
106 }
```

Listing 6.5 The new external *setOperator* function of the *Presaled* contract**Property.sol**

```
104 function setOperator(address operator_) external onlyOwner{  
105     _setOperator(operator_);  
106 }
```

Listing 6.6 The new external *setOperator* function of the *Property* contract

Furthermore, there are additional details that require updating in conjunction with this recommendation, specifically pertaining to the interface of the *Presaled* and *Property* contract.

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 7	Improper Project's Statuses		
Risk	Critical	Likelihood	High
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	<code>contracts/XtatuzProject.sol</code> <code>contracts/Property.sol</code>		
Locations	<code>XtatuzProject.sol</code> <ul style="list-style-type: none"> • L: 226 - 236 (<code>projectStatus</code> function), • L: 150 - 163 (<code>claim</code> function), • L: 165 - 173 (<code>refund</code> function), • L: 175 - 189 (<code>finishProject</code> function), <code>Property.sol</code> L: 65 - 76		

Detailed Issue

We found improper project status considerations that create three issues

1. The **FINISH** status was not considered in the case of the `_underwriteCount >= countReserve`, resulting in the project status falling into the **REFUND** instead of the **FINISH** (L229 - 230 in code snippet 7.1).
2. When the project status is **REFUND**, the user could call the `refund` function to withdraw their tokens, while the operator could still call the `finishProject` function to finish the project simultaneously.
3. Even if the project status is **REFUND**, the operator could still call the `finishProject` function to finish the project well.

XtatuzProject.sol

```

226   function projectStatus() public view returns (IXtatuzProject.Status status) {
227     if (!isFinished && block.timestamp >= startPresale && block.timestamp <=
228       endPresale && countReserve > 0) {
229       return IXtatuzProject.Status.AVAILABLE;
230     } else if (countReserve == 0 || isFinished) {
231       return IXtatuzProject.Status.FINISH;
232     } else if (block.timestamp > endPresale && countReserve > 0) {
233       return IXtatuzProject.Status.REFUND;
234     } else {
235       return IXtatuzProject.Status.UNAVAILABLE;
236     }
  
```

Listing 7.1 The improper project status consideration

To understand the first issue, please consider the scenario below.

1. The project status shows up as ***REFUND*** even if it should be marked as ***FINISH***, resulting in the user could refund their token instead of the operator could finish the project.

The root cause of this issue is that the ***FINISH*** status condition (L229 in code snippet 7.1) was not considered in the case of the `_underwriteCount >= countReserve` (L350 in code snippet 7.2), resulting in the project status falling into the ***REFUND*** (L231 in code snippet 7.1) instead.

XtatuzProject.sol

```

70  modifier isFullReserve() {
71      _checkIsFullReserve();
72      _;
73  }

// (...SNIPPED...)

175 function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
onlyOperator {
176     address referralAddress = IXtatuzRouter(owner()).referralAddress();
177
178     isFinished = true;
179     multiSigMint();
180
181     uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
182     uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
183     IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
184     IERC20(tokenAddress).transfer(referralAddress, referralAmount);
185     IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
186
187     emit FinishProject(projectId, xstatuzWallet_);
188 }

// (...SNIPPED...)

349 function _checkIsFullReserve() internal view {
350     require(_underwriteCount >= countReserve, "PROJECT: NOT_FULL");
351 }

```

Listing 7.2 The *finishProject* project function that lack of checking project status

To understand the second issue, please consider the scenario below.

1. From the first scenario result, if the project status falls to ***REFUND***, it will cause two sub issues.

1.1 If a user calls the ***refund*** function first (code snippet 7.3) and the operator then tries to invoke the ***finishProject*** function afterwards (code snippet 7.4), there may be insufficient funds in the project to pay the beneficiaries. Therefore, the transaction to finish the project would be reverted (L181 - 185 in code snippet 7.4).

1.2 Although the project status falls to ***REFUND***, the operator can still call the ***finishProject*** function (code snippet 7.4). Consequently, the project status will be changed to ***FINISH***, which will cause users to be unable to refund their tokens later.

The root cause of this scenario is the ***finishProject*** function (code snippet 7.4) does not check the project status properly before performing function functionality.

XstatuzProject.sol

```

165  function refund(address member_) public onlyOperator whenNotPaused{
166      uint256[] memory tokenList =
167      IPresaled(_presaledAddress).getPresaledOwner(member_);
168      require(projectStatus() == IXtatuzProject.Status.REFUND, "PROJECT:
169      PROJECT_UNREFUNDED");
170
171      IPresaled(_presaledAddress).burn(tokenList);
172
173      uint256 totalToken = getMemberedNFTList[member_].length * minPrice;
174      IERC20(tokenAddress).transfer(member_, totalToken);
175 }
```

Listing 7.3 The ***refund*** function that check the project status is ***REFUND***

XstatuzProject.sol

```

175  function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
onlyOperator {
176      address referralAddress = IXtatuzRouter(owner()).referralAddress();
177
178      isFinished = true;
179      multiSigMint();
180
181      uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
182      uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
183      IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
184      IERC20(tokenAddress).transfer(referralAddress, referralAmount);
185      IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
186 }
```

```
187     emit FinishProject(projectId, xstatuzWallet_);
188 }
```

Listing 7.4 The *finishProject* project function that lack of checking project status

To understand the third issue, please consider the scenario below.

1. The project status falling to ***FINISH*** from the condition of ***countReserve == 0*** (L229 in code snippet 7.1) is met.

After this step, the users could call the *claim* function immediately because the require statement ***require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT: PROJECT_UNFINISH")*** at line 153 in code snippet 7.5 is passed.

2. The user calls the ***claim*** function (code snippet 7.5) to claim their property tokens.

In this step, if the operator did not invoke the ***finishProject*** function to mint the master token before, the transaction would be reverted due to the ***require*** statement (L159 in code snippet 7.5).

XstatuzProject.sol

```
150 function claim(address member_) public onlyOperator whenNotPaused {
151     uint256[] memory tokenList =
152         IPresaled(_presaledAddress).getPresaledOwner(member_);
153     require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
154     require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
155         PROJECT_UNFINISH");
156
157     IProperty property = IProperty(_propertyAddress);
158     IPresaled presaled = IPresaled(_presaledAddress);
159
160     bool isMintedMaster = property.isMintedMaster();
161     require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");
162
163     presaled.burn(tokenList);
164     property.mintFragment(member_, tokenList);
165 }
```

Listing 7.5 The *claim* function that allows claiming token when the project is finished

Recommendations

We recommend adding the ***PREPARE_FINISH*** status (L12 in code snippet 7.6) to cover in case the project could finish and waiting for the operator to completely finish the project. Then, revise all the project status considerations in the *projectStatus* function (code snippet 7.7) to consider all possible project statuses.

Finally, adding the ***require*** statement (L176 in code snippet 7.8) to ensure that the operator could call the *finishProject* function only if the project status is ***PREPARE_FINISH***.

We provide recommended code to address this issue as a suggested remediation concept only. We recommend that the Xtatuz team should adjust the recommendation code according to the business design.

IXtatuzProject.sol

```

6  interface IXtatuzProject {
7      enum Status {
8          AVAILABLE,
9          FINISH,
10         REFUND,
11         UNAVAILABLE,
12         PREPARE_FINISH
13     }
14
15     // (...SNIPPED...)

```

Listing 7.6 The improved *Status* enum

XtatuzProject.sol

```

226     function projectStatus() public view returns (IXtatuzProject.Status status) {
227         if (isFinished) {
228             return IXtatuzProject.Status.FINISH;
229         } else if (countReserve == 0 || _underwriteCount >= countReserve) {
230             return IXtatuzProject.Status.PREPARE_FINISH;
231         } else if (block.timestamp > endPresale && countReserve > 0 &&
232 !(_underwriteCount >= countReserve)) {
233             return IXtatuzProject.Status.REFUND;
234         } else if (!isFinished && block.timestamp >= startPresale && block.timestamp
235 <= endPresale) {
236             return IXtatuzProject.Status.AVAILABLE;
237         } else {
238             return IXtatuzProject.Status.UNAVAILABLE;
239         }
240     }

```

Listing 7.7 The improved *projectStatus* function that consider all possible project statuses

XtatuzProject.sol

```

175   function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
176     onlyOperator {
177       require(projectStatus() == IXtatuzProject.Status.PREPARE_FINISH, "PROJECT:
178         NOT_READY_TO_FINISH");
179       address referralAddress = IXtatuzRouter(owner()).referralAddress();
180
181       isFinished = true;
182       multiSigMint();
183
184       uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
185       uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
186       IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
187       IERC20(tokenAddress).transfer(referralAddress, referralAmount);
188       IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
189     }
  
```

Listing 7.8 The improved *finishProject* function that checks the project status before performing the finalization process

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 8	Potential Denial-Of-Service On Adding Project Member		
Risk	Critical	Likelihood	High
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 120 - 148, 238 - 240, and 370 - 380</i>		

Detailed Issue

The `addProjectMember` function of the `XtatuzProject` (code snippet 8.1) allows the contract owner to add the new member along with minting the new unowned presale tokens to the given member address. This function contains the function calls to verify the given tokens list by calling the `_checkAvailableNFT` function and pushing the unowned token to the `unavailableNFT` array (L138 and 382 - 287 in code snippet 8.2).

The `_checkAvailableNFT` function has to iterate over all a token list and the `unavailableNFT` list, which tracks the owned token ID, to verify the given tokens (L370 - 380 in code snippet 8.2).

We noticed that the process of verifying the tokens list with the `unavailableNFT` array in the `_checkAvailableNFT` function can consume more gas than the block gas limit **if the number of owned tokens is too large, causing the transaction for adding project member to be reverted, leading to the denial-of-service issue.**

XtatuzProject.sol

```

120  function addProjectMember(address member_, uint256[] memory nftList_)
121    public
122    isAvailable
123    isLeftReserve
124    whenNotPaused
125    onlyOwner
126    returns (uint256)
127  {
128    uint256 amount = nftList_.length;
129    require(amount > 0, "PROJECT: ZERO_AMOUNT");
130    for(uint i = 0; i < amount; ++i){
131      require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
132    }
133    _checkAvailableNFT(nftList_);
134  }
```

```

135     uint256 price = minPrice * amount;
136     projectValue += price;
137
138     _pickupAvailableNFT(nftList_, member_);
139
140     countReserve -= amount;
141
142     projectMember.push(member_);
143
144     IPresaled(_presaledAddress).mint(member_, nftList_);
145     emit AddProjectMember(projectId, member_, price);
146
147     return price;
148 }

//(...SNIPPED...)

382 function _pickupAvailableNFT(uint256[] memory nftList_, address member_)
internal {
383     for (uint256 i = 0; i < nftList_.length; i++) {
384         unavailableNFT.push(nftList_[i]);
385         getMemberedNFTList[member_].push(nftList_[i]);
386     }
387 }
```

Listing 8.1 The *addProjectMember* function of the *XtatuzProject* contract**XtatuzProject.sol**

```

370 function _checkAvailableNFT(uint256[] memory nftList_) internal view {
371     for (uint256 i = 0; i < nftList_.length; i++) {
372         for (uint256 j = 0; j < unavailableNFT.length; j++) {
373             bool isAvailableNFT = false;
374             if (nftList_[i] != unavailableNFT[j]) {
375                 isAvailableNFT = true;
376             }
377             require(isAvailableNFT == true, "PROJECT: UNAVAILABLE_NFT_ID");
378         }
379     }
380 }
```

Listing 8.2 The *_checkAvailableNFT* function of the *XtatuzProject* contract

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features, we recommend redesigning and reimplementing the verification of unowned presale tokens mechanism.

Reassessment

The *Xtatuz* team acknowledged this issue. However, they tested the adding project member process with the maximum number of presale tokens (90 presale tokens) to ensure that their system can perform without creating a denial-of-service issue.

No. 9	Potential Denial-Of-Service On Retrieving Member NFTs		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<code>contracts/Presaled.sol</code> <code>contracts/XtatuzProject.sol</code> <code>contracts/XtatuzRouter.sol</code>		
Locations	<code>Presaled.sol L: 64 - 71</code> <code>XtatuzProject.sol L: 150 - 163, and 165 - 173</code> <code>XtatuzRouter.sol L: 215 - 247</code>		

Detailed Issue

The `getPresaledOwner` function is used in multiple functions across different contracts, listed below:

- The `claim` and `refund` functions of the `XtatuzProject` contract (code snippet 9.2).
These functions are used to claim the property token when the presale project finishes successfully and refund if the presale project is decided to be a refunding phrase.
- The `getAllCollection` function of the `XtatuzRouter` contract (code snippet 9.3).
This function is used to retrieve all the collections owned by the caller.

The `getPresaledOwner` function has to iterate over an owned token list of the given member in order to retrieve the corresponding token IDs (code snippet 9.1).

We noticed that this process of the `getPresaledOwner` function can consume more gas than the block gas limit, particularly if the given member has a large number of owned tokens. This excessive gas consumption may exceed the block gas limit and cause the transaction of the affected functions to be reverted, which can result in a denial-of-service issue.

XtatuzProject.sol

```

64  function getPresaledOwner(address owner) public view returns (uint256[] memory)
65  {
66      uint256 balance = balanceOf(owner);
67      uint256[] memory tokenList = new uint256[](balance);
68      for (uint256 index = 0; index < balance; index++) {
69          tokenList[index] = tokenOfOwnerByIndex(owner, index);
70      }
71  }

```

Listing 9.1 The *getPresaledOwner* function of the *Presaled* contract**XtatuzProject.sol**

```

150 function claim(address member_) public onlyOperator whenNotPaused {
151     uint256[] memory tokenList =
152         IPresaled(_presaledAddress).getPresaledOwner(member_);
153     require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
154     require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
155 PROJECT_UNFINISH");
156
157     IProperty property = IProperty(_propertyAddress);
158     IPresaled presaled = IPresaled(_presaledAddress);
159
160     bool isMintedMaster = property.isMintedMaster();
161     require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");
162
163     presaled.burn(tokenList);
164     property.mintFragment(member_, tokenList);
165 }
166
167 function refund(address member_) public onlyOperator whenNotPaused{
168     uint256[] memory tokenList =
169         IPresaled(_presaledAddress).getPresaledOwner(member_);
170     require(projectStatus() == IXtatuzProject.Status.REFUND, "PROJECT:
171 PROJECT_UNREFUNDED");
172
173     IPresaled(_presaledAddress).burn(tokenList);
174
175     uint256 totalToken = getMemberedNFTList[member_].length * minPrice;
176     IERC20(tokenAddress).transfer(member_, totalToken);
177 }

```

Listing 9.2 The *claim* and *refund* functions of the *XtatuzProject* contract

XtatuzProject.sol

```

215   function getAllCollection() public view returns (Collection[] memory) {
216     uint256[] memory projectList = _memberdProject[msg.sender];
217     Collection[] memory collections = new Collection[](projectList.length);
218     for (uint256 index = 0; index < projectList.length; index++) {
219       if (projectList[index] > 0) {
220         uint256 projectId = projectList[index];
221         address projectAddress =
222           _xtatuzFactory.getProjectAddress(projectId);
223         IXtatuzProject.Status status =
224           IXtatuzProject(projectAddress).projectStatus();
225         if (status == IXtatuzProject.Status.FINISH &&
226             _isMemberClaimed[msg.sender][projectId]) {
227           address propertyAddress =
228             _xtatuzFactory.getPropertyAddress(projectId);
229           uint256[] memory tokenList =
230             IProperty(propertyAddress).getTokenIdList(msg.sender);
231           IProperty.PropertyStatus propStatus =
232             IProperty(propertyAddress).propertyStatus();
233           CollectionType collecType = CollectionType(uint256(propStatus) +
234             1);
235           Collection memory collect = Collection({
236             contractAddress: propertyAddress,
237             tokenIdList: tokenList,
238             collectionType: collecType
239           });
240           collections[index] = collect;
241         } else {
242           address presaledAddress =
243             _xtatuzFactory.getPresaledAddress(projectId);
244           uint256[] memory tokenList =
245             IPresaled(presaledAddress).getPresaledOwner(msg.sender);
246           Collection memory collect = Collection({
247             contractAddress: presaledAddress,
248             tokenIdList: tokenList,
249             collectionType: CollectionType.PRESALE
250           });
251           collections[index] = collect;
252         }
253       }
254     }
255     return collections;
256   }

```

Listing 9.3 The *getAllCollection* function of the *XtatuzRouter* contract

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features, we recommend redesigning and reimplementing the getting owned presale tokens mechanism.

Reassessment

The *Xtatuz* team acknowledged this issue. However, they tested the retrieving member NFTs process with the maximum number of presale tokens (90 presale tokens) to ensure that their system can perform without creating a denial-of-service issue.

No. 10	Potential Denial-Of-Service On Defragmentation		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<i>contract/Property.sol</i>		
Locations	<i>Property.sol L: 78 - 86</i>		

Detailed Issue

The `defragment` function is designed to allow the owner of all fragments (*Property* tokens) to claim the master token (*Property* token ID 0). During the defragmentation process, all fragments are burned, and the master token is transferred to the eligible sender.

However, we noticed that if the number of fragments is too large (**count state variable in code snippet 10.1**), the gas required to burn them may exceed the block gas limit. This can result in the defragment transaction being reverted, which would prevent the member from claiming the master token, leading to the denial-of-service issue.

Property.sol	
78	function defragment() public {
79	require(balanceOf(msg.sender) == count, "PROPERTY:
80	ONLY_ALL_FRAGMENT_OWNER");
81	for (uint256 i = 1; i <= count; i++) {
82	_burn(i);
83	}
84	_approve(msg.sender, 0);
85	safeTransferFrom(address(this), tx.origin, 0);
86	emit Defragment(msg.sender);

Listing 10.1 The `defragment` function

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features, we recommend redesigning and reimplementing the defragmentation process.

Reassessment

The *Xtatuz* team acknowledged this issue. However, they tested the defragmentation process with the maximum number of property tokens (90 property tokens) to ensure that their system can perform without creating a denial-of-service issue.

No. 11	Potential Denial-Of-Service On Retrieving Token List		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<i>contract/Property.sol</i> <i>contract/XtatuzRouter.sol</i>		
Locations	<i>Property.sol</i> L: 101 - 108 <i>XtatuzRouter.sol</i> L: 215 - 247, L: 288 - 299		

Detailed Issue

The *gettokenIdList* function is a convenient method used to retrieve a list of all *Property* tokens owned by a specific member.

This function is used in both the *getAllCollection* and *pullbackInactive* functions as part of their processes. Specifically, the *getAllCollection* function retrieves a list of tokens and sets it as part of the collection, while *pullbackInactive* retrieves a list of tokens and pulls them back to the SPV account.

However, **if the number of tokens retrieved by *gettokenIdList* is too large (L102 - 103 in code snippet 11.1), the gas required to use the *getAllCollection* and *pullbackInactive* functions may exceed the block gas limit.** This can cause the transaction to be reverted, leading to the denial-of-service issue.

Property.sol
<pre> 101 function gettokenIdList(address member) public view returns (uint256[] memory) { 102 uint256 balance = balanceOf(member); 103 uint256[] memory tokenList = new uint256[](balance); 104 for (uint256 index = 0; index < balance; index++) { 105 tokenList[index] = tokenOfOwnerByIndex(member, index); 106 } 107 return tokenList; 108 }</pre>

Listing 11.1 The *gettokenIdList* function

XtatuzRouter.sol

```

215  function getAllCollection() public view returns (Collection[] memory) {
216      uint256[] memory projectList = _memberdProject[msg.sender];
217      Collection[] memory collections = new Collection[](projectList.length);
218      for (uint256 index = 0; index < projectList.length; index++) {
219          if (projectList[index] > 0) {
220              uint256 projectId = projectList[index];
221              address projectAddress =
222                  _xtatuzFactory.getProjectAddress(projectId);
223              IXtatuzProject.Status status =
224                  IXtatuzProject(projectAddress).projectStatus();
225              if (status == IXtatuzProject.Status.FINISH &&
226                  _isMemberClaimed[msg.sender][projectId]) {
227                  address propertyAddress =
228                      _xtatuzFactory.getPropertyAddress(projectId);
229                  uint256[] memory tokenList =
230                      IProperty(propertyAddress).getTokenIdList(msg.sender);
231                  IPProperty.PropertyStatus propStatus =
232                      IPProperty(propertyAddress).propertyStatus();
233                  CollectionType collectType = CollectionType(uint256(propStatus) +
234 1);
235                  Collection memory collect = Collection({
236                      contractAddress: propertyAddress,
237                      tokenIdList: tokenList,
238                      collectionType: collectType
239                  });
240                  collections[index] = collect;
241              } else {
242                  address presaledAddress =
243                      _xtatuzFactory.getPresaledAddress(projectId);
244                  uint256[] memory tokenList =
245                      IPresaled(presaledAddress).getPresaledOwner(msg.sender);
246                  Collection memory collect = Collection({
247                      contractAddress: presaledAddress,
248                      tokenIdList: tokenList,
249                      collectionType: CollectionType.PRESALE
250                  });
251                  collections[index] = collect;
252              }
253          }
254      }
255      return collections;
256  }

```

Listing 11.2 The *getAllCollection* function

XtatuzRouter.sol

```

288 function pullbackInactive(uint256 projectId_, address inactiveWallet_) public
289   onlySpv {
290     require(_isNotice[projectId_][inactiveWallet_] == true, "ROUTER:
291 NOTICE_BEFORE");
292     address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
293     IProperty property = IProperty(propertyAddress);
294     uint256[] memory nftList = property getTokenIdList(inactiveWallet_);
295
296     for(uint i = 0; i < nftList.length; i++){
297       IERC721(propertyAddress).safeTransferFrom(inactiveWallet_, msg.sender,
298         nftList[i]);
299     }
299 }
```

Listing 11.3 The *pullbackInactive* function

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features, we recommend redesigning and reimplementing the mechanism for obtaining a list of token IDs from members, which will also impact the related mechanisms.

Reassessment

The Xtatuz team acknowledged this issue. However, they tested the mechanism for obtaining a list of token IDs from members process with the maximum number of property tokens (90 property tokens) to ensure that their system can perform without creating a denial-of-service issue.

No. 12	Use Of Incorrect Approval Function		
Risk	High	Likelihood	High
Functionality is in use	In use	Impact	Medium
Associated Files	<i>contract/Presaled.sol</i>		
Locations	<i>Presaled.sol L: 43 - 55</i>		

Detailed Issue

The *mint* function enables the operator to mint *Presaled* tokens for the user using a list of valid token IDs. After minting, this function invokes the *setApprovalForAll* function (L53 in code snippet 12.1), which grants permission to the operator for managing all tokens owned by the user.

However, we found that the *mint* function uses the wrong function to grant approval to handle the *Presaled* tokens.

More specifically, when the *setApprovalForAll* function is invoked, the *_operator* is passed as a parameter to the function. **If the caller of the function is also the *_operator* account (e.g., if *msg.sender* is the *_operator* account), the condition *owner != operator* in the *_setApprovalForAll* function will not pass** (L372 in code snippet 12.3), causing the transaction to revert. This is because the *owner* and *operator* parameters are the same address (L368, 369 in code snippet 12.3), which means that the operator will not be granted approval to manage the presale tokens.

Presaled.sol

```

43  function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
44      require(to != address(0), "Presaled: Reciever address is address 0");
45      uint256 amount = tokenIdList_.length;
46      for (uint256 index = 0; index < amount; index++) {
47          uint256 tokenId = tokenIdList_[index];
48          require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
49          _safeMint(to, tokenId);
50          _mintedTimestamp[tokenId] = block.timestamp;
51          _tokenIdCount.increment();
52      }
53      setApprovalForAll(_operator, true);
54      emit Minted(to, tokenIdList_, tokenIdList_.length);
55  }

```

Listing 12.1 The *mint* function**Presaled.sol**

```

35  modifier onlyOperator() {
36      _checkOperator();
37      _
38  }

43 // (...SNIPPED...)

45  function _checkOperator() private view {
46      require(msg.sender == _operator || msg.sender == owner(), "PRESALED: PERMISSION_DENIED");
47  }

```

Listing 12.2 The *_checkOperator* function**ERC721.sol**

```

136  function setApprovalForAll(address operator, bool approved) public virtual override {
137      _setApprovalForAll(_msgSender(), operator, approved);
138  }

// (...SNIPPED...)

367  function _setApprovalForAll(
368      address owner,
369      address operator,
370      bool approved
371  ) internal virtual {
372      require(owner != operator, "ERC721: approve to caller");

```

```

373     _operatorApprovals[owner][operator] = approved;
374     emit ApprovalForAll(owner, operator, approved);
375 }
```

Listing 12.3 The `setApprovalForAll` and `_setApprovalForAll` function

Recommendations

We recommend revising the `mint` function by calling the `_setApprovalForAll` function with the appropriate parameters. The `mint` function can pass the `member's address` as the "owner" parameter and `_operator` as the "operator" parameter, and set the "approved" parameter to `true`. This will grant approval to the operator to handle the member's tokens without causing the transaction to revert.

Presaled.sol

```

43  function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
44    require(to != address(0), "Presaled: Reciever address is address 0");
45    uint256 amount = tokenIdList_.length;
46    for (uint256 index = 0; index < amount; index++) {
47      uint256 tokenId = tokenIdList_[index];
48      require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
49      _safeMint(to, tokenId);
50      _mintedTimestamp[tokenId] = block.timestamp;
51      _tokenIdCount.increment();
52    }
53    _setApprovalForAll(to, _operator, true);
54    emit Minted(to, tokenIdList_, tokenIdList_.length);
55 }
```

Listing 12.4 The improved `mint` function

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 13	Design Flaw In Directly Minting Presale NFTs		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/Presaled.sol</i>		
Locations	<i>Presaled.sol L: 45 - 57</i> (The <i>mint</i> function)		

Detailed Issue

The *mint* function in the *Presaled* Contract is only accessible by the operator and the contract owner, as demonstrated in lines 45 and 90 of the code snippet 13.1.

In order to mint presale tokens, the user must interact with the *XtatuzRouter* contract. The *XtatuzRouter* contract will then track the relevant state variables and make a call to the specified *XtatuzProject* contract, which will verify the conditions and handle the minting of the presale tokens. This includes updating the related state variables in the *XtatuzProject* contract before executing the *mint* function from the *Presaled* contract.

Therefore, It is imperative to adhere to the business conditions and accurately update the crucial state variables in the associated contracts when minting presale tokens.

Our investigation revealed that direct invocation of the *mint* function can result in several issues, such as

- **Mistracking of the state variables used** in the *XtatuzProject* and *XtatuzRouter* contracts.
- **Unbounded token ID** from the maximum supply and **incorrect token ID minting**.
- **Bypassing the business conditions** for minting presale tokens.

Presaled.sol

```

37  modifier onlyOperator() {
38      _checkOperator();
39      _;
40  }

41  //(...SNIPPED...)

45  function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
46      require(to != address(0), "Presaled: Reciever address is address 0");
47      uint256 amount = tokenIdList_.length;
48      for (uint256 index = 0; index < amount; index++) {
49          uint256 tokenId = tokenIdList_[index];
50          require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
51          _safeMint(to, tokenId);
52          _mintedTimestamp[tokenId] = block.timestamp;
53          _tokenIdCount.increment();
54      }
55      setApprovalForAll(_operator, true);
56      emit Minted(to, tokenIdList_, tokenIdList_.length);
57  }

58  //(...SNIPPED...)

89  function _checkOperator() private view {
90      require(msg.sender == _operator || msg.sender == owner(), "PRESALED: PERMISSION_DENIED");
91  }

```

Listing 13.1 The *mint* function of the *Presaled* contract**XtatuzProject.sol**

```

120  function addProjectMember(address member_, uint256[] memory nftList_)
121      public
122      isAvailable
123      isLeftReserve
124      whenNotPaused
125      onlyOwner
126      returns (uint256)
127  {
128      uint256 amount = nftList_.length;
129      require(amount > 0, "PROJECT: ZERO_AMOUNT");
130      for(uint i = 0; i < amount; ++i){
131          require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
132      }
133      _checkAvailableNFT(nftList_);
134
135      uint256 price = minPrice * amount;

```

```

136     projectValue += price;
137
138     _pickupAvailableNFT(nftList_, member_);
139
140     countReserve -= amount;
141
142     projectMember.push(member_);
143
144     IPresaled(_presaledAddress).mint(member_, nftList_);
145     emit AddProjectMember(projectId, member_, price);
146
147     return price;
148 }
```

Listing 13.2 The example relevant state variables of the *XtatuzProject* contract that will not be updated

Recommendations

We recommend that **the accessibility of the *mint* function should be restricted and limited to only the contract owner**. This will help ensure that the business conditions are adhered to and that the crucial state variables are accurately updated when presale tokens are minted.

Presaled.sol

```

45  function mint(address to, uint256[] memory tokenIdList_) public onlyOwner {
46    require(to != address(0), "Presaled: Reciever address is address 0");
47    uint256 amount = tokenIdList_.length;
48    for (uint256 index = 0; index < amount; index++) {
49      uint256 tokenId = tokenIdList_[index];
50      require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
51      _safeMint(to, tokenId);
52      _mintedTimestamp[tokenId] = block.timestamp;
53      _tokenIdCount.increment();
54    }
55    setApprovalForAll(_operator, true);
56    emit Minted(to, tokenIdList_, tokenIdList_.length);
57 }
```

Listing 13.3 The improved *mint* function of the *Presaled* contract

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

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 14	Design Flaw In Directly Burning Presale NFTs		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/Presaled.sol</i>		
Locations	<i>Presaled.sol L: 57 - 62 (The burn function)</i>		

Detailed Issue

The *burn* function in the *Presaled* Contract is only accessible by the operator and the contract owner, as demonstrated in lines 57 and 90 of the code snippet 14.1. This function plays a crucial role in the process of claiming property tokens and in the refund process, as demonstrated in code snippet 14.2.

In order to claim or request a refund, users must interact with the *XtatuzRouter* contract. The *XtatuzRouter* contract will then keep track of the necessary state variables and make a call to the specified *XtatuzProject* contract, which will verify the conditions and handle the claiming and/or refunding process.

Therefore, It is imperative to adhere to the business conditions and accurately update the crucial state variables in the relevant contracts when burning presale tokens.

However, **our investigation revealed that direct invocation of the *burn* function can result in various issues**, such as

- **Mismanagement of the state variables used** in the *XtatuzProject* and *XtatuzRouter* contracts.
- **Accidental burning of presale tokens** by the operator.
- **Bypassing the conditions** for burning presale tokens.

Presaled.sol

```

37  modifier onlyOperator() {
38      _checkOperator();
39      _;
40  }

41  //(...SNIPPED...)

42  function burn(uint256[] memory tokenIdList_) public onlyOperator {
43      for (uint256 index = 0; index < tokenIdList_.length; index++) {
44          _burn(tokenIdList_[index]);
45      }
46      emit Burned(tokenIdList_);
47  }

48  //(...SNIPPED...)

49  function _checkOperator() private view {
50      require(msg.sender == _operator || msg.sender == owner(), "PRESALED:");
51      require(_operator != address(0), "PRESALED:OPERATOR_IS_ZERO");
52      require(owner() != address(0), "PRESALED:OWNER_IS_ZERO");
53  }

```

Listing 14.1 The *mint* function of the *Presaled* contract**XtatuzProject.sol**

```

150  function claim(address member_) public onlyOperator whenNotPaused {
151      uint256[] memory tokenList =
152          IPresaled(_presaledAddress).getPresaledOwner(member_);
153      require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
154      require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
155          PROJECT_UNFINISH");
156
157      IProperty property = IProperty(_propertyAddress);
158      IPresaled presaled = IPresaled(_presaledAddress);
159
160      bool isMintedMaster = property.isMintedMaster();
161      require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");
162
163      presaled.burn(tokenList);
164      property.mintFragment(member_, tokenList);
165  }

166  function refund(address member_) public onlyOperator whenNotPaused{
167      uint256[] memory tokenList =
168          IPresaled(_presaledAddress).getPresaledOwner(member_);
169      require(projectStatus() == IXtatuzProject.Status.REFUND, "PROJECT:
170          PROJECT_UNREFUNDED");
171  }

```

```

169     IPresaled(_presaledAddress).burn(tokenList);
170
171     uint256 totalToken = getMemberedNFTList[member_].length * minPrice;
172     IERC20(tokenAddress).transfer(member_, totalToken);
173 }
```

Listing 14.2 The *claim* and *refund* functions of the *XtatuzProject* contract

Recommendations

We recommend that **the accessibility of the *burn* function should be restricted and limited to only the contract owner**. This will help ensure that the business conditions are adhered to and that the crucial state variables are accurately updated when presale tokens are burned.

Presaled.sol

```

57     function burn(uint256[] memory tokenIdList_) public onlyOwner {
58         for (uint256 index = 0; index < tokenIdList_.length; index++) {
59             _burn(tokenIdList_[index]);
60         }
61         emit Burned(tokenIdList_);
62     }
```

Listing 14.3 The improved *burn* function of the *Presaled* contract

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

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 15	Design Flaw In Directly Minting Property NFTs		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/Property.sol</i>		
Locations	<i>Property.sol L: 65 - 76 (The mintFragment function)</i>		

Detailed Issue

The *mintFragment* function in the *Property* Contract can be executed by the operator, the contract owner, and the router contract, as demonstrated in lines 65 and 134 of the code snippet 15.1. This function plays a crucial role in the process of claiming property tokens.

In order to claim or request a refund, users must interact with the *XtatzuzRouter* contract. The *XtatzuzRouter* contract will then keep track of the necessary state variables and make a call to the specified *XtatzuzProject* contract, which will verify the conditions and handle the claiming process.

It is important to strictly adhere to the business conditions and maintain the accuracy of the state variables in the relevant contracts during the minting of property tokens.

Our investigation has revealed that direct invocation of the *mintFragment* function can result in various issues, such as

- **Mismanagement of the state variables used in *XtatzuzRouter* contracts.**
- **Unbounded token ID** from the maximum supply.
- **Incorrect token ID minting** that potentially leads to incorrect token defragmentation due to sequential burning logic (L80 - 82 in the code snippet 15.1).
- **Bypassing the conditions** for minting property tokens.

Property.sol

```

36  modifier onlyOperator() {
37      _checkOperator();
38      _;
39  }

40  //(...SNIPPED...)

45  function mintFragment(address to, uint256[] memory tokenIdList) public
46  onlyOperator {
47      require(ownerOf(0) == address(this), "PROPERTY: NO_MASTER_NFT");
48      require(isMintedMaster == true, "PROPERTY: MASTER_NOT_MINTED");
49      uint256 amount = tokenIdList.length;
50      for (uint256 index = 0; index < amount; index++) {
51          uint256 tokenId = tokenIdList[index];
52          require(!_exists(tokenId) == false, "PROJECT: ALREADY_EXISTS");
53          _safeMint(to, tokenId);
54      }
55      _setApprovalForAll(to, _routerAddress, true);
56      emit MintFragment(to, tokenIdList);
57  }

58  function defragment() public {
59      require(balanceOf(msg.sender) == count, "PROPERTY:
60      ONLY_ALL_FRAGMENT_OWNER");
61      for (uint256 i = 1; i <= count; i++) {
62          _burn(i);
63      }
64      _approve(msg.sender, 0);
65      safeTransferFrom(address(this), tx.origin, 0);
66      emit Defragment(msg.sender);
67  }

68  //(...SNIPPED...)

132  function _checkOperator() private view {
133      require(
134          msg.sender == _operator || msg.sender == owner() || msg.sender ==
135          _routerAddress,
136          "PROPERTY: ONLY_OPERATOR"
137      );

```

Listing 15.1 The *mintFragment* and *defragment* functions of the *Property* contract

XtatuzProject.sol

```

150  function claim(address member_) public onlyOperator whenNotPaused {
151      uint256[] memory tokenList =
152          IPresaled(_presaledAddress).getPresaledOwner(member_);
153      require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
154      require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
155          PROJECT_UNFINISH");
156
157      IProperty property = IProperty(_propertyAddress);
158      IPresaled presaled = IPresaled(_presaledAddress);
159
160      bool isMintedMaster = property.isMintedMaster();
161      require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");
162
163      presaled.burn(tokenList);
164      property.mintFragment(member_, tokenList);
165  }

```

Listing 15.2 The *claim* of the *XtatuzProject* contract**XtatuzRouter.sol**

```

140  function claim(uint256 projectId_) public {
141      require(_isMemberClaimed[msg.sender][projectId_] == false, "ROUTER:
142          ALREADY_CLAIMED");
143
144      address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
145      IXtatuzProject(projectAddress).claim(msg.sender);
146
147      _isMemberClaimed[msg.sender][projectId_] = true;
148
149  }

```

Listing 15.3 The example relevant state variables of the *XtatuzRouter* contract that will not be updated

Recommendations

We recommend that **the accessibility of the *mintFragment* function should be restricted and limited to only the contract owner**. This will help ensure that the business conditions are adhered to and that the crucial state variables are accurately updated when property tokens are minted.

Property.sol

```
65 function mintFragment(address to, uint256[] memory tokenIdList) public onlyOwner
{
66     require(ownerOf(0) == address(this), "PROPERTY: NO_MASTER_NFT");
67     require(isMintedMaster == true, "PROPERTY: MASTER_NOT_MINTED");
68     uint256 amount = tokenIdList.length;
69     for (uint256 index = 0; index < amount; index++) {
70         uint256 tokenId = tokenIdList[index];
71         require(!_exists(tokenId) == false, "PROJECT: ALREADY_EXISTS");
72         _safeMint(to, tokenId);
73     }
74     _setApprovalForAll(to, _routerAddress, true);
75     emit MintFragment(to, tokenIdList);
76 }
```

Listing 15.4 The improved *mintFragment* function of the *Property* contract

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

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 16	State Inconsistency Upon Directly Claiming Property NFT Or Directly Refunding Presale NFT		
Risk	High	Likelihood	Medium
	Impact	High	
Functionality is in use	In use	Status	Fixed
Associated Files	<i>contracts/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol</i> <ul style="list-style-type: none"> • L: 150 - 163 (The <i>claim</i> function) • L: 165 - 173 (The <i>refund</i> function) 		

Detailed Issue

The *claim* and *refund* functions in the *XtatuzProject* contract can only be executed by the operator and the contract owner, as indicated in lines 150, 165 and 339 of the code snippet 16.1.

In order to claim or request a refund, users must interact with the *XtatuzRouter* contract. The *XtatuzRouter* contract will then keep track of the necessary state variables and make a call to the specified *XtatuzProject* contract for verification of the conditions and execution of the claiming and/or refunding process as in the code snippet 16.2.

However, we discovered that **direct invocation of the *claim* and *refund* functions can result in mismanagement of the state variables used in the *XtatuzRouter* contracts.**

For instance, if the operator directly invokes the *claim* function within the *XtatuzProject* contract, the *_isMemberClaimed* variable in the *XtatuzRouter* contract (L146 in the code snippet 16.2) will not be updated, potentially leading to state inconsistency.

XtatuzProject.sol

```

95  modifier onlyOperator() {
96      _checkOnlyOperator();
97      _;
98  }

//(...SNIPPED...)

150 function claim(address member_) public onlyOperator whenNotPaused {
151     uint256[] memory tokenList =
IPresaled(_presaledAddress).getPresaledOwner(member_);
152     require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
153     require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
PROJECT_UNFINISH");

154     IProperty property = IProperty(_propertyAddress);
155     IPresaled presaled = IPresaled(_presaledAddress);

156     bool isMintedMaster = property.isMintedMaster();
157     require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");

158     presaled.burn(tokenList);
159     property.mintFragment(member_, tokenList);
160 }
161

162 function refund(address member_) public onlyOperator whenNotPaused{
163     uint256[] memory tokenList =
IPresaled(_presaledAddress).getPresaledOwner(member_);
164     require(projectStatus() == IXtatuzProject.Status.REFUND, "PROJECT:
PROJECT_UNREFUNDED");

165     IPresaled(_presaledAddress).burn(tokenList);

166     uint256 totalToken = getMemberedNFTList[member_].length * minPrice;
167     IERC20(tokenAddress).transfer(member_, totalToken);
168 }

//(...SNIPPED...)

338 function _checkOnlyOperator() internal view {
339     require(msg.sender == _operatorAddress || msg.sender == owner(), "PROJECT:
NOT_OPERATOR");
340 }
```

Listing 16.1 The *claim* and *refund* functions of the *XtatuzProject* contract

XtatuzRouter.sol

```

140  function claim(uint256 projectId_) public {
141      require(_isMemberClaimed[msg.sender][projectId_] == false, "ROUTER:
142          ALREADY CLAIMED");
143
144      address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
145      IXtatuzProject(projectAddress).claim(msg.sender);
146
147      _isMemberClaimed[msg.sender][projectId_] = true;
148
149      emit Claimed(projectId_, msg.sender);
150  }
151
152  function refund(uint256 projectId_) public {
153      uint256[] memory memberdProject = _memberdProject[msg.sender];
154      address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
155
156      for (uint256 index = 0; index < memberdProject.length; index++) {
157          if (memberdProject[index] == projectId_) {
158              delete _memberdProject[msg.sender][index];
159          }
160      }
161
162      IXtatuzProject(projectAddress).refund(msg.sender);
163
164  }

```

Listing 16.2 The example relevant state variables of the *XtatuzRouter* contract that will not be updated**Recommendations**

We recommend that **the accessibility of the *claim* and *refund* functions should be restricted and limited to only the contract owner**. This will help ensure that the crucial state variables are properly updated.

XtatuzProject.sol

```

150  function claim(address member_) public onlyOwner whenNotPaused {
151      uint256[] memory tokenList =
152          IPresaled(_presaledAddress).getPresaledOwner(member_);
153      require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
154      require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
155          PROJECT_UNFINISH");
156
157      IProperty property = IProperty(_propertyAddress);
158      IPresaled presaled = IPresaled(_presaledAddress);

```

```
157
158     bool isMintedMaster = property.isMintedMaster();
159     require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");
160
161     presaled.burn(tokenList);
162     property.mintFragment(member_, tokenList);
163 }
164
165 function refund(address member_) public onlyOwner whenNotPaused{
166     uint256[] memory tokenList =
167     IPresaled(_presaledAddress).getPresaledOwner(member_);
168     require(projectStatus() == IXtatuzProject.Status.REFUND, "PROJECT:
169     PROJECT_UNREFUNDED");
170
171     IPresaled(_presaledAddress).burn(tokenList);
172
173     uint256 totalToken = getMemberedNFTList[member_].length * minPrice;
174     IERC20(tokenAddress).transfer(member_, totalToken);
175 }
```

Listing 16.3 The improved *claim* and *refund* function of the *XtatuzProject* contract

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

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 17	Possibly Insufficient Referral Fee		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/XtatuzProject.sol</i> <i>contracts/XtatuzReferral.sol</i>		
Locations	<i>XtatuzProject.sol</i> L:175 - 188 <i>XtatuzReferral.sol</i> : <ul style="list-style-type: none"> • L: 115 - 120 (The <i>setDefaultPercentage</i> function) • L: 128 - 138 (The <i>setReferralLevels</i> function) 		

Detailed Issue

The *XtatuzProject* contract allows the operator to finish the specified presale project through the *finishProject* function. **A fixed 5% fee of the project value is charged for the *XtatuzReferral* contract.**

The *XtatuzReferral* contract provides multiple levels of referral mechanisms. The percentage is used in the calculator to track the total referral amount of each agent (L71 - 98 in the code snippet 17.3). The agent can claim their commission by invoking the *claim* function (L100 - 113 in the code snippet 17.3).

However, we discovered that the referral fee percentage can be dynamically changed through the *setDefaultPercentage* and *setReferralLevels* functions, as indicated in the code snippet 17.2. **This dynamic change can result in insufficient funds being transferred to the agent when claiming.**

To elaborate, let us consider the following scenario:

Suppose that **count = 100**, **countReserve = 0**, **minPrice = 10**, and the level of the *AgentA* = 0.

1. The *XtatuzReferral* operator sets the default percentage to **10** by invoking the *setDefaultPercentage* function: ***setDefaultPercentage(default_ : 10)***.

At this step the *defaultPercentage* will contain the value of 10 percent. This percentage will be used in every buying presale tokens and tracked in the *buyerAgentAmount* mapping (L95 in the code snippet 17.3)

2. A user purchases 100 presale tokens by invoking the *addProjectMember* function and refers to the referral code of the *AgentA* (L108 - 138 in the code snippet 17.4), the ***referralAmount*** will be calculated as follows (L81 in the code snippet 17.3):

```

referralAmount = (amount_ * percentage) / 100
referralAmount = (1000 * 10) / 100; amount_ = 100 * 10
referralAmount = 100

```

At this step, the *buyerAgentAmount* mapping of *AgentA* will contain the value of **100**.

3. After the project is finished, The *XtatuzProject* owner invokes the *finishProject* function, **and 5 percent of the project value will be calculated and transferred to the *XtatuzReferral* contract.**
The *referralAmount* is calculated as follows:

```

referralAmount = (((count - countReserve) * minPrice) * 5) / 100
referralAmount = (((100 - 0) * 10) * 5) / 100
referralAmount = 50

```

Therefore, the *XtatuzProject* will transfer **50** tokens to the *XtatuzReferral* contract

Consequently, when *AgentA* claims their commission, the transaction will be reverted due to insufficient funds.

XtatuzProject.sol

```

175 function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
onlyOperator {
176     address referralAddress = IXtatuzRouter(owner()).referralAddress();
177
178     isFinished = true;
179     multiSigMint();
180
181     uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
182     uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
183     IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
184     IERC20(tokenAddress).transfer(referralAddress, referralAmount);
185     IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
186
187     emit FinishProject(projectId, xstatuzWallet_);
188 }

```

Listing 17.1 The *finishProject* function of the *XtatuzProject* contract

XtatuzReferral.sol

```

115  function setDefaultPercentage(uint256 default_) public onlyOperator {
116      require(default_ > 0, "REFERRAL: NO_ZERO_PERCENT");
117      uint256 prev = defaultPercentage;
118      defaultPercentage = default_;
119      emit ChangeDefaultPercent(prev, default_);
120  }

  //(...SNIPPED...)

128  function setReferralLevels(uint256[] memory percentagePerLevel_) public
onlyOperator {
129      uint256[] memory prevLevels = new uint256[](3);
130      uint256[] memory newLevels = new uint256[](3);
131      require(percentagePerLevel_.length == 3, "REFERRAL: 3_LEVELS");
132      for(uint256 i = 0; i < percentagePerLevel_.length ; i++){
133          prevLevels[i] = levelsPercentage[i + 1];
134          levelsPercentage[i + 1] = percentagePerLevel_[i];
135          newLevels[i] = levelsPercentage[i + 1];
136      }
137      emit SetReferralLevels(prevLevels, newLevels);
138  }

```

Listing 17.2 The *setDefaultPercentage* and *setReferralLevels* functions of the *XtatuzReferral* contract

XtatuzReferral.sol

```

71   function increaseBuyerRef(uint256 projectId_, string memory referral_, uint256
amount_) public onlyOperator {
72       address agentWallet = addressByReferral[referral_];
73       require(agentWallet != address(0), "REFERRAL: INVALID_REFERRAL");
74       uint256 level = referralLevel[projectId_][referral_];
75       uint256 percentage = defaultPercentage;

76
77       if(level != 0) {
78           percentage = levelsPercentage[level];
79       }

81       uint256 referralAmount = (amount_ * percentage) / 100;

82
83       uint256[] memory projectIdList = projectIdsByReferral[referral_];
84
85       bool foundedIndex;
86       for (uint256 index = 0; index < projectIdList.length; index++) {
87           if (projectIdList[index] == projectId_) {
88               foundedIndex = true;
89           }
90       }

```

```

91     if (!foundedIndex) {
92         projectIdsByReferral[referral_].push(projectId_);
93     }
94
95     buyerAgentAmount[referral_][projectId_] += referralAmount;
96     updateReferralAmount(projectId_, amount_);
97     emit IncreaseBuyerRef(projectId_, referralAmount);
98   }
99
100 function claim(string memory referral_, uint projectId_) public {
101     address agent = addressByReferral[referral_];
102     address projectAddress =
103 IXtatuzRouter(owner()).getProjectAddressById(projectId_);
104     require(projectAddress != address(0), "REFERRAL: INVALID_PROJECT_ID");
105
106     IXtatuzProject.Status status =
107 IXtatuzProject(projectAddress).projectStatus();
108     require(status == IXtatuzProject.Status.FINISH, "REFERRAL:
109 PROJECT_NOT_FINISH");
110     require(msg.sender == agent, "REFERRAL: INVALID_ACCOUNT");
111
112     uint256 amount = buyerAgentAmount[referral_][projectId_];
113     buyerAgentAmount[referral_][projectId_] = 0;
114
115     IERC20(tokenAddress).transfer(msg.sender, amount);
116   }

```

Listing 17.3 The *increaseBuyerRef* and *claim* functions of the *XtatuzReferral* contract**XtatuzRouter.sol**

```

108 function addProjectMember(
109     uint256 projectId_,
110     uint256[] memory nftList_,
111     string memory referral_
112 ) public {
113     uint256 amount = nftList_.length;
114     address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
115     IXtatuzProject project = IXtatuzProject(projectAddress);
116     uint256 price = project.addProjectMember(msg.sender, nftList_);
117
118     uint256 minPrice = project.minPrice();
119     IXtatuzReferral referralContract = IXtatuzReferral(_referralAddress);
120     referralContract.increaseBuyerRef(projectId_, referral_, amount * minPrice);
121
122     address tokenAddress = project.tokenAddress();
123     IERC20(tokenAddress).transferFrom(msg.sender, projectAddress, price);
124
125     uint256[] memory memberedProject = _memberedProject[msg.sender];
126     bool foundedIndex;

```

```

127     for (uint256 index = 0; index < memberedProject.length; index++) {
128         if (memberedProject[index] == projectId_) {
129             foundedIndex = true;
130         }
131     }
132     if (!foundedIndex) {
133         _memberdProject[msg.sender].push(projectId_);
134     }
135
136     _isMemberClaimed[msg.sender][projectId_] = false;
137     emit AddProjectMember(projectId_, msg.sender, referral_, price);
138 }
```

Listing 17.4 The *addProjectMember* function of the *XtatuzRouter* contract

Recommendations

We recommend **applying the upper bounding percentage** when determining the percentage and **ensuring that it is consistent with the fixed referral percentage** in the *XtatuzProject* contract and also **checking the sequential of the given level percentages** when invoking the *setReferralLevels* function.

Additionally, we strongly recommend that the *Xtatuz* team should ensure that the funds in the *XtatusRefferal* contract are sufficient to pay for all agents.

XtatuzReferral.sol

```

115 function setDefaultPercentage(uint256 default_) public onlyOperator {
116     require(default_ > 0 && default_ <= MAX_PERCENT, "REFERRAL:
117 INVALID_PERCENT");
118     uint256 prev = defaultPercentage;
119     defaultPercentage = default_;
120     emit ChangeDefaultPercent(prev, default_);
121 }

//(...SNIPPED...)

128 function setReferralLevels(uint256[] memory percentagePerLevel_) public
onlyOperator {
129     uint256[] memory prevLevels = new uint256[](3);
130     uint256[] memory newLevels = new uint256[](3);
131     require(percentagePerLevel_.length == 3, "REFERRAL: 3_LEVELS");
132     require(percentagePerLevel_[0] < percentagePerLevel_[1] &&
percentagePerLevel_[1] < percentagePerLevel_[2], "REFERRAL:
133 INVALID_PERCENT_LEVELS");
134     for(uint256 i = 0; i < percentagePerLevel_.length ; i++){
135         require(percentagePerLevel_[i] > 0 && percentagePerLevel_[i] <= MAX_PERCENT,
"REFERRAL: INVALID_PERCENT");
136         prevLevels[i] = levelsPercentage[i + 1];
137 }
```

```

136         levelsPercentage[i + 1] = percentagePerLevel_[i];
137         newLevels[i] = levelsPercentage[i + 1];
138     }
139     emit SetReferralLevels(prevLevels, newLevels);
140 }
```

Listing 17.4 The improved `setDefaultPercentage` and `setReferralLevels` functions of the `XtatuzReferral` contract

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

Reassessment

The Xtatuz team fixed this issue by introducing the constant `MAX_PERCENTAGE` variable (L25 in the code snippet below) to set an upper bound on the maximum percentage that can be applied to the referral fee percentage.

By setting this bound, the team has ensured that the referral fee percentage does not exceed the fixed 5% fee of the project value charged in the `XtatuzProject` contract.

XtatuzReferral.sol

```

10  contract XtatuzReferral is Ownable {
11      // (...SNIPPED...)
12
13      uint256 public constant MAX_PERCENTAGE = 5;
14
15      uint256 public maxPercentage = 5;
16      uint256 public defaultPercentage = 3;
17      // (...SNIPPED...)
18
19      function setDefaultPercentage(uint256 default_) public onlyOperator {
20          require(default_ > 0 && default_ <= maxPercentage, "REFERRAL:
21              INVALID_PERCENT");
22
23          uint256 prev = defaultPercentage;
24          defaultPercentage = default_;
25          emit ChangeDefaultPercent(prev, default_);
26      }
27
28
29      function setMaxPercentage(uint256 max_) public onlyOperator {
30          require(max_ > 0 && max_ <= MAX_PERCENTAGE, "REFERRAL:
31              INVALID_PERCENT");
32
33          uint256 prev = maxPercentage;
34          maxPercentage = max_;
35          emit ChangeMaxPercent(prev, max_);
36      }
37
38      // (...SNIPPED...)
39
40      function setReferralLevels(uint256[] memory percentagePerLevel_) public
```

```

151   onlyOperator {
152     uint256[] memory prevLevels = new uint256[](3);
153     uint256[] memory newLevels = new uint256[](3);
154     require(percentagePerLevel_.length == 3, "REFERRAL: 3_LEVELS");
155     require(
156       percentagePerLevel_[0] < percentagePerLevel_[1] &&
157       percentagePerLevel_[1] < percentagePerLevel_[2],
158       "REFERRAL: INVALID_PERCENT_LEVELS"
159     );
160     for (uint256 i = 0; i < percentagePerLevel_.length; i++) {
161       require(percentagePerLevel_[i] > 0 && percentagePerLevel_[i] <=
162         maxPercentage, "REFERRAL: INVALID_PERCENT");
163       prevLevels[i] = levelsPercentage[i + 1];
164       levelsPercentage[i + 1] = percentagePerLevel_[i];
165       newLevels[i] = levelsPercentage[i + 1];
166     }
167     emit SetReferralLevels(prevLevels, newLevels);
168   }
169   // (...SNIPPED...)
170 }
```

Listing 17.5 The improved setting percentage process of the *XtatuzReferral* contract

No. 18	Lack Of Updating State Variables Upon Claiming Property NFT Or Refunding Presale NFT		
Risk	High	Likelihood	Medium
		Impact	High
Functionality is in use	In use	Status	Acknowledged
Associated Files	contracts/XtatuzProject.sol		
Locations	XtatuzProject.sol L: 120 - 148, 150 - 163, 165 - 173, 370 - 380 and 382 - 387		

Detailed Issue

Inside the `_checkAvailableNFT` and `_pickupAvailableNFT` functions (L370 - 380 and 382 - 387 in code snippet 18.1) there are use the `unavailableNFT` state to track unavailable tokens (L372 and 385 in code snippet 18.1) and the `getMemberedNFTList` state to track tokens that the user owned (L386 in code snippet 18.1).

We noticed that the `addProjectMember` function invokes the `_checkAvailableNFT` (L133 in code snippet 18.2) and `_pickupAvailableNFT` (L138 in code snippet 18.2) functions to check that all tokens in the `nftList_` parameter are available and updates the `unavailableNFT` and `getMemberedNFTList` state.

However, we found that the `claim` and `refund` functions (L150 - 163 and 165 - 173 in code snippet 18.3) do not update the `unavailableNFT` and `getMemberedNFTList` states, leading to the state inconsistency issue.

XtatuzProject.sol

```

370  function _checkAvailableNFT(uint256[] memory nftList_) internal view {
371      for (uint256 i = 0; i < nftList_.length; i++) {
372          for (uint256 j = 0; j < unavailableNFT.length; j++) {
373              bool isAvailableNFT = false;
374              if (nftList_[i] != unavailableNFT[j]) {
375                  isAvailableNFT = true;
376              }
377              require(isAvailableNFT == true, "PROJECT: UNAVAILABLE_NFT_ID");
378          }
379      }
380  }
381
382  function _pickupAvailableNFT(uint256[] memory nftList_, address member_)
internal {
383      for (uint256 i = 0; i < nftList_.length; i++) {

```

```

384         unavailableNFT.push(nftList_[i]);
385         getMemberedNFTList[member_].push(nftList_[i]);
386     }
387 }
```

Listing 18.1 The `_checkAvailableNFT` and `_pickupAvailableNFT` functions that allows the operator to extend presale by condition

XtatuzProject.sol

```

120 function addProjectMember(address member_, uint256[] memory nftList_)
121   public
122   isAvailable
123   isLeftReserve
124   whenNotPaused
125   onlyOwner
126   returns (uint256)
127 {
128   uint256 amount = nftList_.length;
129   require(amount > 0, "PROJECT: ZERO_AMOUNT");
130   for(uint i = 0; i < amount; ++i){
131     require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
132   }
133   _checkAvailableNFT(nftList_);
134
135   uint256 price = minPrice * amount;
136   projectValue += price;
137
138   _pickupAvailableNFT(nftList_, member_);
139
140   countReserve -= amount;
141
142   projectMember.push(member_);
143
144   IPresaled(_presaledAddress).mint(member_, nftList_);
145   emit AddProjectMember(projectId, member_, price);
146
147   return price;
148 }
```

Listing 18.2 The `addProjectMember` function

XtatuzProject.sol

```

150  function claim(address member_) public onlyOperator whenNotPaused {
151      uint256[] memory tokenList =
152          IPresaled(_presaledAddress).getPresaledOwner(member_);
153      require(tokenList.length > 0, "PROJECT: TOKENLIST_ZERO");
154      require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
155          PROJECT_UNFINISH");
156
157      IProperty property = IProperty(_propertyAddress);
158      IPresaled presaled = IPresaled(_presaledAddress);
159
160      bool isMintedMaster = property.isMintedMaster();
161      require(isMintedMaster == true, "PROJECT: MASTER_NOT_MINTED");
162
163      presaled.burn(tokenList);
164      property.mintFragment(member_, tokenList);
165  }
166
167  function refund(address member_) public onlyOperator whenNotPaused{
168      uint256[] memory tokenList =
169          IPresaled(_presaledAddress).getPresaledOwner(member_);
170      require(projectStatus() == IXtatuzProject.Status.REFUND, "PROJECT:
171          PROJECT_UNREFUNDED");
172
173      IPresaled(_presaledAddress).burn(tokenList);
174
175      uint256 totalToken = getMemberedNFTList[member_].length * minPrice;
176      IERC20(tokenAddress).transfer(member_, totalToken);
177  }

```

Listing 18.3 The *claim* and *refund* functions that lack of update states

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features, we recommend redesigning and reimplementing both *claim* and *refund* functions and their related subsystems to update the *unavailableNFT* and *getMemberedNFTList* states properly.

Reassessment

The Xtatuz team acknowledged this issue. Since the state variables mentioned would no longer be used once the project status is *Finish* or *Refund*.

No. 19	Possibly Unable To Finish Project		
Risk	High	Likelihood	Medium
Functionality is in use	In use	Impact	High
Associated Files	<i>contract/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 175 - 188</i>		

Detailed Issue

The `addProjectMember` function enables users to purchase presale tokens, while the `finishProject` function allows the operator to finalize the project and distribute the funds to relevant parties, including the *project owner, referral contract, and Xtatuz wallet*.

We found that the `projectValue` state variable is increased with each presale token purchase (L136 in code snippet 19.1) and is used to calculate the funds to be distributed to the project owner (L183 in code snippet 19.2).

However, we noticed that the `xtatuzAmount` variable (L182 in code snippet 19.2) stores 10% of the whole project value. This means that if the `projectValue` is less than 10% of the total project value, the formula `projectValue - xtatuzAmount` will result in a negative value, which will prevent the operator from completing the project.

In addition, we observed that the project finishing condition `_underwriteCount >= countReserve` permits a project to be completed even if the presale does not sell at least 10% of the project's value.

As a result, if the presale falls short of selling at least 10% of the project's value, the transaction will revert, and the operator will be unable to complete the project.

XtatuzProject.sol
<pre> 120 function addProjectMember(address member_, uint256[] memory nftList_) 121 public 122 isAvailable 123 isLeftReserve 124 whenNotPaused 125 onlyOwner 126 returns (uint256) </pre>

```

127  {
128      uint256 amount = nftList_.length;
129      require(amount > 0, "PROJECT: ZERO_AMOUNT");
130      for(uint i = 0; i < amount; ++i){
131          require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
132      }
133      _checkAvailableNFT(nftList_);
134
135      uint256 price = minPrice * amount;
136      projectValue += price;
137
138      _pickupAvailableNFT(nftList_, member_);
139
140      countReserve -= amount;
141
142      projectMember.push(member_);
143
144      IPresaled(_presaledAddress).mint(member_, nftList_);
145      emit AddProjectMember(projectId, member_, price);
146
147      return price;
148  }

```

Listing 19.1 The *addProjectMember* function**XstatuzProject.sol**

```

175  function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
onlyOperator {
176      address referralAddress = IXstatuzRouter(owner()).referralAddress();
177
178      isFinished = true;
179      multiSigMint();
180
181      uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
182      uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
183      IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
184      IERC20(tokenAddress).transfer(referralAddress, referralAmount);
185      IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
186
187      emit FinishProject(projectId, xstatuzWallet_);
188  }

```

Listing 19.2 The *finishProject* function

Recommendations

To prevent issues with completing the project, **we recommend that the operator ensures the presale tokens are sold for at least 10% of the total project value**. This can be achieved by setting a minimum target for presale token purchases.

Additionally, we noticed that one of the conditions for finishing a project is that it must achieve a sales threshold `_underwriteCount >= countReserve`, as indicated on line 350 in code snippet 19.3. This expression is checking whether the sales threshold required to complete the project (`_underwriteCount`) is greater than or equal to the number of remaining NFT tokens available for purchase (`countReserve`).

Therefore, one possible mitigating solution is to ensure that the sales threshold (`_underwriteCount`) is set at a level that is more than 10% of the maximum NFT tokens available as shown in the code snippet 19.4.

XstatuzProject.sol

```

70  modifier isFullReserve() {
71      _checkIsFullReserve();
72  }
73 }

// (...SNIPPED...)

349 function _checkIsFullReserve() internal view {
350     require(_underwriteCount >= countReserve, "PROJECT: NOT_FULL");
351 }
```

Listing 19.3 The `isFullReserve` modifier

XstatuzProject.sol

```

111 function setUnderwriteCount(uint256 underwriteCount_) public onlyOperator {
112     uint256 threshold = (count - underwriteCount_) * 100 / count;
113     require(threshold >= 10, "PROJECT: INVALID_UNDERWRITE_COUNT");
114     _underwriteCount = underwriteCount_;
115 }
```

Listing 19.4 The `setUnderwriteCount` function

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

Reassessment

This issue was acknowledged by the Xtatuz team and they have guaranteed that presale tokens will be sold for more than 10% of the total project value.

No. 20	Arbitrarily Setting Presale Period		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	contracts/XtatuzRouter.sol contracts/XtatuzProject.sol contracts/Property.sol		
Locations	XtatuzRouter.sol L: 74 - 106 XtatuzProject.sol L: 105 - 109, 218 - 224, and 273 -289		

Detailed Issue

We noticed that the **setPresalePeriod** function was invoked once (L104 in code snippet 20.1) after the project contract was created (L103 in code snippet 20.2) to set the presale period.

The **extendEndPresale** function (L218 - 224 in code snippet 20.2) allows the operator to extend the presale one time by considering the remaining time (L274 - 288 in code snippet 20.2).

However, we found that the **setPresalePeriod** function (L105 - 109 in code snippet 20.3) conflicts with the **_extendEndPresale** function logic (L273 - 289 in code snippet 20.2), **in addition to being invoked once when creating an XtatuzProject contract, it allows the operator sets arbitrary presale period immediately** (L103 in code snippet 20.1).

XtatuzRouter.sol

```

74   function createProject(
75     uint256 count_,
76     uint256 underwriteCount_,
77     address tokenAddress_,
78     string memory name_,
79     string memory symbol_,
80     uint256 startPresale_,
81     uint256 endPresale_
82   ) public onlySpv {
83
84     // (...SNIPPED...)
85
86     IXtatuzFactory.ProjectPrepareData memory data =
87     IXtatuzFactory.ProjectPrepareData({
88       projectId_: projectId,
89       spv_: msg.sender,
90     );
91
92     bytes memory dataBytes = abi.encode(data);
93
94     bytes32 id = keccak256(dataBytes);
95
96     bytes32[] memory projectIds = new bytes32[](1);
97     projectIds[0] = id;
98
99     bytes32[] memory projectNames = new bytes32[](1);
100    projectNames[0] = name_;
101
102    bytes32[] memory projectSymbols = new bytes32[](1);
103    projectSymbols[0] = symbol_;
104
105    bytes32[] memory projectStartPresale = new bytes32[](1);
106    projectStartPresale[0] = startPresale_;
107
108    bytes32[] memory projectEndPresale = new bytes32[](1);
109    projectEndPresale[0] = endPresale_;
110
111    bytes32[] memory projectUnderwriteCount = new bytes32[](1);
112    projectUnderwriteCount[0] = underwriteCount_;
113
114    bytes32[] memory projectTokenAddress = new bytes32[](1);
115    projectTokenAddress[0] = tokenAddress_;
116
117    bytes32[] memory projectCount = new bytes32[](1);
118    projectCount[0] = count_;
119
120    bytes32[] memory projectSpv = new bytes32[](1);
121    projectSpv[0] = msg.sender;
122
123    bytes32[] memory projectUnderwrite = new bytes32[](1);
124    projectUnderwrite[0] = underwriteCount_;
125
126    bytes32[] memory projectId = new bytes32[](1);
127    projectId[0] = id;
128
129    bytes32[] memory projectSymbol = new bytes32[](1);
130    projectSymbol[0] = symbol_;
131
132    bytes32[] memory projectName = new bytes32[](1);
133    projectName[0] = name_;
134
135    bytes32[] memory projectStartPresaleTime = new bytes32[](1);
136    projectStartPresaleTime[0] = startPresale_;
137
138    bytes32[] memory projectEndPresaleTime = new bytes32[](1);
139    projectEndPresaleTime[0] = endPresale_;
140
141    bytes32[] memory projectTokenAddress2 = new bytes32[](1);
142    projectTokenAddress2[0] = tokenAddress_;
143
144    bytes32[] memory projectCount2 = new bytes32[](1);
145    projectCount2[0] = count_;
146
147    bytes32[] memory projectSpv2 = new bytes32[](1);
148    projectSpv2[0] = msg.sender;
149
150    bytes32[] memory projectUnderwrite2 = new bytes32[](1);
151    projectUnderwrite2[0] = underwriteCount_;
152
153    bytes32[] memory projectId2 = new bytes32[](1);
154    projectId2[0] = id;
155
156    bytes32[] memory projectSymbol2 = new bytes32[](1);
157    projectSymbol2[0] = symbol_;
158
159    bytes32[] memory projectName2 = new bytes32[](1);
160    projectName2[0] = name_;
161
162    bytes32[] memory projectStartPresaleTime2 = new bytes32[](1);
163    projectStartPresaleTime2[0] = startPresale_;
164
165    bytes32[] memory projectEndPresaleTime2 = new bytes32[](1);
166    projectEndPresaleTime2[0] = endPresale_;
167
168    bytes32[] memory projectTokenAddress3 = new bytes32[](1);
169    projectTokenAddress3[0] = tokenAddress_;
170
171    bytes32[] memory projectCount3 = new bytes32[](1);
172    projectCount3[0] = count_;
173
174    bytes32[] memory projectSpv3 = new bytes32[](1);
175    projectSpv3[0] = msg.sender;
176
177    bytes32[] memory projectUnderwrite3 = new bytes32[](1);
178    projectUnderwrite3[0] = underwriteCount_;
179
180    bytes32[] memory projectId3 = new bytes32[](1);
181    projectId3[0] = id;
182
183    bytes32[] memory projectSymbol3 = new bytes32[](1);
184    projectSymbol3[0] = symbol_;
185
186    bytes32[] memory projectName3 = new bytes32[](1);
187    projectName3[0] = name_;
188
189    bytes32[] memory projectStartPresaleTime3 = new bytes32[](1);
190    projectStartPresaleTime3[0] = startPresale_;
191
192    bytes32[] memory projectEndPresaleTime3 = new bytes32[](1);
193    projectEndPresaleTime3[0] = endPresale_;
194
195    bytes32[] memory projectTokenAddress4 = new bytes32[](1);
196    projectTokenAddress4[0] = tokenAddress_;
197
198    bytes32[] memory projectCount4 = new bytes32[](1);
199    projectCount4[0] = count_;
200
201    bytes32[] memory projectSpv4 = new bytes32[](1);
202    projectSpv4[0] = msg.sender;
203
204    bytes32[] memory projectUnderwrite4 = new bytes32[](1);
205    projectUnderwrite4[0] = underwriteCount_;
206
207    bytes32[] memory projectId4 = new bytes32[](1);
208    projectId4[0] = id;
209
210    bytes32[] memory projectSymbol4 = new bytes32[](1);
211    projectSymbol4[0] = symbol_;
212
213    bytes32[] memory projectName4 = new bytes32[](1);
214    projectName4[0] = name_;
215
216    bytes32[] memory projectStartPresaleTime4 = new bytes32[](1);
217    projectStartPresaleTime4[0] = startPresale_;
218
219    bytes32[] memory projectEndPresaleTime4 = new bytes32[](1);
220    projectEndPresaleTime4[0] = endPresale_;
221
222    bytes32[] memory projectTokenAddress5 = new bytes32[](1);
223    projectTokenAddress5[0] = tokenAddress_;
224
225    bytes32[] memory projectCount5 = new bytes32[](1);
226    projectCount5[0] = count_;
227
228    bytes32[] memory projectSpv5 = new bytes32[](1);
229    projectSpv5[0] = msg.sender;
230
231    bytes32[] memory projectUnderwrite5 = new bytes32[](1);
232    projectUnderwrite5[0] = underwriteCount_;
233
234    bytes32[] memory projectId5 = new bytes32[](1);
235    projectId5[0] = id;
236
237    bytes32[] memory projectSymbol5 = new bytes32[](1);
238    projectSymbol5[0] = symbol_;
239
240    bytes32[] memory projectName5 = new bytes32[](1);
241    projectName5[0] = name_;
242
243    bytes32[] memory projectStartPresaleTime5 = new bytes32[](1);
244    projectStartPresaleTime5[0] = startPresale_;
245
246    bytes32[] memory projectEndPresaleTime5 = new bytes32[](1);
247    projectEndPresaleTime5[0] = endPresale_;
248
249    bytes32[] memory projectTokenAddress6 = new bytes32[](1);
250    projectTokenAddress6[0] = tokenAddress_;
251
252    bytes32[] memory projectCount6 = new bytes32[](1);
253    projectCount6[0] = count_;
254
255    bytes32[] memory projectSpv6 = new bytes32[](1);
256    projectSpv6[0] = msg.sender;
257
258    bytes32[] memory projectUnderwrite6 = new bytes32[](1);
259    projectUnderwrite6[0] = underwriteCount_;
260
261    bytes32[] memory projectId6 = new bytes32[](1);
262    projectId6[0] = id;
263
264    bytes32[] memory projectSymbol6 = new bytes32[](1);
265    projectSymbol6[0] = symbol_;
266
267    bytes32[] memory projectName6 = new bytes32[](1);
268    projectName6[0] = name_;
269
270    bytes32[] memory projectStartPresaleTime6 = new bytes32[](1);
271    projectStartPresaleTime6[0] = startPresale_;
272
273    bytes32[] memory projectEndPresaleTime6 = new bytes32[](1);
274    projectEndPresaleTime6[0] = endPresale_;
275
276    bytes32[] memory projectTokenAddress7 = new bytes32[](1);
277    projectTokenAddress7[0] = tokenAddress_;
278
279    bytes32[] memory projectCount7 = new bytes32[](1);
280    projectCount7[0] = count_;
281
282    bytes32[] memory projectSpv7 = new bytes32[](1);
283    projectSpv7[0] = msg.sender;
284
285    bytes32[] memory projectUnderwrite7 = new bytes32[](1);
286    projectUnderwrite7[0] = underwriteCount_;
287
288    bytes32[] memory projectId7 = new bytes32[](1);
289    projectId7[0] = id;
290
291    bytes32[] memory projectSymbol7 = new bytes32[](1);
292    projectSymbol7[0] = symbol_;
293
294    bytes32[] memory projectName7 = new bytes32[](1);
295    projectName7[0] = name_;
296
297    bytes32[] memory projectStartPresaleTime7 = new bytes32[](1);
298    projectStartPresaleTime7[0] = startPresale_;
299
299    bytes32[] memory projectEndPresaleTime7 = new bytes32[](1);
300    projectEndPresaleTime7[0] = endPresale_;
301
301    bytes32[] memory projectTokenAddress8 = new bytes32[](1);
302    projectTokenAddress8[0] = tokenAddress_;
303
303    bytes32[] memory projectCount8 = new bytes32[](1);
304    projectCount8[0] = count_;
305
305    bytes32[] memory projectSpv8 = new bytes32[](1);
306    projectSpv8[0] = msg.sender;
307
307    bytes32[] memory projectUnderwrite8 = new bytes32[](1);
308    projectUnderwrite8[0] = underwriteCount_;
309
309    bytes32[] memory projectId8 = new bytes32[](1);
310    projectId8[0] = id;
311
311    bytes32[] memory projectSymbol8 = new bytes32[](1);
312    projectSymbol8[0] = symbol_;
313
313    bytes32[] memory projectName8 = new bytes32[](1);
314    projectName8[0] = name_;
315
315    bytes32[] memory projectStartPresaleTime8 = new bytes32[](1);
316    projectStartPresaleTime8[0] = startPresale_;
317
317    bytes32[] memory projectEndPresaleTime8 = new bytes32[](1);
318    projectEndPresaleTime8[0] = endPresale_;
319
319    bytes32[] memory projectTokenAddress9 = new bytes32[](1);
320    projectTokenAddress9[0] = tokenAddress_;
321
321    bytes32[] memory projectCount9 = new bytes32[](1);
322    projectCount9[0] = count_;
323
323    bytes32[] memory projectSpv9 = new bytes32[](1);
324    projectSpv9[0] = msg.sender;
325
325    bytes32[] memory projectUnderwrite9 = new bytes32[](1);
326    projectUnderwrite9[0] = underwriteCount_;
327
327    bytes32[] memory projectId9 = new bytes32[](1);
328    projectId9[0] = id;
329
329    bytes32[] memory projectSymbol9 = new bytes32[](1);
330    projectSymbol9[0] = symbol_;
331
331    bytes32[] memory projectName9 = new bytes32[](1);
332    projectName9[0] = name_;
333
333    bytes32[] memory projectStartPresaleTime9 = new bytes32[](1);
334    projectStartPresaleTime9[0] = startPresale_;
335
335    bytes32[] memory projectEndPresaleTime9 = new bytes32[](1);
336    projectEndPresaleTime9[0] = endPresale_;
337
337    bytes32[] memory projectTokenAddress10 = new bytes32[](1);
338    projectTokenAddress10[0] = tokenAddress_;
339
339    bytes32[] memory projectCount10 = new bytes32[](1);
340    projectCount10[0] = count_;
341
341    bytes32[] memory projectSpv10 = new bytes32[](1);
342    projectSpv10[0] = msg.sender;
343
343    bytes32[] memory projectUnderwrite10 = new bytes32[](1);
344    projectUnderwrite10[0] = underwriteCount_;
345
345    bytes32[] memory projectId10 = new bytes32[](1);
346    projectId10[0] = id;
347
347    bytes32[] memory projectSymbol10 = new bytes32[](1);
348    projectSymbol10[0] = symbol_;
349
349    bytes32[] memory projectName10 = new bytes32[](1);
350    projectName10[0] = name_;
351
351    bytes32[] memory projectStartPresaleTime10 = new bytes32[](1);
352    projectStartPresaleTime10[0] = startPresale_;
353
353    bytes32[] memory projectEndPresaleTime10 = new bytes32[](1);
354    projectEndPresaleTime10[0] = endPresale_;
355
355    bytes32[] memory projectTokenAddress11 = new bytes32[](1);
356    projectTokenAddress11[0] = tokenAddress_;
357
357    bytes32[] memory projectCount11 = new bytes32[](1);
358    projectCount11[0] = count_;
359
359    bytes32[] memory projectSpv11 = new bytes32[](1);
360    projectSpv11[0] = msg.sender;
361
361    bytes32[] memory projectUnderwrite11 = new bytes32[](1);
362    projectUnderwrite11[0] = underwriteCount_;
363
363    bytes32[] memory projectId11 = new bytes32[](1);
364    projectId11[0] = id;
365
365    bytes32[] memory projectSymbol11 = new bytes32[](1);
366    projectSymbol11[0] = symbol_;
367
367    bytes32[] memory projectName11 = new bytes32[](1);
368    projectName11[0] = name_;
369
369    bytes32[] memory projectStartPresaleTime11 = new bytes32[](1);
370    projectStartPresaleTime11[0] = startPresale_;
371
371    bytes32[] memory projectEndPresaleTime11 = new bytes32[](1);
372    projectEndPresaleTime11[0] = endPresale_;
373
373    bytes32[] memory projectTokenAddress12 = new bytes32[](1);
374    projectTokenAddress12[0] = tokenAddress_;
375
375    bytes32[] memory projectCount12 = new bytes32[](1);
376    projectCount12[0] = count_;
377
377    bytes32[] memory projectSpv12 = new bytes32[](1);
378    projectSpv12[0] = msg.sender;
379
379    bytes32[] memory projectUnderwrite12 = new bytes32[](1);
380    projectUnderwrite12[0] = underwriteCount_;
381
381    bytes32[] memory projectId12 = new bytes32[](1);
382    projectId12[0] = id;
383
383    bytes32[] memory projectSymbol12 = new bytes32[](1);
384    projectSymbol12[0] = symbol_;
385
385    bytes32[] memory projectName12 = new bytes32[](1);
386    projectName12[0] = name_;
387
387    bytes32[] memory projectStartPresaleTime12 = new bytes32[](1);
388    projectStartPresaleTime12[0] = startPresale_;
389
389    bytes32[] memory projectEndPresaleTime12 = new bytes32[](1);
390    projectEndPresaleTime12[0] = endPresale_;
391
391    bytes32[] memory projectTokenAddress13 = new bytes32[](1);
392    projectTokenAddress13[0] = tokenAddress_;
393
393    bytes32[] memory projectCount13 = new bytes32[](1);
394    projectCount13[0] = count_;
395
395    bytes32[] memory projectSpv13 = new bytes32[](1);
396    projectSpv13[0] = msg.sender;
397
397    bytes32[] memory projectUnderwrite13 = new bytes32[](1);
398    projectUnderwrite13[0] = underwriteCount_;
399
399    bytes32[] memory projectId13 = new bytes32[](1);
400    projectId13[0] = id;
401
401    bytes32[] memory projectSymbol13 = new bytes32[](1);
402    projectSymbol13[0] = symbol_;
403
403    bytes32[] memory projectName13 = new bytes32[](1);
404    projectName13[0] = name_;
405
405    bytes32[] memory projectStartPresaleTime13 = new bytes32[](1);
406    projectStartPresaleTime13[0] = startPresale_;
407
407    bytes32[] memory projectEndPresaleTime13 = new bytes32[](1);
408    projectEndPresaleTime13[0] = endPresale_;
409
409    bytes32[] memory projectTokenAddress14 = new bytes32[](1);
410    projectTokenAddress14[0] = tokenAddress_;
411
411    bytes32[] memory projectCount14 = new bytes32[](1);
412    projectCount14[0] = count_;
413
413    bytes32[] memory projectSpv14 = new bytes32[](1);
414    projectSpv14[0] = msg.sender;
415
415    bytes32[] memory projectUnderwrite14 = new bytes32[](1);
416    projectUnderwrite14[0] = underwriteCount_;
417
417    bytes32[] memory projectId14 = new bytes32[](1);
418    projectId14[0] = id;
419
419    bytes32[] memory projectSymbol14 = new bytes32[](1);
420    projectSymbol14[0] = symbol_;
421
421    bytes32[] memory projectName14 = new bytes32[](1);
422    projectName14[0] = name_;
423
423    bytes32[] memory projectStartPresaleTime14 = new bytes32[](1);
424    projectStartPresaleTime14[0] = startPresale_;
425
425    bytes32[] memory projectEndPresaleTime14 = new bytes32[](1);
426    projectEndPresaleTime14[0] = endPresale_;
427
427    bytes32[] memory projectTokenAddress15 = new bytes32[](1);
428    projectTokenAddress15[0] = tokenAddress_;
429
429    bytes32[] memory projectCount15 = new bytes32[](1);
430    projectCount15[0] = count_;
431
431    bytes32[] memory projectSpv15 = new bytes32[](1);
432    projectSpv15[0] = msg.sender;
433
433    bytes32[] memory projectUnderwrite15 = new bytes32[](1);
434    projectUnderwrite15[0] = underwriteCount_;
435
435    bytes32[] memory projectId15 = new bytes32[](1);
436    projectId15[0] = id;
437
437    bytes32[] memory projectSymbol15 = new bytes32[](1);
438    projectSymbol15[0] = symbol_;
439
439    bytes32[] memory projectName15 = new bytes32[](1);
440    projectName15[0] = name_;
441
441    bytes32[] memory projectStartPresaleTime15 = new bytes32[](1);
442    projectStartPresaleTime15[0] = startPresale_;
443
443    bytes32[] memory projectEndPresaleTime15 = new bytes32[](1);
444    projectEndPresaleTime15[0] = endPresale_;
445
445    bytes32[] memory projectTokenAddress16 = new bytes32[](1);
446    projectTokenAddress16[0] = tokenAddress_;
447
447    bytes32[] memory projectCount16 = new bytes32[](1);
448    projectCount16[0] = count_;
449
449    bytes32[] memory projectSpv16 = new bytes32[](1);
450    projectSpv16[0] = msg.sender;
451
451    bytes32[] memory projectUnderwrite16 = new bytes32[](1);
452    projectUnderwrite16[0] = underwriteCount_;
453
453    bytes32[] memory projectId16 = new bytes32[](1);
454    projectId16[0] = id;
455
455    bytes32[] memory projectSymbol16 = new bytes32[](1);
456    projectSymbol16[0] = symbol_;
457
457    bytes32[] memory projectName16 = new bytes32[](1);
458    projectName16[0] = name_;
459
459    bytes32[] memory projectStartPresaleTime16 = new bytes32[](1);
460    projectStartPresaleTime16[0] = startPresale_;
461
461    bytes32[] memory projectEndPresaleTime16 = new bytes32[](1);
462    projectEndPresaleTime16[0] = endPresale_;
463
463    bytes32[] memory projectTokenAddress17 = new bytes32[](1);
464    projectTokenAddress17[0] = tokenAddress_;
465
465    bytes32[] memory projectCount17 = new bytes32[](1);
466    projectCount17[0] = count_;
467
467    bytes32[] memory projectSpv17 = new bytes32[](1);
468    projectSpv17[0] = msg.sender;
469
469    bytes32[] memory projectUnderwrite17 = new bytes32[](1);
470    projectUnderwrite17[0] = underwriteCount_;
471
471    bytes32[] memory projectId17 = new bytes32[](1);
472    projectId17[0] = id;
473
473    bytes32[] memory projectSymbol17 = new bytes32[](1);
474    projectSymbol17[0] = symbol_;
475
475    bytes32[] memory projectName17 = new bytes32[](1);
476    projectName17[0] = name_;
477
477    bytes32[] memory projectStartPresaleTime17 = new bytes32[](1);
478    projectStartPresaleTime17[0] = startPresale_;
479
479    bytes32[] memory projectEndPresaleTime17 = new bytes32[](1);
480    projectEndPresaleTime17[0] = endPresale_;
481
481    bytes32[] memory projectTokenAddress18 = new bytes32[](1);
482    projectTokenAddress18[0] = tokenAddress_;
483
483    bytes32[] memory projectCount18 = new bytes32[](1);
484    projectCount18[0] = count_;
485
485    bytes32[] memory projectSpv18 = new bytes32[](1);
486    projectSpv18[0] = msg.sender;
487
487    bytes32[] memory projectUnderwrite18 = new bytes32[](1);
488    projectUnderwrite18[0] = underwriteCount_;
489
489    bytes32[] memory projectId18 = new bytes32[](1);
490    projectId18[0] = id;
491
491    bytes32[] memory projectSymbol18 = new bytes32[](1);
492    projectSymbol18[0] = symbol_;
493
493    bytes32[] memory projectName18 = new bytes32[](1);
494    projectName18[0] = name_;
495
495    bytes32[] memory projectStartPresaleTime18 = new bytes32[](1);
496    projectStartPresaleTime18[0] = startPresale_;
497
497    bytes32[] memory projectEndPresaleTime18 = new bytes32[](1);
498    projectEndPresaleTime18[0] = endPresale_;
499
499    bytes32[] memory projectTokenAddress19 = new bytes32[](1);
500    projectTokenAddress19[0] = tokenAddress_;
501
501    bytes32[] memory projectCount19 = new bytes32[](1);
502    projectCount19[0] = count_;
503
503    bytes32[] memory projectSpv19 = new bytes32[](1);
504    projectSpv19[0] = msg.sender;
505
505    bytes32[] memory projectUnderwrite19 = new bytes32[](1);
506    projectUnderwrite19[0] = underwriteCount_;
507
507    bytes32[] memory projectId19 = new bytes32[](1);
508    projectId19[0] = id;
509
509    bytes32[] memory projectSymbol19 = new bytes32[](1);
510    projectSymbol19[0] = symbol_;
511
511    bytes32[] memory projectName19 = new bytes32[](1);
512    projectName19[0] = name_;
513
513    bytes32[] memory projectStartPresaleTime19 = new bytes32[](1);
514    projectStartPresaleTime19[0] = startPresale_;
515
515    bytes32[] memory projectEndPresaleTime19 = new bytes32[](1);
516    projectEndPresaleTime19[0] = endPresale_;
517
517    bytes32[] memory projectTokenAddress20 = new bytes32[](1);
518    projectTokenAddress20[0] = tokenAddress_;
519
519    bytes32[] memory projectCount20 = new bytes32[](1);
520    projectCount20[0] = count_;
521
521    bytes32[] memory projectSpv20 = new bytes32[](1);
522    projectSpv20[0] = msg.sender;
523
523    bytes32[] memory projectUnderwrite20 = new bytes32[](1);
524    projectUnderwrite20[0] = underwriteCount_;
525
525    bytes32[] memory projectId20 = new bytes32[](1);
526    projectId20[0] = id;
527
527    bytes32[] memory projectSymbol20 = new bytes32[](1);
528    projectSymbol20[0] = symbol_;
529
529    bytes32[] memory projectName20 = new bytes32[](1);
530    projectName20[0] = name_;
531
531    bytes32[] memory projectStartPresaleTime20 = new bytes32[](1);
532    projectStartPresaleTime20[0] = startPresale_;
533
533    bytes32[] memory projectEndPresaleTime20 = new bytes32[](1);
534    projectEndPresaleTime20[0] = endPresale_;
535
535    bytes32[] memory projectTokenAddress21 = new bytes32[](1);
536    projectTokenAddress21[0] = tokenAddress_;
537
537    bytes32[] memory projectCount21 = new bytes32[](1);
538    projectCount21[0] = count_;
539
539    bytes32[] memory projectSpv21 = new bytes32[](1);
540    projectSpv21[0] = msg.sender;
541
541    bytes32[] memory projectUnderwrite21 = new bytes32[](1);
542    projectUnderwrite21[0] = underwriteCount_;
543
543    bytes32[] memory projectId21 = new bytes32[](1);
544    projectId21[0] = id;
545
545    bytes32[] memory projectSymbol21 = new bytes32[](1);
546    projectSymbol21[0] = symbol_;
547
547    bytes32[] memory projectName21 = new bytes32[](1);
548    projectName21[0] = name_;
549
549    bytes32[] memory projectStartPresaleTime21 = new bytes32[](1);
550    projectStartPresaleTime21[0] = startPresale_;
551
551    bytes32[] memory projectEndPresaleTime21 = new bytes32[](1);
552    projectEndPresaleTime21[0] = endPresale_;
553
553    bytes32[] memory projectTokenAddress22 = new bytes32[](1);
554    projectTokenAddress22[0] = tokenAddress_;
555
555    bytes32[] memory projectCount22 = new bytes32[](1);
556    projectCount22[0] = count_;
557
557    bytes32[] memory projectSpv22 = new bytes32[](1);
558    projectSpv22[0] = msg.sender;
559
559    bytes32[] memory projectUnderwrite22 = new bytes32[](1);
560    projectUnderwrite22[0] = underwriteCount_;
561
561    bytes32[] memory projectId22 = new bytes32[](1);
562    projectId22[0] = id;
563
563    bytes32[] memory projectSymbol22 = new bytes32[](1);
564    projectSymbol22[0] = symbol_;
565
565    bytes32[] memory projectName22 = new bytes32[](1);
566    projectName22[0] = name_;
567
567    bytes32[] memory projectStartPresaleTime22 = new bytes32[](1);
568    projectStartPresaleTime22[0] = startPresale_;
569
569    bytes32[] memory projectEndPresaleTime22 = new bytes32[](1);
570    projectEndPresaleTime22[0] = endPresale_;
571
571    bytes32[] memory projectTokenAddress23 = new bytes32[](1);
572    projectTokenAddress23[0] = tokenAddress_;
573
573    bytes32[] memory projectCount23 = new bytes32[](1);
574    projectCount23[0] = count_;
575
575    bytes32[] memory projectSpv23 = new bytes32[](1);
576    projectSpv23[0] = msg.sender;
577
577    bytes32[] memory projectUnderwrite23 = new bytes32[](1);
578    projectUnderwrite23[0] = underwriteCount_;
579
579    bytes32[] memory projectId23 = new bytes32[](1);
580    projectId23[0] = id;
581
581    bytes32[] memory projectSymbol23 = new bytes32[](1);
582    projectSymbol23[0] = symbol_;
583
583    bytes32[] memory projectName23 = new bytes32[](1);
584    projectName23[0] = name_;
585
585    bytes32[] memory projectStartPresaleTime23 = new bytes32[](1);
586    projectStartPresaleTime23[0] = startPresale_;
587
587    bytes32[] memory projectEndPresaleTime23 = new bytes32[](1);
588    projectEndPresaleTime23[0] = endPresale_;
589
589    bytes32[] memory projectTokenAddress24 = new bytes32[](1);
590    projectTokenAddress24[0] = tokenAddress_;
591
591    bytes32[] memory projectCount24 = new bytes32[](1);
592    projectCount24[0] = count_;
593
593    bytes32[] memory projectSpv24 = new bytes32[](1);
594    projectSpv24[0] = msg.sender;
595
595    bytes32[] memory projectUnderwrite24 = new bytes32[](1);
596    projectUnderwrite24[0] = underwriteCount_;
597
597    bytes32[] memory projectId24 = new bytes32[](1);
598    projectId24[0] = id;
599
599    bytes32[] memory projectSymbol24 = new bytes32[](1);
600    projectSymbol24[0] = symbol_;
601
601    bytes32[] memory projectName24 = new bytes32[](1);
602    projectName24[0] = name_;
603
603    bytes32[] memory projectStartPresaleTime24 = new bytes32[](1);
604    projectStartPresaleTime24[0] = startPresale_;
605
605    bytes32[] memory projectEndPresaleTime24 = new bytes32[](1);
606    projectEndPresaleTime24[0] = endPresale_;
607
607    bytes32[] memory projectTokenAddress25 = new bytes32[](1);
608    projectTokenAddress25[0] = tokenAddress_;
609
609    bytes32[] memory projectCount25 = new bytes32[](1);
610    projectCount25[0] = count_;
611
611    bytes32[] memory projectSpv25 = new bytes32[](1);
612    projectSpv25[0] = msg.sender;
613
613    bytes32[] memory projectUnderwrite25 = new bytes32[](1);
614    projectUnderwrite25[0] = underwriteCount_;
615
615    bytes32[] memory projectId25 = new bytes32[](1);
616    projectId25[0] = id;
617
617    bytes32[] memory projectSymbol25 = new bytes32[](1);
618    projectSymbol25[0] = symbol_;
619
619    bytes32[] memory projectName25 = new bytes32[](1);
620    projectName25[0] = name_;
621
621    bytes32[] memory projectStartPresaleTime25 = new bytes32[](1);
622    projectStartPresaleTime25[0] = startPresale_;
623
623    bytes32[] memory projectEndPresaleTime25 = new bytes32[](1);
624    projectEndPresaleTime25[0] = endPresale_;
625
625    bytes32[] memory projectTokenAddress26 = new bytes32[](1);
626    projectTokenAddress26[0] = tokenAddress_;
627
627    bytes32[] memory projectCount26 = new bytes32[](1);
628    projectCount26[0] = count_;
629
629    bytes32[] memory projectSpv26 = new bytes32[](1);
630    projectSpv26[0] = msg.sender;
631
631    bytes32[] memory projectUnderwrite26 = new bytes32[](1);
632    projectUnderwrite26[0] = underwriteCount_;
633
633    bytes32[] memory projectId26 = new bytes32[](1);
634    projectId26[0] = id;
635
635    bytes32[] memory projectSymbol26 = new bytes32[](1);
636    projectSymbol26[0] = symbol_;
637
637    bytes32[] memory projectName26 = new bytes32[](1);
638    projectName26[0] = name_;
639
639    bytes32[] memory projectStartPresaleTime26 = new bytes32[](1);
640    projectStartPresaleTime26[0] = startPresale_;
641
641    bytes32[] memory projectEndPresaleTime26 = new bytes32[](1);
642    projectEndPresaleTime26[0] = endPresale_;
643
643    bytes32[] memory projectTokenAddress27 = new bytes32[](1);
644    projectTokenAddress27[0] = tokenAddress_;
645
645    bytes32[] memory projectCount27 = new bytes32[](1);
646    projectCount27[0] = count_;
647
647    bytes32[] memory projectSpv27 = new bytes32[](1);
648    projectSpv27[0
```

```

94     trustee_: msg.sender,
95     count_: count_,
96     underwriteCount_: underwriteCount_,
97     tokenAddress_: tokenAddress_,
98     membershipAddress_: _membershipAddress,
99     name_: name_,
100    symbol_: symbol_,
101    routerAddress: address(this)
102  });
103  address projectAddress = _xtatuzFactory.createProjectContract(data);
104  IXtatuzProject(projectAddress).setPresalePeriod(startPresale_, endPresale_);
105  emit CreatedProject(projectId, projectAddress);
106 }

```

Listing 20.1 The `createProject` function
that invokes the `setPresalePeriod` one time

XtatuzProject.sol

```

218 function extendEndPresale() public onlyOperator {
219     require(block.timestamp > (endPresale - 1 days), "PROJECT: NOT_END_PREV");
220     require(_isTriggeredEndpresale == false, "PROJECT: EXTENDED_PRESALE");
221     if (_isTriggeredEndpresale == false) {
222         _extendEndPresale();
223     }
224 }

// (...SNIPPED...)

273 function _extendEndPresale() internal {
274     require(_isTriggeredEndpresale == false, "PROJECT: EXTENDED_PRESALE");
275     if (block.timestamp > (endPresale - 1 days)) {
276         uint256 absoluteCount = count - _underwriteCount;
277         uint256 percent = ((count - countReserve) * 100) / absoluteCount;
278         if (percent >= 95) {
279             endPresale += 5 days;
280         } else if (percent >= 85 && percent < 95) {
281             endPresale += 10 days;
282         } else if (percent >= 65 && percent < 85) {
283             endPresale += 15 days;
284         } else {
285             endPresale += 30 days;
286         }
287         _isTriggeredEndpresale = true;
288     }
289 }

```

Listing 20.2 The `extendEndPresale` function that
allows the operator to extend presale by condition

XstatuzProject.sol

```

105   function setPresalePeriod(uint256 startPresale_, uint256 endPresale_) public
106     onlyOperator {
107       require(endPresale_ > startPresale_, "PROJECT: WRONG_END_DATE");
108       startPresale = startPresale_;
109     }
  
```

Listing 20.3 The *setPresalePeriod* function that allows the operator to set arbitrary presale period

Recommendations

We recommend **changing the *setPresalePeriod* function visibility to *internal* and restrict access to the only owner** (L105 in code snippet 20.4). Then, adding the *startPresale_* and *endPresale_* constructor parameters (L51 - 52 in code snippet 20.5) and invoking the *setPresalePeriod* function one time at an *XstatuzProject* contract constructor (L58 in code snippet 20.5).

However, we noticed there is a use of *Factory Pattern* to create a contract, the related code should be adjusted accordingly.

XstatuzProject.sol

```

105   function setPresalePeriod(uint256 startPresale_, uint256 endPresale_) internal
106     onlyOwner {
107       require(endPresale_ > startPresale_, "PROJECT: WRONG_END_DATE");
108       startPresale = startPresale_;
109     }
  
```

Listing 20.4 The improved *setPresalePeriod* function

XstatuzProject.sol

```

42   constructor(
43     uint256 projectId_,
44     address operator_,
45     address trustee_,
46     uint256 count_,
47     uint256 underwriteCount_,
48     address tokenAddress_,
49     address propertyAddress_,
50     address presaledAddress_,
51     uint256 startPresale_,
52     uint256 endPresale_
53   ) {
  
```

```
54     _transferOperator(operator_);
55     _transferTrustee(trustee_);
56     _initialData(count_, underwriteCount_, tokenAddress_, propertyAddress_,
57 presaledAddress_);
58     setPresalePeriod(startPresale_, endPresale_);
59     projectId = projectId_;
60     _projectOwner = tx.origin;
61 }
```

Listing 20.5 The improved *XtatuzProject* contract constructor

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

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 21	Possibly Setting Inconsistent Token URI		
Risk	Medium	Likelihood	Medium
Functionality is in use	In use	Status	Acknowledged
Associated Files	<i>contracts/XtatuzRouter.sol</i> <i>contracts/Property.sol</i>		
Locations	<i>XtatuzRouter.sol L: 166 - 189</i> <i>Property.sol L: 110 - 122</i>		

Detailed Issue

The *nftReroll* function of the *XtatuzRouter* contract (L166 - 189 in code snippet 21.2) allows the token owner to reroll their property token URI.

By default, if it is not set to the specific token URI, the returned *tokenURI* function of the *Property* contract will start with base URI and end with token ID (L119 in code snippet 21.1).

For example, the returned **Token ID 1 URI** is *ipfs://baseURI/1*.

Property.sol	
110	function <i>tokenURI</i> (uint256 tokenId_) public view virtual override returns (string memory) {
111	_requireMinted(tokenId_);
112	
113	string memory _tokenURI = _tokenURIs[tokenId_];
114	string memory base = _baseURI();
115	
116	if (bytes(_tokenURI).length > 0) {
117	return _tokenURI;
118	} else {
119	return string(abi.encodePacked(base, tokenId_.toString()));
120	}
121	}

Listing 21.1 The *tokenURI* function that returns the ending URI with the token ID by default

However, we noticed that the reroll mechanism makes it possible to set incorrect token URI, which confuses the user. By following the crucial steps to set a new URI.

1. the *getRerollData* function will be invoked to get the reroll data (L176 in code snippet 21.2).
2. A URI will be randomly selected and used as a new URI (L180 - 181 in code snippet 21.2).
3. The previous URI will be set back to the reroll data with the same index that enables the next reroll to have a chance to get the same one (L182 in code snippet 21.2).

XstatuzRouter.sol

```

166  function nftReroll(uint256 projectId_, uint256 tokenId_) public {
167      require(_rerollAddress != address(0), "ROUTER: NO_REROLL_ADDRESS");
168
169      address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
170      IProperty property = IProperty(propertyAddress);
171      IXtatuzReroll rerollContract = IXtatuzReroll(_rerollAddress);
172
173      address tokenAddress = rerollContract.tokenAddress();
174      string memory prevUri = property.tokenURI(tokenId_);
175      uint256 fee = rerollContract.rerollFee();
176      string[] memory rerollData = rerollContract.getRerollData(projectId_);
177      address tokenOwner = property.ownerOf(tokenId_);
178      require(tokenOwner == msg.sender, "ROUTER: NOT_NFT_OWNER");
179
180      uint256 newIndex = block.timestamp % rerollData.length;
181      property.setTokenURI(tokenId_, rerollData[newIndex]);
182      rerollData[newIndex] = prevUri;
183      rerollContract.setRerollData(projectId_, rerollData);
184
185      IERC20(tokenAddress).transferFrom(msg.sender, address(this), fee);
186      _totalRerollFee[projectId_] += fee;
187
188      emit NFTReroll(msg.sender, projectId_, tokenId_);
189  }

```

Listing 21.2 The *nftReroll* function of the *XstatuzRouter* contract

To explain this issue, please consider the scenario below.

Assume that the *rerollData* array is:

[0]: *ipfs://baseURI/101*

[1]: *ipfs://baseURI/102*

[2]: *ipfs://baseURI/103*

For Example:

1. The **Token ID 1** owner calls the *nftReroll* function to reroll their token.

In this step, from the start the **Token ID 1 URI** is *ipfs://baseURI/1*

Finally, assume the result of the new **Token ID 1 URI** is *ipfs://baseURI/101* from *rerollData[0]*

And then, set the previous URI *ipfs://baseURI/1* back to *rerollData[0]*:

[0]: *ipfs://baseURI/1*

[1]: *ipfs://baseURI/102*

[2]: *ipfs://baseURI/103*

2. The **Token ID 2** owner calls the *nftReroll* function to reroll their token.

In this step, from the start the **Token ID 2 URI** is *ipfs://baseURI/2*

Finally, assume the result of the new **Token ID 2 URI** is *ipfs://baseURI/1* from *rerollData[0]*

And then, set the previous uri *ipfs://baseURI/2* back to *rerollData[0]*:

[0]: *ipfs://baseURI/2*

[1]: *ipfs://baseURI/102*

[2]: *ipfs://baseURI/103*

As shown in the scenario above, the inconsistent token URI with the token ID could cause confusion for the user.

Example of inconsistent URI

Token ID 1 URI is *ipfs://baseURI/101*

Token ID 2 URI is *ipfs://baseURI/1*

Recommendations

We recommend redesigning and reimplementing the reroll mechanism or changing the token URI not to have a meaning that conveys the token ID to prevent confusing the user.

Reassessment

The Xtatuz team acknowledged this issue.

No. 22	Lack Of Time Delay In Pullback Process		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	<code>contract/XtatuzRouter.sol</code>		
Locations	<code>XtatuzRouter.sol L: 277 - 281, L: 283 - 286, L: 288 - 299</code>		

Detailed Issue

The `pullbackInactive` function (code snippet 22.2) allows SPV account to pull back property tokens from an inactive wallet. In the pullback process, the SPV account has to notice the inactive wallet by invoking the `noticeToInactiveWallet` function (L283 - 287 in code snippet 22.1) for notice to the inactive wallet user before pulling back the tokens.

However, we found that **there is no time delay mechanism in this process, which means the SPV account can pullback tokens from an inactive wallet immediately after invoking the `noticeToInactiveWallet` function.**

More specifically, the `_isNotice[projectId_][inactiveWallet_]` will be set to `true` when the SPV account invokes the `noticeToInactiveWallet` function, which means that the `require` statement in `pullbackInactive` function (L289 in code snippet 22.2) will pass and allows SPV account to pull back property tokens from an inactive wallet.

Consequently, the absence of a time delay mechanism during the pullback can result in an unfair or confusing experience for the owner of the inactive wallet.

XstatuzRouter.sol

```

277  function noticeReply(uint256 projectId_) public {
278      address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
279      require(projectAddress != address(0), "ROUTER: INVALID_PROJECT_ID");
280      _isNotice[projectId_][msg.sender] = false;
281  }
282
283  function noticeToInactiveWallet(uint256 projectId_, address inactiveWallet_)
public onlySpv {
284      _isNotice[projectId_][inactiveWallet_] = true;
285      emit NoticeToInactiveWallet(projectId_, inactiveWallet_);
286  }

```

Listing 22.1 The *noticeReply* and *noticeToInactiveWallet* functions
XstatuzRouter.sol

```

288  function pullbackInactive(uint256 projectId_, address inactiveWallet_) public
onlySpv {
289      require(_isNotice[projectId_][inactiveWallet_] == true, "ROUTER:
NOTICE_BEFORE");
290      address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
291      IProperty property = IProperty(propertyAddress);
292      uint256[] memory nftList = property.getTokenIdList(inactiveWallet_);
293
294      for(uint i = 0; i < nftList.length; i++){
295          IERC721(propertyAddress).safeTransferFrom(inactiveWallet_, msg.sender,
nftList[i]);
296      }
297
298      emit PullbackInactive(projectId_, inactiveWallet_);
299  }

```

Listing 22.2 The *pullbackInactive* function

Recommendations

We recommend implementing a time delay mechanism and improving event emission for the pullback process to prevent potential confusion or unfairness to the inactive wallet owner. This time delay will allow the inactive wallet owner sufficient time to reactivate their wallet and prevent the tokens from being pulled back.

XstatuzRouter.sol

```

// NEW STATES
25 uint256 public constant PULLBACK_PERIOD = 30 days;
26 mapping(uint256 => mapping(address => uint256)) public isNoticeTimestamp;

// NEW EVENTS
63 event NoticeReply(uint256 indexed projectId, address indexed inactiveWallet_,
64 uint256 noticeTimestamp);
event NoticeToInactiveWallet(uint256 indexed projectId, address indexed
  inactiveWallet_, uint256 noticeTimestamp);

```

Listing 22.3 The improved *states* and *events***XstatuzRouter.sol**

```

277 function noticeReply(uint256 projectId_) public {
278   address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
279   require(projectAddress != address(0), "ROUTER: INVALID_PROJECT_ID");
280   _isNotice[projectId_][msg.sender] = false;
281   isNoticeTimestamp[projectId_][msg.sender] = block.timestamp;
282   emit NoticeReply(projectId_, msg.sender, block.timestamp);
283 }
284
285 function noticeToInactiveWallet(uint256 projectId_, address inactiveWallet_)
286 public onlySpv {
287   _isNotice[projectId_][inactiveWallet_] = true;
288   isNoticeTimestamp[projectId_][inactiveWallet_] = block.timestamp +
289   PULLBACK_PERIOD;
290   emit NoticeToInactiveWallet(projectId_, inactiveWallet_, block.timestamp);
291 }

```

Listing 22.4 The improved *noticeReply* and *noticeToInactiveWallet* functions**XstatuzRouter.sol**

```

288 function pullbackInactive(uint256 projectId_, address inactiveWallet_) public
onlySpv {
289   require(_isNotice[projectId_][inactiveWallet_] == true, "ROUTER:
NOTICE_BEFORE");
290   require(isNoticeTimestamp[projectId_][inactiveWallet_] < block.timestamp);
291   address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
292   IProperty property = IProperty(propertyAddress);
293   uint256[] memory nftList = property.getTokenIdList(inactiveWallet_);
294
295   for(uint i = 0; i < nftList.length; i++){
296     IERC721(propertyAddress).safeTransferFrom(inactiveWallet_, msg.sender,
nftList[i]);
297   }

```

```
298
299     emit PullbackInactive(projectId_, inactiveWallet_);
300 }
```

Listing 22.5 The improved *pullbackInactive* function

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 23	Potential Denial-Of-Service On Getting Member Projects		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	<code>contract/XtatuzRouter.sol</code>		
Locations	<code>XtatuzRouter.sol L: 108 - 138, L: 151 - 164, L: 215 - 247</code>		

Detailed Issue

The `_memberdProject` state is a mapping that associates a particular member with a list of projects in which they participate (L38 in code snippet 23.1). This state is updated when a user invokes the `addProjectMember` function, which will first check whether the user is not already a participant, and then adds the `projectId_` parameter to the `_memberdProject` state to keep track of the projects in which the user participates (L133 in code snippet 23.1).

We discovered that the `_memberdProject` state is also utilized in the `refund` and `getAllCollection` functions as part of their respective processes. Specifically, in the `refund` function, the `_memberdProject` state is used to ensure that the member is eligible to receive a refund (L152 in code snippet 23.2), while in the `getAllCollection` function, it is used to retrieve a list of projects in which a particular member is involved (L216 in code snippet 23.3).

However, if the project list retrieved from `_memberdProject` is too large, the `refund` and `getAllCollection` functions may fail due to exceeding the block gas limit. This can cause the transaction to be reverted, leading to the denial-of-service issue.

XtatuzRouter.sol

```

38  mapping(address => uint256[]) private _memberdProject;

        // (...SNIPPED...)

108 function addProjectMember(
109     uint256 projectId_,
110     uint256[] memory nftList_,
111     string memory referral_
112 ) public {
113     uint256 amount = nftList_.length;
114     address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
115     IXTatuzProject project = IXTatuzProject(projectAddress);

```

```

116     uint256 price = project.addProjectMember(msg.sender, nftList_);
117
118     uint256 minPrice = project.minPrice();
119     IXstatuzReferral referralContract = IXstatuzReferral(_referralAddress);
120     referralContract.increaseBuyerRef(projectId_, referral_, amount * minPrice);
121
122     address tokenAddress = project.tokenAddress();
123     IERC20(tokenAddress).transferFrom(msg.sender, projectAddress, price);
124
125     uint256[] memory memberedProject = _memberdProject[msg.sender];
126     bool foundedIndex;
127     for (uint256 index = 0; index < memberedProject.length; index++) {
128         if (memberedProject[index] == projectId_) {
129             foundedIndex = true;
130         }
131     }
132     if (!foundedIndex) {
133         _memberdProject[msg.sender].push(projectId_);
134     }
135
136     _isMemberClaimed[msg.sender][projectId_] = false;
137     emit AddProjectMember(projectId_, msg.sender, referral_, price);
138 }
```

Listing 23.1 The *addProjectMember* function**XstatuzRouter.sol**

```

151 function refund(uint256 projectId_) public {
152     uint256[] memory memberedProject = _memberdProject[msg.sender];
153     address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
154
155     for (uint256 index = 0; index < memberedProject.length; index++) {
156         if (memberedProject[index] == projectId_) {
157             delete _memberdProject[msg.sender][index];
158         }
159     }
160
161     IXstatuzProject(projectAddress).refund(msg.sender);
162
163     emit Refunded(projectId_, msg.sender);
164 }
```

Listing 23.2 The *refund* function

XtatuzRouter.sol

```

215  function getAllCollection() public view returns (Collection[] memory) {
216      uint256[] memory projectList = _memberdProject[msg.sender];
217      Collection[] memory collections = new Collection[](projectList.length);
218      for (uint256 index = 0; index < projectList.length; index++) {
219          if (projectList[index] > 0) {
220              uint256 projectId = projectList[index];
221              address projectAddress =
222                  _xtatuzFactory.getProjectAddress(projectId);
223              IXtatuzProject.Status status =
224                  IXtatuzProject(projectAddress).projectStatus();
225              if (status == IXtatuzProject.Status.FINISH &&
226                  _isMemberClaimed[msg.sender][projectId]) {
227                  address propertyAddress =
228                      _xtatuzFactory.getPropertyAddress(projectId);
229                  uint256[] memory tokenList =
230                      IProperty(propertyAddress).getTokenIdList(msg.sender);
231                  IProperty.PropertyStatus propStatus =
232                      IProperty(propertyAddress).propertyStatus();
233                  CollectionType collecType = CollectionType(uint256(propStatus) +
234 1);
235                  Collection memory collect = Collection({
236                      contractAddress: propertyAddress,
237                      tokenIdList: tokenList,
238                      collectionType: collecType
239                  });
240                  collections[index] = collect;
241              } else {
242                  address presaledAddress =
243                      _xtatuzFactory.getPresaledAddress(projectId);
244                  uint256[] memory tokenList =
245                      IPresaled(presaledAddress).getPresaledOwner(msg.sender);
246                  Collection memory collect = Collection({
247                      contractAddress: presaledAddress,
248                      tokenIdList: tokenList,
249                      collectionType: CollectionType.PRESALE
250                  });
251                  collections[index] = collect;
252              }
253          }
254      }
255      return collections;
256  }

```

Listing 23.3 The *getAllCollection* function

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features, we recommend redesigning and reimplementing the mechanism for obtaining a list of project IDs from a member, which will also impact the related mechanisms.

Reassessment

This issue was acknowledged by the *Xtatuz* team.

No. 24	Potential Denial-Of-Service On Get All Collections		
Risk	Medium	Likelihood	Medium
Functionality is in use	In use	Impact	Medium
Associated Files	<code>contracts/XtatuzRouter.sol</code> <code>contracts/Property.sol</code> <code>contracts/Presaled.sol</code>		
Locations	<code>XtatuzRouter.sol L: 215 - 247</code> <code>Property.sol L: 101 - 108</code> <code>Presaled.sol L: 64 - 71</code>		

Detailed Issue

The `getAllCollection` function of the `XtatuzRouter` contract allows the member to get all tokens from all their participating projects.

We notice that in the `getAllCollection` function (L215 - 247 in code snippet 24.1), there are the for-loops that iterate through the `projectList` array (L218 - 245 in code snippet 24.1). The loop would iterate over all array elements to look for the tokens through the `getTokenIdList` and `getPresaledOwner` functions (L225 and 236 in code snippet 24.1).

Additionally, inside the `getTokenIdList` and `getPresaledOwner` functions, there are the *for-loops* that iterate through the token balances to look up at the token by index (L103 - 106 in code snippet 24.2 and 65 - 69 in code snippet 24.3).

These processes can consume a lot of gas and the gas used is prone to exceeding the block gas limit if the number of associated arrays are too large. In the case of exceeding the block gas limit, a transaction would be reverted, leading to the denial-of-service issue to the affected functions.

XtatuzRouter.sol

```

215  function getAllCollection() public view returns (Collection[] memory) {
216      uint256[] memory projectList = _memberdProject[msg.sender];
217      Collection[] memory collections = new Collection[](projectList.length);
218      for (uint256 index = 0; index < projectList.length; index++) {
219          if (projectList[index] > 0) {
220              uint256 projectId = projectList[index];
221              address projectAddress =
222                  _xtatuzFactory.getProjectAddress(projectId);
223              IXTatuzProject.Status status =

```

```

223   IXtatuzProject(projectAddress).projectStatus();
224     if (status == IXtatuzProject.Status.FINISH &&
225       _isMemberClaimed[msg.sender][projectId]) {
226       address propertyAddress =
227       _xtatuzFactory.getPropertyAddress(projectId);
228       uint256[] memory tokenList =
229       IProperty(propertyAddress).getTokenIdList(msg.sender);
230       IProperty.PropertyStatus propStatus =
231       IProperty(propertyAddress).propertyStatus();
232       CollectionType collectType = CollectionType(CollectionType(uint256(propStatus) +
233         1));
234       Collection memory collect = Collection({
235         contractAddress: propertyAddress,
236         tokenIdList: tokenList,
237         collectionType: collectType
238       });
239       collections[index] = collect;
240     } else {
241       address presaledAddress =
242       _xtatuzFactory.getPresaledAddress(projectId);
243       uint256[] memory tokenList =
244       IPresaled(presaledAddress).getPresaledOwner(msg.sender);
245       Collection memory collect = Collection({
246         contractAddress: presaledAddress,
247         tokenIdList: tokenList,
248         collectionType: CollectionType.PRESALE
249       });
250       collections[index] = collect;
251     }
252   }
253   return collections;
254 }
```

Listing 24.1 The `getAllCollection` function that iterates all projects, which can consume more gas than the block gas limit

Property.sol

```

101 function getTokenIdList(address member) public view returns (uint256[] memory) {
102     uint256 balance = balanceOf(member);
103     uint256[] memory tokenList = new uint256[](balance);
104     for (uint256 index = 0; index < balance; index++) {
105         tokenList[index] = tokenOfOwnerByIndex(member, index);
106     }
107     return tokenList;
108 }
```

Listing 24.2 The `getTokenIdList` function that iterates all token balances by index, which can consume more gas than the block gas limit

Presaled.sol

```

64 function getPresaledOwner(address owner) public view returns (uint256[] memory)
{
65     uint256 balance = balanceOf(owner);
66     uint256[] memory tokenList = new uint256[](balance);
67     for (uint256 index = 0; index < balance; index++) {
68         tokenList[index] = tokenOfOwnerByIndex(owner, index);
69     }
70     return tokenList;
71 }
```

Listing 24.2 The `getPresaledOwner` function that iterates all token balances by index, which can consume more gas than the block gas limit

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features.

One possible mitigating solution for this, we recommend adding a new function that looks up a specific project and adding more related functions that apply the pagination concept.

We provide recommended code to address this issue as a suggested remediation concept only. We recommend that the Xtatuz team should adjust the recommendation code according to the business design.

Property.sol

```

4  interface IProperty {
    // (...SNIPPED...)
46     function getTokenIdListFrom(uint256 page, uint256 contentsPerPage, address
member) external view returns (uint256[] memory);
47 }
```

Listing 24.4 The new *getTokenIdListFrom* function that apply the pagination concept

Property.sol

```

5  interface IPresaled {
    // (...SNIPPED...)
25     function getPresaledOwnerFrom(uint256 page, uint256 contentsPerPage, address
owner_) external view returns (uint256[] memory);
26 }
```

Listing 24.5 The new *getPresaledOwnerFrom* function that apply the pagination concept

XtatuzRouter.sol

```

300 function.getProjectCollectionFrom(uint256 page, uint256 contentsPerPage, uint256
projectId) public view returns (Collection memory) {
301     address projectAddress = _xtatuzFactory.getProjectAddress(projectId);
302     IXtatuzProject.Status status =
IXtatuzProject(projectAddress).projectStatus();
303     if (status == IXtatuzProject.Status.FINISH &&
_isMemberClaimed[msg.sender][projectId]) {
304         address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId);
305         uint256[] memory tokenList =
IProperty(propertyAddress).getTokenIdListFrom(page, contentsPerPage,
msg.sender);
306         IProperty.PropertyStatus propStatus =
IProperty(propertyAddress).propertyStatus();
307         CollectionType collectType = CollectionType(uint256(propStatus) + 1);
308         return Collection({
309             contractAddress: propertyAddress,
310             tokenIdList: tokenList,
311             collectionType: collectType
312         });
313     } else {
314         address presaledAddress = _xtatuzFactory.getPresaledAddress(projectId);
```

```

315         uint256[] memory tokenList =
316             IPresaled(presaledAddress).getPresaledOwnerFrom(page, contentsPerPage,
317             msg.sender);
317         return Collection({
318             contractAddress: presaledAddress,
319             tokenIdList: tokenList,
320             collectionType: CollectionType.PRESALE
321         });
321     }
322 }
```

Listing 24.6 The *getProjectCollectionFrom* function
that look up a specific project

Property.sol

```

146 function getTokenIdListFrom(uint256 page, uint256 contentsPerPage, address
147 member) external view returns (uint256[] memory) {
148     require(page > 0, "page start from 1");
149     uint256 balance = balanceOf(member);
150     page -= 1;
151     uint256 from = page * contentsPerPage;
152     uint256 to = from + contentsPerPage;
153     uint256 contentLength = balance;
154
155     if (to >= contentLength && (to - contentLength) > contentsPerPage) {
156         return new uint256[](0);
157     }
158
159     contentsPerPage = to < contentLength ? contentsPerPage : contentLength % contentsPerPage;
160
161     uint256[] memory contents = new uint256[](contentsPerPage);
162     for (uint256 index = 0; index < contentsPerPage; index++) {
163         contents[index] = tokenOfOwnerByIndex(member, from + index);
164     }
165
166     return contents;
166 }
```

Listing 24.7 The *getTokenIdListFrom* function implementation that
apply the pagination concept

Presaled.sol

```

100 function getPresaledOwnerFrom(uint256 page, uint256 contentsPerPage, address
101   owner_) external view returns (uint256[] memory) {
102     require(page > 0, "page start from 1");
103     uint256 balance = balanceOf(owner_);
104     page -= 1;
105     uint256 from = page * contentsPerPage;
106     uint256 to = from + contentsPerPage;
107     uint256 contentLength = balance;
108
109     if (to >= contentLength && (to - contentLength) > contentsPerPage) {
110       return new uint256[](0);
111     }
112
113     contentsPerPage = to < contentLength ? contentsPerPage : contentLength % contentsPerPage;
114
115     uint256[] memory contents = new uint256[](contentsPerPage);
116     for (uint256 index = 0; index < contentsPerPage; index++) {
117       contents[index] = tokenOfOwnerByIndex(owner_, from + index);
118     }
119
120     return contents;
  }
```

Listing 24.8 The `getPresaledOwnerFrom` function implementation that apply the pagination concept

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

Reassessment

The Xtatuz team decided to remove the `getAllCollection` function. Hence, this issue was fixed.

No. 25	Permanent Lock Of Ether Funds In Contracts		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	contracts/PresaledFactory.sol contracts/PropertyFactory.sol contracts/ProjectFactory.sol contracts/XtatuzFactory.sol		
Locations	<i>PresaledFactory.sol</i> L: 9 - 20 <i>PropertyFactory.sol</i> L: 9 - 20 <i>ProjectFactory.sol</i> L: 9 - 27 <i>XtatuzFactory.sol</i> L: 47 - 102		

Detailed Issue

We discovered that the following functions have been declared as payable functions:

- The *createPresale* function of the *PresaledFactory* contract
- The *createProperty* function of the *PropertyFactory* contract
- The *createProject* function of the *ProjectFactory* contract
- The *createProjectContract* function of the *XtatuzFactory* contract

A payable function is a function that is capable of receiving Ether. This feature allows the developer to keep a record of deposits and implement any additional necessary logic.

However, we have observed that there is no withdrawal function for withdrawing the Ether in the mentioned contracts. As a result, if the contract owner mistakenly invokes any of these functions along with an Ether deposit, they will permanently lose those funds.

ProjectFactory.sol

```

9  function createProperty(
10    string memory _name,
11    string memory _symbol,
12    bytes32 _salt,
13    address operator_,
14    address routerAddress_,
15    uint256 count_
16 ) public payable onlyOwner returns (address) {
17   address propertyAddress = address(new Property{salt: _salt}(_name, _symbol,

```

```
operator_, routerAddress_, count_));  
18     Property(propertyAddress).transferOwnership(msg.sender);  
19     return propertyAddress;  
20 }
```

Listing 25.1 The *createProperty* function of the *ProjectFactory* contract,
one of the payable functions that cause the issue

Recommendations

We recommend **removing the payable modifier** from the associated functions. This is due to the lack of provision for the withdrawal of Ether in the contracts, leading to a potentially permanent loss of funds for the contract owner in the event of mistakenly invoking a payable function with an Ether deposit.

Reassessment

The Xtatuz team fixed this issue according to our suggestion.

No. 26	Double Registering Project Member		
Risk	Medium	Likelihood	High
Functionality is in use	In use	Impact	Low
Associated Files	<i>contract/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 120 - 148</i>		

Detailed Issue

We found no validation of existing project members in the *addProjectMember* function. This issue will have an impact on the *projectMember* state, which is utilized within the system.

Without checking the duplicated project members, the size of the *projectMember* state may increase, leading to potential functional impairments or other issues within the system.

XtatuzProject.sol

```

120  function addProjectMember(address member_, uint256[] memory nftList_)
121    public
122    isAvailable
123    isLeftReserve
124    whenNotPaused
125    onlyOwner
126    returns (uint256)
127  {
128    // (...SNIPPED...)
129
130    projectMember.push(member_);
131
132    IPresaled(_presaledAddress).mint(member_, nftList_);
133    emit AddProjectMember(projectId, member_, price);
134
135    return price;
136  }

```

Listing 26.1 The *addProjectMember* function

Recommendations

We recommend implementing a mapping variable to track the existence of members, which would help to validate and prevent the duplication of members before adding a project member to the *projectMember* state.

XtatuzProject.sol

```

36 address[] public projectMember;
37 mapping(address => bool) public memberExists;

// (...SNIPPED...)

120 function addProjectMember(address member_, uint256[] memory nftList_)
121   public
122     isAvailable
123     isLeftReserve
124     whenNotPaused
125     onlyOwner
126     returns (uint256)
127 {
128   uint256 amount = nftList_.length;
129   require(amount > 0, "PROJECT: ZERO_AMOUNT");
130   for(uint i = 0; i < amount; ++i){
131     require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
132   }
133   _checkAvailableNFT(nftList_);
134
135   uint256 price = minPrice * amount;
136   projectValue += price;
137
138   _pickupAvailableNFT(nftList_, member_);
139
140   countReserve -= amount;
141   if (memberExists[member_] == false) {
142     memberExists[member_] = true;
143     projectMember.push(member_);
144   }
145
146   IPresaled(_presaledAddress).mint(member_, nftList_);
147   emit AddProjectMember(projectId, member_, price);
148
149   return price;
150 }
```

Listing 26.2 Improved XtatuzProject contract

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 27	Usage Of Unsafe Token Transfer Functions		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/XtatuzRouter.sol</i> <i>contracts/XtatuzReferral.sol</i> <i>contracts/XtatuzProject.sol</i>		
Locations	<i>XtatuzRouter.sol</i> <ul style="list-style-type: none"> • L123 (<i>The addProjectMember function</i>), • L185 (<i>The nftReroll function</i>), • L198 (<i>The claimRerollFee function</i>) <i>XtatuzReferral.sol L:112 (The claim function)</i> <i>XtatuzProject.sol</i> <ul style="list-style-type: none"> • L: 172 (<i>The refund function</i>) • L: 183,184, and 185 (<i>The finishProject function</i>) 		

Detailed Issue

We found some usage of the *ERC20's transfer* and *transferFrom* functions that are providing unsafe token transfers (e.g., the *addProjectMember* function in L123 in the code snippet below) as follows.

1. In the ***addProjectMember* function (L123 in *XtatuzRouter.sol*)**
2. In the ***nftReroll* function (L185 in *XtatuzRouter.sol*)**
3. In the ***claimRerollFee* function (L198 in *XtatuzRouter.sol*)**
4. In the ***claim* function (L112 in *XtatuzReferral.sol*)**
5. In the ***refund* function (L172 in *XtatuzProject.sol*)**
6. In the ***finishProject* function (L183,184, and185 in *XtatuzProject.sol*)**

The use of unsafe functions could lead to unexpected token transfer errors.

XtatuzRouter.sol

```

108 function addProjectMember(
109     uint256 projectId_,
110     uint256[] memory nftList_,
111     string memory referral_
112 ) public {
113     uint256 amount = nftList_.length;
114     address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
115     IXtatuzProject project = IXtatuzProject(projectAddress);
116     uint256 price = project.addProjectMember(msg.sender, nftList_);
117
118     uint256 minPrice = project.minPrice();
119     IXtatuzReferral referralContract = IXtatuzReferral(_referralAddress);
120     referralContract.increaseBuyerRef(projectId_, referral_, amount * minPrice);
121
122     address tokenAddress = project.tokenAddress();
123     IERC20(tokenAddress).transferFrom(msg.sender, projectAddress, price);
124
125     uint256[] memory memberedProject = _memberdProject[msg.sender];
126     bool foundedIndex;
127     for (uint256 index = 0; index < memberedProject.length; index++) {
128         if (memberedProject[index] == projectId_) {
129             foundedIndex = true;
130         }
131     }
132     if (!foundedIndex) {
133         _memberdProject[msg.sender].push(projectId_);
134     }
135
136     _isMemberClaimed[msg.sender][projectId_] = false;
137     emit AddProjectMember(projectId_, msg.sender, referral_, price);
138 }
```

Listing 27.1 The *addProjectMember*,
one of the functions that use an unsafe *transfer* function

Recommendations

We recommend applying the safer functions as follows.

- *ERC20's transfer* function -> ***SafeERC20's safeTransfer*** function
- *ERC20's transferFrom* function -> ***SafeERC20's safeTransferFrom*** function

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 28	Double Spending On The <i>finishProject</i> Function		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/XtatuzFactory.sol</i>		
Locations	<i>XtatuzFactory.sol L: 175 - 188</i>		

Detailed Issue

The *XtatuzProject* contract allows the contract owner or operator to finish the specified presale project through the *finishProject* function.

However, we discovered that **the *finishProject* function can be called more than once, even if it has already been invoked**, due to the lack of project status validation as shown in code snippet 17.1. This flaw has led to the incorrect transfer of funds.

XtatuzProject.sol

```

175  function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
176    onlyOperator {
177      address referralAddress = IXtatuzRouter(owner()).referralAddress();
178
179      isFinished = true;
180      multiSigMint();
181
182      uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
183      uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
184      IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
185      IERC20(tokenAddress).transfer(referralAddress, referralAmount);
186      IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
187
188    }
  
```

Listing 28.1 The *finishProject* function of the *XtatuzProject* contract

Recommendations

We recommend **implementing the proper project status validation** within the `finishProject` function. It will ensure the prevention of duplicate calls to the `finishProject` function.

Since this issue relates to ***Issue #7 (Improper Project's Statuses)*** so please refer to that issue for more details.

XstatuzProject.sol

```

175  function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
176    onlyOperator {
177      require(projectStatus() == IXtatuzProject.Status.PREPARE_FINISH, "PROJECT:
178      PROJECT_NOT_PREPARE_FINISH");
179      address referralAddress = IXtatuzRouter(owner()).referralAddress();
180
181      isFinished = true;
182      multiSigMint();
183
184      uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
185      uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
186      IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
187      IERC20(tokenAddress).transfer(referralAddress, referralAmount);
188      IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
189    }
  
```

Listing 28.2 The improved `finishProject` function of the `XstatuzProject` contract

XstatuzProject.sol

```

226  function projectStatus() public view returns (IXtatuzProject.Status status) {
227    if (isFinished) {
228      return IXtatuzProject.Status.FINISH;
229    } else if (countReserve == 0 || _underwriteCount >= countReserve) {
230      return IXtatuzProject.Status.PREPARE_FINISH;
231    } else if (block.timestamp > endPresale && countReserve > 0 &&
232    !_underwriteCount >= countReserve) {
233      return IXtatuzProject.Status.REFUND;
234    } else if (!isFinished && block.timestamp >= startPresale && block.timestamp
235    <= endPresale) {
236      return IXtatuzProject.Status.AVAILABLE;
237    } else {
238      return IXtatuzProject.Status.UNAVAILABLE;
  
```

Listing 28.3 The improved `projectStatus` function that consider all possible project statuses

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 29	Unchecking Bounds Of Presale Token ID		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Status	High
Associated Files	<i>contract/XtatzuProject.sol</i> <i>contract/Presaled.sol</i>		
Locations	<i>XtatzuProject.sol L: 190 - 195</i> <i>Presaled.sol L: 43 - 55</i>		

Detailed Issue

The *ownerClaimLeft* function allows the project owner to claim the left of *Presaled* tokens after the project status is set to finish.

However, suppose the project owner mistakenly passes an unbound token ID into the *leftNFTList* parameter. In that case, the mint function will mint an incorrect token ID that is outside the allowed range, and this can cause effects on the system.

For more details, the *addProjectMember* function verifies that the token ID provided in the *nftList* parameter is within the allowed range of the count state variable before proceeding to mint the tokens. **In contrast, the *ownerClaimLeft* function does not validate the token ID, which can result in the minting of invalid tokens (code snippet 29.3).**

XtatzuProject.sol

```

190  function ownerClaimLeft(uint256[] memory leftNFTList) public onlyProjectOwner {
191      require(projectStatus() == IXtatzuProject.Status.FINISH, "PROJECT:
PROJECT_UNFINISH");
192      _checkAvailableNFT(leftNFTList);
193      _pickupAvailableNFT(leftNFTList, msg.sender);
194      IPresaled(_presaledAddress).mint(msg.sender, leftNFTList);
195  }

```

Listing 29.1 The *ownerClaimLeft* function

Presaled.sol

```

43  function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
44    require(to != address(0), "Presaled: Reciever address is address 0");
45    uint256 amount = tokenIdList_.length;
46    for (uint256 index = 0; index < amount; index++) {
47      uint256 tokenId = tokenIdList_[index];
48      require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
49      _safeMint(to, tokenId);
50      _mintedTimestamp[tokenId] = block.timestamp;
51      _tokenIdCount.increment();
52    }
53    setApprovalForAll(_operator, true);
54    emit Minted(to, tokenIdList_, tokenIdList_.length);
55  }

```

Listing 29.2 The *refund* function**XstatuzProject.sol**

```

23  uint256 public count;

// (...SNIPPED...)
120 function addProjectMember(address member_, uint256[] memory nftList_)
121   public
122   isAvailable
123   isLeftReserve
124   whenNotPaused
125   onlyOwner
126   returns (uint256)
127 {
128   uint256 amount = nftList_.length;
129   require(amount > 0, "PROJECT: ZERO_AMOUNT");
130   for(uint i = 0; i < amount; ++i){
131     require(nftList_[i] <= count, "PROJECT: ID_OVER_FRAGMENT");
132   }
133   _checkAvailableNFT(nftList_);
134
135   uint256 price = minPrice * amount;
136   projectValue += price;
137
138   _pickupAvailableNFT(nftList_, member_);
139
140   countReserve -= amount;
141
142   projectMember.push(member_);
143
144   IPresaled(_presaledAddress).mint(member_, nftList_);
145   emit AddProjectMember(projectId, member_, price);
146

```

```

147     return price;
148 }
```

Listing 29.3 The `addProjectMember` function

Recommendations

To prevent the minting of incorrect token IDs in the `ownerClaimLeft` function, we recommend implementing a validation condition to check that the token ID falls within the allowed range. This condition can be similar to the one used in the `addProjectMember` function, which checks that the token ID provided in the `nftList` parameter is within the allowed range of the `count` state variable.

XtatuzProject.sol

```

190 function ownerClaimLeft(uint256[] memory leftNFTList) public onlyProjectOwner {
191     require(projectStatus() == IXtatuzProject.Status.FINISH, "PROJECT:
192 PROJECT_UNFINISH");
193     uint256 amount = leftNFTList.length;
194     require(amount > 0, "PROJECT: ZERO_AMOUNT");
195     for(uint i = 0; i < amount; ++i) {
196         require(leftNFTList[i] != 0 && leftNFTList[i] <= count, "PROJECT:
197 INVALID_FRAGMENT");
198     }
199     _checkAvailableNFT(leftNFTList);
200     _pickupAvailableNFT(leftNFTList, msg.sender);
201     IPresaled(_presaledAddress).mint(msg.sender, leftNFTList);
```

Listing 29.4 The improve `ownerClaimLeft` function

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

Reassessment

This issue was acknowledged by the Xtatuz team.

No. 30	Lack Of Resetting States On Operator And Trustee Transfers		
Risk	Medium	Likelihood	Medium
Functionality is in use	In use	Status	Acknowledged
Associated Files	<i>contracts/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 197 - 216 and 265 - 309</i>		

Detailed Issue

The *multiSigMint* function (L197 - 204 in code snippet 30.1) requires the operator and trustee to call this function once to execute the *mintMaster* functions (L201 in code snippet 30.2).

However, we noticed that the operator and trustee could transfer to another by calling the *transferOperator* and *transferTrustee* functions.

For this reason, the issue occurs when the operator or trustee calls the *multiSigMint* function before transferring the right to another address.

Let's consider the following scenario to understand this issue.

Assume that, **OperatorA** is an operator and **TrusteeA** is a trustee.

1. The **TrusteeA** calls the *multiSigMint* function.

At this step, the *_multiSigMint[TrusteeA's address] = true*.

2. The **TrusteeA** calls the *transferTrustee(TrusteeB's address)* to transfer the right to **TrusteeB**.

After this step, if **OperatorA** invokes the *multiSigMint* function the condition (L200 in code snippet 16.1) would not pass because of the state.

_multiSigMint[OperatorA's address] = true, and

_multiSigMint[TrusteeB's address] = false.

3. The **TrusteeB** calls the *transferTrustee(TrusteeA's address)* to transfer the right back to **TrusteeA**.
4. An **OperatorA** invokes the *multiSigMint* function

In this step, the `_multiSigMint[OperatorA's address] = true`, and

`_multiSigMint[TrusteeA's address] = true` (the result of step 1).

Then, the condition `_multiSigMint[_operatorAddress] && _multiSigMint[_trusteeAddress]` (L200 in code snippet 30.1) would be passed and perform the `mintMaster` function immediately.

The root cause of this issue is that the `transferOperator` and `transferTrustee` functions (L265 -267 and 269 - 271 in code snippet 30.2) are not clearing the `_multiSigMint` state before transferring, resulting in the `multiSigMint` function using the stale state for consideration (L200 in code snippet 30.1).

A similar issue is shared in the `multiSigBurn` function (L206 - 216 in code snippet 30.1) as well.

XstatuzProject.sol

```

197 function multiSigMint() public isFullReserve spvAndTrustee {
198     _multiSigMint[msg.sender] = true;
199
200     if (_multiSigMint[_operatorAddress] && _multiSigMint[_trusteeAddress]) {
201         IProperty(_propertyAddress).mintMaster();
202         checkCanClaim = true;
203     }
204 }
205
206 function multiSigBurn() public isFullReserve spvAndTrustee {
207     address masterOwner = IERC721(_propertyAddress).ownerOf(0);
208     require(masterOwner == _propertyAddress, "PROJECT: NOT_MASTER_OWNER");
209
210     _multiSigBurn[msg.sender] = true;
211
212     if (_multiSigBurn[_operatorAddress] && _multiSigBurn[_trusteeAddress]) {
213         IProperty(_propertyAddress).burnMaster();
214         checkCanClaim = false;
215     }
216 }
```

Listing 30.1 The `multiSigMint` and `_multiSigBurn` functions that require the operator and trustee to perform the function functionality

XstatuzProject.sol

```

265 function transferOperator(address newOperator_) public whenNotPaused
266     onlyOperator {
267     _transferOperator(newOperator_);
268 }
269 function transferTrustee(address newTrustee_) public whenNotPaused onlyOperator
270 {
    _transferTrustee(newTrustee_);
```

```

271   }

    // (...SNIPPED...)

299   function _transferOperator(address newOperator_) internal
300     ProhibitZeroAddress(newOperator_) {
301       address prevOperator = _operatorAddress;
302       _operatorAddress = newOperator_;
303       emit OperatorTransferred(prevOperator, newOperator_);
304     }
305
306   function _transferTrustee(address newTrustee_) internal
307     ProhibitZeroAddress(newTrustee_) {
308       address prevTrustee = _trusteeAddress;
309       _trusteeAddress = newTrustee_;
310       emit TrusteeTransferred(prevTrustee, newTrustee_);
311     }
  
```

Listing 30.2 The `transferOperator` and `transferTrustee` functions that lack clearing state.

Recommendations

We recommend clearing the `_multiSigMint` and `_multiSigBurn` states before transferring like L301 - 302 and 309 - 310 in code snippet 30.3.

XtatuzProject.sol

```

299   function _transferOperator(address newOperator_) internal
300     ProhibitZeroAddress(newOperator_) {
301       address prevOperator = _operatorAddress;
302       _multiSigMint[prevOperator] = false;
303       _multiSigBurn[prevOperator] = false;
304       _operatorAddress = newOperator_;
305       emit OperatorTransferred(prevOperator, newOperator_);
306     }
307
308   function _transferTrustee(address newTrustee_) internal
309     ProhibitZeroAddress(newTrustee_) {
310       address prevTrustee = _trusteeAddress;
311       _multiSigMint[prevTrustee] = false;
312       _multiSigBurn[prevTrustee] = false;
313       _trusteeAddress = newTrustee_;
314       emit TrusteeTransferred(prevTrustee, newTrustee_);
  
```

Listing 30.3 The improved `_transferOperator` and `_transferTrustee` functions.

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

Reassessment

This issue was acknowledged by the *Xtatuz* team.

No. 31	Unsafe Privileged Role Transfer		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	contracts/XtatuzReferral.sol contracts/XtatuzProject.sol contracts/XtatuzRouter.sol		
Locations	XtatuzReferral.sol L: 140 - 142 XtatuzProject.sol L: 261 - 263, and 265 - 267 XtatuzRouter.sol L: 273 - 275		

Detailed Issue

The `XtatuzReferral`, `XtatuzProject`, and `XtatuzRouter` contracts implement the following privileged role transfer functions:

- The **`XtatuzReferral`** contract
 - The `transferOperator` function.
- The **`XtatuzProject`** contract
 - The `transferProjectOwner` function.
 - The `transferOperator` function.
- The **`XtatuzRouter`** contract
 - The `transferSpv` function.

These functions can transfer the operator, project owner and/or `Xtatuz` special purpose vehicle (SPV) of the contract from the current one to another.

XtatuzProject.sol

```

265  function transferOperator(address newOperator_) public whenNotPaused
266    onlyOperator {
267      _transferOperator(newOperator_);
268    }
269
270    //(...SNIPPED...)
271
272  function _transferOperator(address newOperator_) internal
273    ProhibitZeroAddress(newOperator_) {
274    address prevOperator = _operatorAddress;
275  }

```

```

301     _operatorAddress = newOperator_;
302     emit OperatorTransferred(prevOperator, newOperator_);
303 }
```

Listing 31.1 The *transferOperator* function of the *XtatuzProject*,
the example privileged role transfer function.

From the code snippet above, the address variable *newOperator_* (L265 in code snippet 31.1) may be incorrectly specified by the current operator by mistake; for example, an address that a new operator does not own was inputted. Consequently, the new operator loses ownership of the contract immediately, and this action is unrecoverable.

Recommendations

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

XtatuzProject.sol

```

265 function transferOperator(address _candidateOperator) public whenNotPaused
onlyOperator {
266     require(_candidateOperator != address(0), "PROJECT: candidate operator is the
zero address");
267     candidateOperator = _candidateOperator;
268     emit NewCandidateOperator(_candidateOperator);
269 }
270
271 function claimOperator() external {
272     require(candidateOperator == _msgSender(), "PROJECT: caller is not the
candidate operator");
273     _transferOperator(candidateOperator);
274     candidateOperator = address(0);
275 }
```

Listing 31.2 The example recommended two-step privileged role transfer mechanism for the *XtatuzProject* contract

This mechanism works as follows.

1. The current operator or the current owner invokes the *transferOperator* function by specifying the candidate operator address *_candidateOperator* (L265 in code snippet 31.2).
2. The candidate operator proves access to his/her account and claims the operator role transfer by invoking the *claimOperator* function (L271 - 275 in code snippet 31.2)

The recommended mechanism ensures that the roled users of the contract would be transferred to another one who can access his/her account only.

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

Reassessment

This issue was acknowledged by the *Xtatuz* team.

No. 32	Lack Of Proper Validation On The underwriteCount Parameter		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	<i>contract/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 317 - 332</i>		

Detailed Issue

We noticed that the `_initialData` function lacks proper validation on the `count` and `underwriteCount` parameters. This is causing issues with a system operation, such as the `_extendEndPresale` function, which may result in incorrect data being stored or processed.

More specifically, if the `count` state variable is less than the `underwriteCount` state variable, the `absoluteCount` variable will be negative (L276 in code snippet 32.3). This causes the transaction to revert when the `_extendEndPresale` function is invoked.

In other cases, improper validation can affect the `_checkIsFullReserve` function and subsequently impact the state condition of the project (L350 in code snippet 32.4).

XtatuzProject.sol

```

317 function _initialData(
318     uint256 count_,
319     uint256 underwriteCount_,
320     address tokenAddress_,
321     address propertyAddress_,
322     address presaledAddress_
323 ) internal {
324     require(count_ > 0, "PROJECT: COUNT_ZERO");
325     require(tokenAddress_ != address(0), "PROJECT: ADDRESS_ZERO");
326     count = count_;
327     countReserve = count_;
328     underwriteCount = underwriteCount_;
329     tokenAddress = tokenAddress_;
330     propertyAddress = propertyAddress_;
331     presaledAddress = presaledAddress_;
332 }
```

Listing 32.1 The `_initialData` function

XstatuzProject.sol

```

111  function setUnderwriteCount(uint256 underwriteCount_) public onlyOperator {
112      require(underwriteCount_ < count, "PROJECT: INVALID_COUNT");
113      _underwriteCount = underwriteCount_;
114  }

```

Listing 32.2 The `setUnderwriteCount` function**XstatuzProject.sol**

```

273  function _extendEndPresale() internal {
274      require(_isTriggeredEndpresale == false, "PROJECT: EXTENDED_PRESALE");
275      if (block.timestamp > (endPresale - 1 days)) {
276          uint256 absoluteCount = count - _underwriteCount;
277          uint256 percent = ((count - countReserve) * 100) / absoluteCount;
278          if (percent >= 95) {
279              endPresale += 5 days;
280          } else if (percent >= 85 && percent < 95) {
281              endPresale += 10 days;
282          } else if (percent >= 65 && percent < 85) {
283              endPresale += 15 days;
284          } else {
285              endPresale += 30 days;
286          }
287          _isTriggeredEndpresale = true;
288      }
289  }

```

Listing 32.3 The `_extendEndPresale` function**XstatuzProject.sol**

```

349  function _checkIsFullReserve() internal view {
350      require(_underwriteCount >= countReserve, "PROJECT: NOT_FULL");
351  }

```

Listing 32.4 The `_checkIsFullReserve` function

Recommendations

We recommend using the `setUnderwriteCount` function to ensure that the `_underwriteCount` state is set correctly.

XstatuzProject.sol

```
317 function _initialData(
318     uint256 count_,
319     uint256 underwriteCount_,
320     address tokenAddress_,
321     address propertyAddress_,
322     address presaledAddress_
323 ) internal {
324     require(count_ > 0, "PROJECT: COUNT_ZERO");
325     require(tokenAddress_ != address(0), "PROJECT: ADDRESS_ZERO");
326     count = count_;
327     countReserve = count_;
328     setUnderwriteCount(underwriteCount_);
329     tokenAddress = tokenAddress_;
330     _propertyAddress = propertyAddress_;
331     _presaledAddress = presaledAddress_;
332 }
```

Listing 32.5 The improved `_initialData` function

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

Reassessment

The Xstatuz team has acknowledged this issue and indicated that the number of `_underwriteCount` will be determined by the business model.

No. 33	Lack Of Upper Bound On Referral Percentage		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	<code>contracts/XtatuzReferral.sol</code>		
Locations	<code>XtatuzReferral.sol L: 115-120, and L: 128-138</code>		

Detailed Issue

The `XtatuzReferral` contract provides multiple levels of referral mechanisms. The percentage of each level will be utilized for the calculation of the referral amount, as shown in code snippet 33.1.

The percentage could be specified via the `setDefaultPercentage` and `setReferralLevels` functions as shown in code snippet 33.2.

We, however, discovered that **there is no upper bound on the percentage value that can be set by these functions. This could lead to the provision of incorrect percentages, resulting in insufficient commission funds to be claimed by the agent.**

XtatuzReferral.sol

```

71  function increaseBuyerRef(uint256 projectId_, string memory referral_, uint256
72    amount_) public onlyOperator {
73    address agentWallet = addressByReferral[referral_];
74    require(agentWallet != address(0), "REFERRAL: INVALID_REFERRAL");
75    uint256 level = referralLevel[projectId_][referral_];
76    uint256 percentage = defaultPercentage;
77
78    if(level != 0) {
79      percentage = levelsPercentage[level];
80    }
81
82    uint256 referralAmount = (amount_ * percentage) / 100;
83
84    uint256[] memory projectIdList = projectIdsByReferral[referral_];
85
86    bool foundedIndex;
87    for (uint256 index = 0; index < projectIdList.length; index++) {
88      if (projectIdList[index] == projectId_) {

```

```

88         foundedIndex = true;
89     }
90   }
91   if (!foundedIndex) {
92     projectIdsByReferral[referral_].push(projectId_);
93   }
94
95   buyerAgentAmount[referral_][projectId_] += referralAmount;
96   updateReferralAmount(projectId_, amount_);
97   emit IncreaseBuyerRef(projectId_, referralAmount);
98 }
```

Listing 33.1 The *increaseBuyerRef* function of the *XtatuzReferral* contract**XtatuzReferral.sol**

```

115  function setDefaultPercentage(uint256 default_) public onlyOperator {
116    require(default_ > 0, "REFERRAL: NO_ZERO_PERCENT");
117    uint256 prev = defaultPercentage;
118    defaultPercentage = default_;
119    emit ChangeDefaultPercent(prev, default_);
120  }

//(...SNIPPED...)

128  function setReferralLevels(uint256[] memory percentagePerLevel_) public
onlyOperator {
129    uint256[] memory prevLevels = new uint256[](3);
130    uint256[] memory newLevels = new uint256[](3);
131    require(percentagePerLevel_.length == 3, "REFERRAL: 3_LEVELS");
132    for(uint256 i = 0; i < percentagePerLevel_.length ; i++){
133      prevLevels[i] = levelsPercentage[i + 1];
134      levelsPercentage[i + 1] = percentagePerLevel_[i];
135      newLevels[i] = levelsPercentage[i + 1];
136    }
137    emit SetReferralLevels(prevLevels, newLevels);
138 }
```

Listing 33.2 The *setDefaultPercentage* and *setReferralLevels* functions of the *XtatuzReferral* contract

Recommendations

We recommend **setting the maximum upper bound of the referral fee percentage** in the `setDefaultPercentage` and `setReferralLevels` functions as shown in the code snippet 33.3.

XtatuzReferral.sol

```

115 function setDefaultPercentage(uint256 default_) public onlyOperator {
116     require(default_ > 0 && default_ <= MAX_PERCENT, "REFERRAL:
117     INVALID_PERCENT");
118     uint256 prev = defaultPercentage;
119     defaultPercentage = default_;
120     emit ChangeDefaultPercent(prev, default_);
121 }
122
123 //(...SNIPPED...)
124
125 function setReferralLevels(uint256[] memory percentagePerLevel_) public
126 onlyOperator {
127     uint256[] memory prevLevels = new uint256[](3);
128     uint256[] memory newLevels = new uint256[](3);
129     require(percentagePerLevel_.length == 3, "REFERRAL: 3_LEVELS");
130     require(percentagePerLevel_[0] < percentagePerLevel_[1] &&
131     percentagePerLevel_[1] < percentagePerLevel_[2], "REFERRAL:
132     INVALID_PERCENT_LEVELS");
133     for(uint256 i = 0; i < percentagePerLevel_.length ; i++){
134         require(percentagePerLevel_[i] > 0 && percentagePerLevel_[i] <= MAX_PERCENT,
135         "REFERRAL: INVALID_PERCENT");
136         prevLevels[i] = levelsPercentage[i + 1];
137         levelsPercentage[i + 1] = percentagePerLevel_[i];
138         newLevels[i] = levelsPercentage[i + 1];
139     }
140     emit SetReferralLevels(prevLevels, newLevels);
141 }
```

Listing 33.3 The improved `setDefaultPercentage` and `setReferralLevels` functions of the `XtatuzReferral` contract

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue by introducing the constant **MAX_PERCENTAGE** variable (L25 in the code snippet below) to set an upper bound on the maximum percentage that can be applied to the referral fee percentage.

XtatuzReferral.sol

```

10  contract XtatuzReferral is Ownable {
11      // (...SNIPPED...)
12      uint256 public constant MAX_PERCENTAGE = 5;
13      uint256 public maxPercentage = 5;
14      uint256 public defaultPercentage = 3;
15      // (...SNIPPED...)
16      function setDefaultPercentage(uint256 default_) public onlyOperator {
17          require(default_ > 0 && default_ <= maxPercentage, "REFERRAL:
18          INVALID_PERCENT");
19          uint256 prev = defaultPercentage;
20          defaultPercentage = default_;
21          emit ChangeDefaultPercent(prev, default_);
22      }
23
24
25      function setMaxPercentage(uint256 max_) public onlyOperator {
26          require(max_ > 0 && max_ <= MAX_PERCENTAGE, "REFERRAL:
27          INVALID_PERCENT");
28          uint256 prev = maxPercentage;
29          maxPercentage = max_;
30          emit ChangeMaxPercent(prev, max_);
31      }
32      // (...SNIPPED...)
33      function setReferralLevels(uint256[] memory percentagePerLevel_) public
34          onlyOperator {
35          uint256[] memory prevLevels = new uint256[](3);
36          uint256[] memory newLevels = new uint256[](3);
37          require(percentagePerLevel_.length == 3, "REFERRAL: 3_LEVELS");
38          require(
39              percentagePerLevel_[0] < percentagePerLevel_[1] &&
40              percentagePerLevel_[1] < percentagePerLevel_[2],
41              "REFERRAL: INVALID_PERCENT_LEVELS"
42          );
43          for (uint256 i = 0; i < percentagePerLevel_.length; i++) {
44              require(percentagePerLevel_[i] > 0 && percentagePerLevel_[i] <=
45                  maxPercentage, "REFERRAL: INVALID_PERCENT");
46              prevLevels[i] = levelsPercentage[i + 1];
47              levelsPercentage[i + 1] = percentagePerLevel_[i];
48              newLevels[i] = levelsPercentage[i + 1];
49          }
50          emit SetReferralLevels(prevLevels, newLevels);
51      }
52      // (...SNIPPED...)
53  }

```

Listing 33.4 The improved setting percentage process of the *XtatuzReferral* contract

No. 34	Incorrectly Updating Referral Amount		
Risk	Medium	Likelihood	High
Functionality is in use	In use	Impact	Low
Associated Files	<code>contract/XtatuzReferral.sol</code>		
Locations	<code>XtatuzReferral.sol L: 59 - 61, L: 71 - 98</code>		

Detailed Issue

The `updateReferralAmount` function is responsible for updating the `_referralAmount` state, which summarizes the referral amounts for each project (L59 - 61 in code snippet 34.2).

The `amount_` parameter represents the total amount of the order and is passed into the `updateReferralAmount` function (L71 in code snippet 34.1), while the `referralAmount` variable is calculated from the order amount and is used to determine the referral fee for that order (L81 in code snippet 34.1).

We found that the `updateReferralAmount` was being passed the wrong parameter to update the `_referralAmount` state, which caused the state to be updated incorrectly.

As a result, using the incorrect parameter could result in inconsistencies in the total of referral fee.

XtatuzReferral.sol

```

71  function increaseBuyerRef(uint256 projectId_, string memory referral_, uint256
72    amount_) public onlyOperator {
73      // (...SNIPPED...)
74
75      uint256 referralAmount = (amount_ * percentage) / 100;
76
77      uint256[] memory projectIdList = projectIdsByReferral[referral_];
78
79      bool foundedIndex;
80      for (uint256 index = 0; index < projectIdList.length; index++) {
81          if (projectIdList[index] == projectId_) {
82              foundedIndex = true;
83          }
84      }
85      if (!foundedIndex) {
86          projectIdsByReferral[referral_].push(projectId_);
87      }
88  }
89
90
91
92
93

```

```

94     buyerAgentAmount[referral_][projectId_] += referralAmount;
95     updateReferralAmount(projectId_, amount_);
96     emit IncreaseBuyerRef(projectId_, referralAmount);
97 }

```

Listing 34.1 The *increaseBuyerRef* function

XstatuzReferral.sol

```

59     function updateReferralAmount(uint256 projectId_, uint256 amount_) public
60         onlyOperator {
61         _referralAmount[projectId_] += amount_;
62     }

```

Listing 34.2 The *updateReferralAmount* function

Recommendations

To resolve the issue with updating the *referralAmount* state, we suggest using the *referralAmount* variable (*L81 in code snippet 34.3*) as a parameter for the *updateReferralAmount* function instead of the *amount_* parameter. This will ensure that the *_referralAmount* state is updated correctly based on the referral fee calculated from the order amount.

XstatuzReferral.sol

```

71     function increaseBuyerRef(uint256 projectId_, string memory referral_, uint256
72         amount_) public onlyOperator {
73         // (...SNIPPED...)
74
75         uint256 referralAmount = (amount_ * percentage) / 100;
76
77         uint256[] memory projectIdList = projectIdsByReferral[referral_];
78
79         bool foundedIndex;
80         for (uint256 index = 0; index < projectIdList.length; index++) {
81             if (projectIdList[index] == projectId_) {
82                 foundedIndex = true;
83             }
84         }
85         if (!foundedIndex) {
86             projectIdsByReferral[referral_].push(projectId_);
87         }
88
89         buyerAgentAmount[referral_][projectId_] += referralAmount;
90         updateReferralAmount(projectId_, referralAmount);
91         emit IncreaseBuyerRef(projectId_, referralAmount);
92     }

```

Listing 34.3 The improved *increaseBuyerRef* function

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 35	Recommended Improving Transparency And Trustworthiness		
Risk	Medium	Likelihood	Low
Functionality is in use	In use	Impact	High
Associated Files	<i>contracts/XtatuzProject.sol</i> <i>contracts/XtatuzReroll.sol</i>		
Locations	<i>XtatuzProject.sol</i> L: 197 - 204 and 206 - 216 <i>XtatuzReroll.sol</i> L: 44 - 47 and 49 - 51		

Detailed Issue

The following lists all privileged functions that should be improved transparency and trustworthiness:

- The **XtatuzReroll** contract
 - The *setRerollData* function.
 - The *setFee* function.
- The **XtatuzProject** contract
 - The *multiSigMint* function.
 - The *multiSigBurn* function.

Our analysis found that those functions listed can change important states, which could affect the users' assets. For this reason, we consider that those functions should be improved for transparency and trustworthiness.

Recommendations

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

Reassessment

This issue was acknowledged by the *Xtatuz* team.

No. 36	Permanent Lock Of Referral Fee		
Risk	Medium	Likelihood	Medium
Functionality is in use	In use	Status	Fixed
Associated Files	<code>contracts/XtatuzReferral.sol</code>		
Locations	<code>XtatuzReferral.sol L:71 - 98</code>		

Detailed Issue

The `XtatuzProject` contract allows the contract owner or operator to finish the specified presale project through the `finishProject` function. A fixed 5% fee of the project value is charged for the `XtatuzReferral` contract (L181 and 184 in the code snippet 36.2).

The `XtatuzReferral` contract provides multiple levels of referral mechanisms. The percentage is used in the calculator to track the total referral amount of each agent (L71 - 98 in the code snippet 36.1), and agents are able to claim their commission once the project has been finished.

However, we found that there is a potential for tokens to remain in the `XtatuzReferral` contract. This is due to the dynamic change of the referral percentage, which can fluctuate and be less than the fixed percentage (5%) that is charged in the `XtatuzProject` contract, and there is no mechanism to withdraw any remaining tokens.

XtatuzReferral.sol

```

71  function increaseBuyerRef(uint256 projectId_, string memory referral_, uint256
72    amount_) public onlyOperator {
73      address agentWallet = addressByReferral[referral_];
74      require(agentWallet != address(0), "REFERRAL: INVALID_REFERRAL");
75      uint256 level = referralLevel[projectId_][referral_];
76      uint256 percentage = defaultPercentage;
77
78      if(level != 0) {
79          percentage = levelsPercentage[level];
80      }
81
82      uint256 referralAmount = (amount_ * percentage) / 100;
83
84      uint256[] memory projectIdList = projectIdsByReferral[referral_];

```

```

84
85     bool foundedIndex;
86     for (uint256 index = 0; index < projectIdList.length; index++) {
87         if (projectIdList[index] == projectId_) {
88             foundedIndex = true;
89         }
90     }
91     if (!foundedIndex) {
92         projectIdByReferral[referral_].push(projectId_);
93     }
94
95     buyerAgentAmount[referral_][projectId_] += referralAmount;
96     updateReferralAmount(projectId_, amount_);
97     emit IncreaseBuyerRef(projectId_, referralAmount);
98 }
```

Listing 36.1 The *increaseBuyerRef* function of the *XstatuzReferral* contract**XstatuzProject.sol**

```

175     function finishProject(address xstatuzWallet_) public isFullReserve whenNotPaused
176     onlyOperator {
177         address referralAddress = IXstatuzRouter(owner()).referralAddress();
178
179         isFinished = true;
180         multiSigMint();
181
182         uint256 referralAmount = (((count - countReserve) * minPrice) * 5) / 100;
183         uint256 xstatuzAmount = ((count * minPrice) * 10) / 100;
184         IERC20(tokenAddress).transfer(_projectOwner, projectValue - xstatuzAmount);
185         IERC20(tokenAddress).transfer(referralAddress, referralAmount);
186         IERC20(tokenAddress).transfer(xstatuzWallet_, xstatuzAmount - referralAmount);
187
188         emit FinishProject(projectId, xstatuzWallet_);
189     }
```

Listing 36.2 The *finishProject* function of the *XstatuzProject* contract

Recommendations

We recommend applying the mapping variable to track the total remaining token, *buyerAgentDepositLeft* mapping (L97 in the code snippet 36.3) and introducing the *withdrawFundsLeft* function (L158 - 168 in the code snippet 36.3) that allows only the contract owner to withdraw the remaining tokens once the project has been finished.

XtatuZReferral.sol

```

71  function increaseBuyerRef(uint256 projectId_, string memory referral_, uint256
72    amount_) public onlyOperator {
73      address agentWallet = addressByReferral[referral_];
74      require(agentWallet != address(0), "REFERRAL: INVALID_REFERRAL");
75      uint256 level = referralLevel[projectId_][referral_];
76      uint256 percentage = defaultPercentage;
77
78      if(level != 0) {
79          percentage = levelsPercentage[level];
80      }
81
82      uint256 referralAmount = (amount_ * percentage) / 100;
83      uint256 totalDeposit = (amount_ * 5) / 100;
84
85      uint256[] memory projectIdList = projectIdsByReferral[referral_];
86
87      bool foundedIndex;
88      for (uint256 index = 0; index < projectIdList.length; index++) {
89          if (projectIdList[index] == projectId_) {
90              foundedIndex = true;
91          }
92          if (!foundedIndex) {
93              projectIdsByReferral[referral_].push(projectId_);
94          }
95
96          buyerAgentAmount[referral_][projectId_] += referralAmount;
97          buyerAgentDepositLeft[referral_][projectId_] += (totalDeposit -
98          referralAmount);
99          updateReferralAmount(projectId_, amount_);
100         emit IncreaseBuyerRef(projectId_, referralAmount);
101     }

158  function withdrawFundsLeft(string memory referral_, uint projectId_) external
159    onlyOwner {
160      uint256 amount = buyerAgentDepositLeft[referral_][projectId_];
161      require(amount > 0, "REFERRAL: NO_LEFT_FUND");
162      address projectAddress =

```

```
162     IXtatuzRouter(owner()).getProjectAddressById(projectId_);
163     require(projectAddress != address(0), "REFERRAL: INVALID_PROJECT_ID");
164     IXtatuzProject.Status status =
165         IXtatuzProject(projectAddress).projectStatus();
166     require(status == IXtatuzProject.Status.FINISH, "REFERRAL:
167     PROJECT_NOT_FINISH");
168     buyerAgentDepositLeft[referral_][projectId_] = 0;
169     IERC20(tokenAddress).safeTransfer(msg.sender, amount);
170 }
```

Listing 36.3 The improved *increaseBuyerRef* function and the implementation of the *withdrawFundsLeft* function

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 37	Incorrect Time Verification On Extending Presale Period		
Risk	Low	Likelihood	Low
Functionality is in use	In use	Impact	Medium
Associated Files	<i>contract/XtatuzProject.sol</i>		
Locations	<i>XtatuzProject.sol L: 218 - 224, L: 273 - 289</i>		

Detailed Issue

We discovered the improper logic in the statement, which allows the operator to incorrectly extend the end presale date.

The root cause of this issue is the ***block.timestamp > (endPresale - 1 days)*** statement (L219 in code snippet 37.1 and L275 in code snippet 37.2). This statement allows access not only to the day preceding the end of the presale date but also to the day following it because the ***block.timestamp*** keeps increasing, which makes this state will always pass after the presale end date is reached.

As a result, if the ***extendEndPresale*** function is called by an operator after the end presale date, the calculation of the ***endPresale*** state variable will produce an incorrect result.

XtatuzProject.sol

```

218  function extendEndPresale() public onlyOperator {
219      require(block.timestamp > (endPresale - 1 days), "PROJECT: NOT_END_PREV");
220      require(_isTriggeredEndpresale == false, "PROJECT: EXTENDED_PRESALE");
221      if (_isTriggeredEndpresale == false) {
222          _extendEndPresale();
223      }
224  }
```

Listing 37.1 The ***extendEndPresale*** function

XtatuzProject.sol

```

273   function _extendEndPresale() internal {
274     require(_isTriggeredEndpresale == false, "PROJECT: EXTENDED_PRESALE");
275     if (block.timestamp > (endPresale - 1 days)) {
276       uint256 absoluteCount = count - _underwriteCount;
277       uint256 percent = ((count - countReserve) * 100) / absoluteCount;
278       if (percent >= 95) {
279         endPresale += 5 days;
280       } else if (percent >= 85 && percent < 95) {
281         endPresale += 10 days;
282       } else if (percent >= 65 && percent < 85) {
283         endPresale += 15 days;
284       } else {
285         endPresale += 30 days;
286       }
287       _isTriggeredEndpresale = true;
288     }
289   }

```

Listing 37.2 The `_extendEndPresale` function**Recommendations**

We recommend improving the `require` statement to ensure that the operator can only call the `extendEndPresale` function the day preceding the end presale date following the business requirement.

XtatuzProject.sol

```

218   function extendEndPresale() public onlyOperator {
219     _extendEndPresale();
220   }

// (...SNIPPED...)

269   function _extendEndPresale() internal {
270     require(block.timestamp > (endPresale - 1 days) && block.timestamp <
endPresale, "PROJECT: ONLY_THE_EXTENDING_PERIOD");
271     require(_isTriggeredEndpresale == false, "PROJECT: EXTENDED_PRESALE");

273     uint256 absoluteCount = count - _underwriteCount;
274     uint256 percent = ((count - countReserve) * 100) / absoluteCount;
275     if (percent >= 95) {
276       endPresale += 5 days;
277     } else if (percent >= 85 && percent < 95) {
278       endPresale += 10 days;
279     } else if (percent >= 65 && percent < 85) {
280       endPresale += 15 days;
281     } else {

```

```
282         endPresale += 30 days;
283     }
284     _isTriggeredEndpresale = true;
285 }
```

Listing 37.3 Improved `extendEndPresale` and `_extendEndPresale` functions

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 38	Potential Denial-Of-Service On Setting Reroll Data		
Risk	Low	Likelihood	Low
Functionality is in use	In use	Impact	Medium
Associated Files	<i>contracts/XtatuzReroll.sol</i>		
Locations	<i>XtatuzReroll.sol L: 44 - 47</i>		

Detailed Issue

The `setRerollData` function of the `XtatuzReroll` contract allows the operator to set all the reroll data.

However, we noticed that in the `setRerollData` function (L44 - 49 in the code snippet below), there are assign all `rerollData_` arrays to the `rerollData[projectId_]` directly (L48 in the code snippet below). Behind the *ETHEREUM VIRTUAL MACHINE (EVM)*, the statement would iterate over all array elements to store.

This process can consume a lot of gas and the gas used is prone to exceeding the block gas limit if the operator needs to provide a large number of `rerollData_` (L44 in the code snippet below). In the case of exceeding the block gas limit, a transaction would be reverted, leading to the denial-of-service issue to the affected function.

XtatuzReroll.sol

```

44  function setRerollData(uint256 projectId_, string[] memory rerollData_) public
45    onlyOperator {
46      require(rerollData_.length > 0, "REROLL: OUT_OF_DATA");
47      rerollData[projectId_] = rerollData_;
    }
```

Listing 38.1 The `setRerollData` function that set all reroll data, which can consume more gas than the block gas limit

Recommendations

Since no recommended code or solution can fully fix this issue without breaking the contract's features, we recommend redesigning and reimplementing the operation to manage the `rerollData` state mechanism.

Reassessment

The *Xtatuz* team acknowledged this issue. However, the *Xtatuz* team would control the number of *rerollData_* array parameters to avoid the denial-of-service issue.

No. 39	Compiler Is Not Locked To Specific Version		
Risk	Low	Likelihood	Low
Functionality is in use	In use	Impact	Medium
Status	Fixed		
Associated Files	<i>contract/Presaled.sol</i> <i>contract/Property.sol</i> <i>contract/PresaledFactory.sol</i> <i>contract/PropertyFactory.sol</i> <i>contract/ProjectFactory.sol</i> <i>contract/XtatuzFactory.sol</i> <i>contract/XtatuzProject.sol</i> <i>contract/XtatusReroll.sol</i> <i>contract/XtatuzReferral.sol</i> <i>contract/XtatuzRouter.sol</i>		
Locations	<i>Presaled.sol L: 2</i> <i>Property.sol L: 2</i> <i>PresaledFactory.sol L: 2</i> <i>PropertyFactory.sol L: 2</i> <i>ProjectFactory.sol L: 2</i> <i>XtatuzFactory.sol L: 2</i> <i>XtatuzProject.sol L: 2</i> <i>XtatusReroll.sol L: 2</i> <i>XtatuzReferral.sol L: 2</i> <i>XtatuzRouter.sol L: 2</i>		

Detailed Issue

We found that the smart contracts in this project should be deployed with the compiler version used in the development and testing process.

The compiler version that is not strictly locked via the *pragma* statement may make the contract incompatible against unforeseen circumstances.

List of smart contracts that should lock to the specific version.

- **Presaled.sol**
- **Property.sol**
- **PresaledFactory.sol**
- **PropertyFactory.sol**
- **ProjectFactory.sol**
- **XtatuzFactory.sol**
- **XtatuzProject.sol**
- **XtatusReroll.sol**

- **XstatuzReferral.sol**
- **XstatuzRouter.sol**

An example code that is not locked to specific version (e.g., using => or ^ directive) is shown below

Presaled.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.17;
```

Listing 39.1 The *Presaled* contract

Recommendations

We recommend locking the pragma version like the example code snippet below.

```
pragma solidity 0.8.17;
// or
pragma solidity =0.8.17;

contract SemVerFloatingPragmaFixed {
```

Reference: <https://swcregistry.io/docs/SWC-103>

Reassessment

The Xstatuz team adopted our recommended code to fix this issue.

No. 40	Lack Of Validating Of Reroll Data		
Risk	Low	Likelihood	Low
Functionality is in use	In use	Impact	Medium
Associated Files	<i>contracts/XtatuzProject.sol</i>		
Locations	<i>XtatuzRouter.sol L: 166 - 189</i>		

Detailed Issue

The *nftReroll* function of the *XtatuzRouter* contract (L166 - 189 in the code snippet below) allows the token owner to reroll their property token URI.

We discovered, if the *rerollData* is empty (L176 in the code snippet below), the ***block.timestamp % rerollData.length*** statement (L180 in the code snippet below) would be reverted due to the ***block.timestamp % 0***.

XtatuzRouter.sol

```

166 function nftReroll(uint256 projectId_, uint256 tokenId_) public {
167     require(_rerollAddress != address(0), "ROUTER: NO_REROLL_ADDRESS");
168
169     address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
170     IProperty property = IProperty(propertyAddress);
171     IXtatuzReroll rerollContract = IXtatuzReroll(_rerollAddress);
172
173     address tokenAddress = rerollContract.tokenAddress();
174     string memory prevUri = property.tokenURI(tokenId_);
175     uint256 fee = rerollContract.rerollFee();
176     string[] memory rerollData = rerollContract.getRerollData(projectId_);
177     address tokenOwner = property.ownerOf(tokenId_);
178     require(tokenOwner == msg.sender, "ROUTER: NOT_NFT_OWNER");
179
180     uint256 newIndex = block.timestamp % rerollData.length;
181     property.setTokenURI(tokenId_, rerollData[newIndex]);
182     rerollData[newIndex] = prevUri;
183     rerollContract.setRerollData(projectId_, rerollData);
184
185     IERC20(tokenAddress).transferFrom(msg.sender, address(this), fee);
186     _totalRerollFee[projectId_] += fee;
187
188     emit NFTReroll(msg.sender, projectId_, tokenId_);

```

189 }

Listing 40.1 The *nftReroll* function of the *XtatuzProject* contract

Recommendations

We recommend **improving the validation by checking that the *rerollData* is not empty before continuing the reroll process** as shown at L177 in the code snippet below.

XtatuzRouter.sol

```

166 function nftReroll(uint256 projectId_, uint256 tokenId_) public {
167     require(_rerollAddress != address(0), "ROUTER: NO_REROLL_ADDRESS");
168
169     address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
170     IProperty property = IProperty(propertyAddress);
171     IXtatuzReroll rerollContract = IXtatuzReroll(_rerollAddress);
172
173     address tokenAddress = rerollContract.tokenAddress();
174     string memory prevUri = property.tokenURI(tokenId_);
175     uint256 fee = rerollContract.rerollFee();
176     string[] memory rerollData = rerollContract.getRerollData(projectId_);
177     require(rerollData.length > 0, "ROUTER: NO_REROLL_DATA");
178     address tokenOwner = property.ownerOf(tokenId_);
179     require(tokenOwner == msg.sender, "ROUTER: NOT_NFT_OWNER");
180
181     uint256 newIndex = block.timestamp % rerollData.length;
182     property.setTokenURI(tokenId_, rerollData[newIndex]);
183     rerollData[newIndex] = prevUri;
184     rerollContract.setRerollData(projectId_, rerollData);
185
186     IERC20(tokenAddress).transferFrom(msg.sender, address(this), fee);
187     _totalRerollFee[projectId_] += fee;
188
189     emit NFTReroll(msg.sender, projectId_, tokenId_);
190 }
```

Listing 40.2 The improved *nftReroll* function of the *XtatuzProject* contract

Reassessment

The *Xtatuz* team adopted our recommended code to fix this issue.

No. 41	Missing Validation Of Zero Address		
Risk	Informational	Likelihood	Low
Functionality is in use	In use	Impact	Low
Associated Files	<code>contracts/XtatuzRouter.sol</code>		
Locations	<code>XtatuzRouter.sol L: 43 - 50</code>		

Detailed Issue

We found that there is no validation process to prevent the assignment of address zero within the constructor of the `XtatuzRouter` contract. The usage of address zero could have unforeseen implications, which presents a potential security issue.

XtatuzRouter.sol

```

43  constructor(
44      address spv_,
45      address factoryAddress_
46  ) {
47      _transferSpv(spv_);
48      _xtatuzFactory = IXtatuzFactory(factoryAddress_);
49      _projectIdCounter.increment();
50 }
```

Listing 41.1 The `constructor` of the `XtatuzRouter` contract

Recommendations

We recommend adding the `require` statement to avoid assigning the zero address as shown in the code snippet below.

XtatuzRouter.sol

```

43  constructor(
44      address spv_,
45      address factoryAddress_
46  ) {
47      require(spv_ != address(0) && factoryAddress_ != address(0), "ROUTER: INVALID
48      ADDRESS");
```

```
48     _transferSpv(spv_);
49     _xtatuzFactory = IXtatuzFactory(factoryAddress_);
50     _projectIdCounter.increment();
51 }
```

Listing 41.2 The improved *constructor* of the *XtatuzRouter* contract

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

Reassessment

The Xtatuz team fixed this issue by adopting our recommended code.

No. 42	Predictable Randomness		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Acknowledged
Associated Files	<code>contract/XtatuzRouter.sol</code>		
Locations	<code>XtatuzRouter.sol L: 166 - 189</code>		

Detailed Issue

We found that the `nftReroll` function contains weak randomness code (L180 in code snippet 42.1), which is generated using a predictable pseudo-random number generator `newIndex` value that could potentially be manipulated by an attacker, compromising the fairness of the contract.

To elaborate, the attacker can create a malicious contract using the same logic that checks the outcome of the `newIndex` that manipulates from `block.timestamp` and `rerollData.length` variables, which allows the attacker to predict the `newIndex` result accurately.

We report this information for `acknowledgement` only since the *Property NFT* without rarity, the randomness that generates the `newIndex` is not harmful to the system.

XtatuzProject.sol

```

166  function nftReroll(uint256 projectId_, uint256 tokenId_) public {
167      require(_rerollAddress != address(0), "ROUTER: NO_REROLL_ADDRESS");
168
169      address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
170      IProperty property = IProperty(propertyAddress);
171      IXtatuzReroll rerollContract = IXtatuzReroll(_rerollAddress);
172
173      address tokenAddress = rerollContract.tokenAddress();
174      string memory prevUri = property.tokenURI(tokenId_);
175      uint256 fee = rerollContract.rerollFee();
176      string[] memory rerollData = rerollContract.getRerollData(projectId_);
177      address tokenOwner = property.ownerOf(tokenId_);
178      require(tokenOwner == msg.sender, "ROUTER: NOT_NFT_OWNER");
179
180      uint256 newIndex = block.timestamp % rerollData.length;
181      property.setTokenURI(tokenId_, rerollData[newIndex]);
182      rerollData[newIndex] = prevUri;
183      rerollContract.setRerollData(projectId_, rerollData);

```

```
184     IERC20(tokenAddress).transferFrom(msg.sender, address(this), fee);
185     _totalRerollFee[projectId_] += fee;
186
187     emit NFTReroll(msg.sender, projectId_, tokenId_);
188 }
189 }
```

Listing 42.1 The *nftReroll* function

Recommendations

Since the *Property NFT* without rarity, the randomness that generates the **newIndex** is not harmful to the system.

We report this information for acknowledgement only, as we care about the fairness of the contract.

Reassessment

This issue was acknowledged by the *Xtatuz* team.

No. 43	Recommended Improving Event Emissions		
Risk	Informational	Likelihood	Low
Functionality is in use	In use	Impact	Low
Associated Files	contracts/XtatuzRouter.sol		
Locations	XtatuzRouter.sol L: 58, 264		

Detailed Issue

We noticed that the `setPropertyStatus` function could be **improved by emitting the previous status for improving traceability**.

XtatuzRouter.sol

```

58  event ChangePropertyStatus(uint256 indexed projectId, IProperty.PropertyStatus
      status);

      // (...SNIPPED...)

261  function setPropertyStatus(uint256 projectId_, IProperty.PropertyStatus status)
262  public onlySpv {
263      address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
264      IProperty(propertyAddress).setPropertyStatus(status);
265      emit ChangePropertyStatus(projectId_, status);
    }
```

Listing 43.1 The `ChangePropertyStatus` event that could improve

Recommendations

We recommend emitting the *previous status* on the *setPropertyStatus* function to improve transparency and traceability.

XtatuzRouter.sol

```
58 event ChangePropertyStatus(uint256 indexed projectId, IProperty.PropertyStatus
prevStatus, IProperty.PropertyStatus newStatus);

// (...SNIPPED...)

261 function setPropertyStatus(uint256 projectId_, IProperty.PropertyStatus status)
public onlySpv {
    address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
    IProperty.PropertyStatus prevStatus =
IProperty(propertyAddress).propertyStatus();
    IProperty(propertyAddress).setPropertyStatus(status);
    emit ChangePropertyStatus(projectId_, prevStatus, newStatus);
266 }
```

Listing 43.2 The improved *ChangePropertyStatus* event

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 44	Recommended Improving Transparency And Traceability Of Crucial Variables		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Partially Fixed
Associated Files	<i>contracts/Presaled.sol</i> <i>contracts/Property.sol</i> <i>contracts/XtatuzFactory.sol</i> <i>contracts/XtatuzProject.sol</i> <i>contracts/XtatuzReroll.sol</i> <i>contracts/XtatuzReferral.sol</i> <i>contracts/XtatuzRouter.sol</i>		
Locations	<i>Presaled.sol: L: 12, 13, and 14</i> <i>Property.sol: L: 27, 28, and 29</i> <i>XtatuzFactory.sol: L: 22, 23, and 24</i> <i>XtatuzProject.sol: L: 14, 15, 16, 19, 20, 25, 34, 39, and 40</i> <i>XtatuzReroll.sol: L: 16, and 17</i> <i>XtatuzReferral.sol: L: 25</i> <i>XtatuzRouter.sol: L: 20, 21, 22, 23, 24, 38, and 41</i>		

Detailed Issue

We found certain variables from various contracts are declared as **private** that necessitate enhanced transparency and traceability by changing their declaration to **public**. The mentioned variables are listed below:

- The **Presaled** contract
 - The `_operator` address variable.
 - The `_routerAddress` address variable.
 - The `baseURI` string variable.
- The **Property** contract
 - The `_operator` address variable.
 - The `_routerAddress` address variable.
 - The `baseURI` string variable.

- The **XtatuzFactory** contract
 - The `_propertyFactory` address variable.
 - The `_presaledFactory` address variable.
 - The `_projectFactory` address variable.
- The **XtatuzProject** contract
 - The `_operatorAddress` address variable.
 - The `_trusteeAddress` address variable.
 - The `_projectOwner` address variable.
 - The `_propertyAddress` address variable.
 - The `_presaledAddress` address variable.
 - The `_underwriteCount` unsigned-integer variable.
 - The `_isTriggeredEndpresale` boolean variable.
 - The `_multiSigMint` mapping.
 - The `_multiSigBurn` mapping.
- The **XtatuzReroll** contract
 - The `_operator` address variable.
 - The `_routerAddress` address variable.
- The **XtatuzReferral** contract
 - The `_operatorAddress` address variable.
- The **XtatuzRouter** contract
 - The `_spvAddress` address variable.
 - The `_xtatuzFactoryAddress` address variable.
 - The `_membershipAddress` address variable.
 - The `_rerollAddress` address variable.
 - The `_referralAddress` address variable.
 - The `_memberdProject` mapping.
 - The `_isNotice` mapping.

Recommendations

We recommend changing the declaration of the associated variables to **public**, in order to enhance transparency and traceability.

Reassessment

The *Xtatuz* team partially fixed this issue by enhancing the transparency and traceability of certain associated variables. However, some variables have been kept private.

No. 45	Recommended Enforcing Checks-Effects-Interactions Pattern		
Risk	Informational	Likelihood	Low
Functionality is in use	In use	Impact	Low
Associated Files	<i>contract/Presaled.sol</i> <i>contract/XtatuzRouter.sol</i>		
Locations	<i>Presaled.sol L: 43 - 55</i> <i>Router.sol L: 108 - 138, 140 - 149, 166 - 189 and 191 - 201</i>		

Detailed Issue

We noticed that the functions below ***do not follow the checks-effects-interactions pattern***, which is the best practice coding style to prevent potential reentrancy attacks.

List of functions that do not follow the checks-effects-interactions pattern.

- The ***mint*** function in the *Presaled* contract
- The ***addProjectMember*** function in the *XtatuzRouter* contract
- The ***claim*** function in the *XtatuzRouter* contract
- The ***nftReroll*** function in the *XtatuzRouter* contract
- The ***claimRerollFee*** function in the *XtatuzRouter* contract

Even if there are no reentrancy issues, we recommend that the list of functions above should enforce the *checks-effects-interactions* pattern.

For example, in the code snippet 45.1 below, the *mint* function transfers the presale token (*interactions part*) in L49 before updating the state variables (*effects part*) in L50 - 51.

Presaled.sol	
43	function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
44	require(to != address(0), "Presaled: Reciever address is address 0");
45	uint256 amount = tokenIdList_.length;
46	for (uint256 index = 0; index < amount; index++) {
47	uint256 tokenId = tokenIdList_[index];
48	require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
49	_safeMint(to, tokenId);
50	_mintedTimestamp[tokenId] = block.timestamp;
51	_tokenIdCount.increment();

```

52     }
53     setApprovalForAll(_operator, true);
54     emit Minted(to, tokenIdList_, tokenIdList_.length);
55 }
```

Listing 45.1 The *mint* function**XstatuzRouter.sol**

```

108 function addProjectMember(
109     uint256 projectId_,
110     uint256[] memory nftList_,
111     string memory referral_
112 ) public {
113     uint256 amount = nftList_.length;
114     address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
115     IXtatuzProject project = IXtatuzProject(projectAddress);
116     uint256 price = project.addProjectMember(msg.sender, nftList_);
117
118     uint256 minPrice = project.minPrice();
119     IXtatuzReferral referralContract = IXtatuzReferral(_referralAddress);
120     referralContract.increaseBuyerRef(projectId_, referral_, amount * minPrice);
121
122     address tokenAddress = project.tokenAddress();
123     IERC20(tokenAddress).transferFrom(msg.sender, projectAddress, price);
124
125     uint256[] memory memberedProject = _memberdProject[msg.sender];
126     bool foundedIndex;
127     for (uint256 index = 0; index < memberedProject.length; index++) {
128         if (memberedProject[index] == projectId_) {
129             foundedIndex = true;
130         }
131     }
132     if (!foundedIndex) {
133         _memberdProject[msg.sender].push(projectId_);
134     }
135
136     _isMemberClaimed[msg.sender][projectId_] = false;
137     emit AddProjectMember(projectId_, msg.sender, referral_, price);
138 }
```

Listing 45.2 The *addProjectMember* function**XstatuzRouter.sol**

```

140 function claim(uint256 projectId_) public {
141     require(_isMemberClaimed[msg.sender][projectId_] == false, "ROUTER:  
ALREADY CLAIMED");
142 }
```

```

143     address projectAddress = _xtatuzFactory.getProjectAddress(projectId_);
144     IXtatuzProject(projectAddress).claim(msg.sender);
145
146     _isMemberClaimed[msg.sender][projectId_] = true;
147
148     emit Claimed(projectId_, msg.sender);
149 }
```

Listing 45.3 The *claim* function**XstatuzRouter.sol**

```

166 function nftReroll(uint256 projectId_, uint256 tokenId_) public {
167   require(_rerollAddress != address(0), "ROUTER: NO_REROLL_ADDRESS");
168
169   address propertyAddress = _xtatuzFactory.getPropertyAddress(projectId_);
170   IProperty property = IProperty(propertyAddress);
171   IXtatuzReroll rerollContract = IXtatuzReroll(_rerollAddress);
172
173   address tokenAddress = rerollContract.tokenAddress();
174   string memory prevUri = property.tokenURI(tokenId_);
175   uint256 fee = rerollContract.rerollFee();
176   string[] memory rerollData = rerollContract.getRerollData(projectId_);
177   address tokenOwner = property.ownerOf(tokenId_);
178   require(tokenOwner == msg.sender, "ROUTER: NOT_NFT_OWNER");
179
180   uint256 newIndex = block.timestamp % rerollData.length;
181   property.setTokenURI(tokenId_, rerollData[newIndex]);
182   rerollData[newIndex] = prevUri;
183   rerollContract.setRerollData(projectId_, rerollData);
184
185   IERC20(tokenAddress).transferFrom(msg.sender, address(this), fee);
186   _totalRerollFee[projectId_] += fee;
187
188   emit NFTReroll(msg.sender, projectId_, tokenId_);
189 }
```

Listing 45.4 The *nftReroll* function**XstatuzRouter.sol**

```

191 function claimRerollFee(uint256 projectId_) public onlySrv {
192   require(_totalRerollFee[projectId_] > 0, "ROUTER: OUT_OF_FEE");
193
194   IXtatuzReroll rerollContract = IXtatuzReroll(_rerollAddress);
195   address tokenAddress = rerollContract.tokenAddress();
196
197   uint256 totalFee = _totalRerollFee[projectId_];
198   IERC20(tokenAddress).transfer(msg.sender, totalFee);
```

```

199     _totalRerollFee[projectId_] = 0;
200     emit ClaimedRerollFee(msg.sender, projectId_, totalFee);
201 }
```

Listing 45.5 The *claimRerollFee* contract

Recommendations

We recommend enforcing the checks-effects-interactions pattern to all of the functions below.

- The ***mint*** function in the *Presaled* contract
- The ***addProjectMember*** function in the *XtatuzRouter* contract
- The ***claim*** function in the *XtatuzRouter* contract
- The ***nftReroll*** function in the *XtatuzRouter* contract
- The ***claimRerollFee*** function in the *XtatuzRouter* contract

The example below is how to fix this issue, we moved the *interactions* part (the `_safeMint` function L51 in code snippet below) to get executed after the *effects* part (L49 - 50 in code snippet below).

Presaled.sol

```

43 function mint(address to, uint256[] memory tokenIdList_) public onlyOperator {
44     require(to != address(0), "Presaled: Reciever address is address 0");
45     uint256 amount = tokenIdList_.length;
46     for (uint256 index = 0; index < amount; index++) {
47         uint256 tokenId = tokenIdList_[index];
48         require(!_exists(tokenId) == false, "Presaled: TokenId already exists");
49         _mintedTimestamp[tokenId] = block.timestamp;
50         _tokenIdCount.increment();
51         _safeMint(to, tokenId);
52     }
53     setApprovalForAll(_operator, true);
54     emit Minted(to, tokenIdList_, tokenIdList_.length);
55 }
```

Listing 45.6 The improved *mint* function

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 46	Recommended Event Emissions For Transparency And Traceability		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	<code>contracts/Presaled.sol</code> <code>contracts/PresaledFactory.sol</code> <code>contracts/Property.sol</code> <code>contracts/PropertyFactory.sol</code> <code>contracts/ProjectFactory.sol</code> <code>contracts/XtatuzFactory.sol</code> <code>contracts/XtatuzReroll.sol</code> <code>contracts/XtatuzReferral.sol</code> <code>contracts/XtatuzProject.sol</code> <code>contracts/XtatuzRouter.sol</code>		
Locations	<i>Several functions throughout multiple contracts</i>		

Detailed Issue

We consider operations of the following state-changing functions important and require proper event emissions for improving transparency and traceability:

- The **Presaled** contract
 - The `setBaseURI` function
 - The `_setOperator` function
- The **PresaledFactory** contract
 - The `createPresale` function
- The **Property** contract
 - The `setPropertyStatus` function
 - The `_setOperator` function
 - The `setBaseURI` function
 - The `setTokenURI` function
- The **PropertyFactory** contract
 - The `createProperty` function

- The **ProjectFactory** contract
 - The *createProject* function
- The **XtatuzFactory** contract
 - The *createProjectContract* function
- The **XtatuzReroll** contract
 - The *setRerollData* function
 - The *setFee* function
- The **XtatuzReferral** contract
 - The *updateReferralAmount* function
 - The *claim* function
- The **XtatuzProject** contract
 - The *setPresalePeriod* function
 - The *setUnderwriteCount* function
 - The *claim* function
 - The *refund* function
 - The *extendEndPresale* function
 - The *ownerClaimLeft* function
 - The *multiSigMint* function
 - The *multiSigBurn* function
 - The *_initialData* function
- The **XtatuzRouter** contract
 - The *setRerollAddress* function
 - The *setMembershipAddress* function
 - The *setReferralAddress* function

Recommendations

We recommend **emitting relevant events** on the associated functions to improve transparency and traceability.

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 47	Recommended Removing Unused Code		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	<i>contract/Presaled.sol</i> <i>contract/Property.sol</i> <i>contract/PresaledFactory.sol</i> <i>contract/PropertyFactory.sol</i> <i>contract/XtatuzFactory.sol</i> <i>contract/XtatuzReroll.sol</i> <i>contract/XtatuzReferral.sol</i> <i>contract/XtatuzRouter.sol</i>		
Locations	<i>Presaled.sol L: 10</i> <i>Property.sol L: 8, 9</i> <i>PresaledFactory.sol L: 5</i> <i>PropertyFactory.sol L: 5</i> <i>XtatuzFactory.sol L: 4</i> <i>XtatuzReroll L: 4, 6, 8, 11, 15</i> <i>XtatuzReferral.sol L: 5, 6, 8, 10, 11 and 12</i> <i>XtatuzRouter.sol L38, 57, 98</i>		

Detailed Issue

We found that there were sections of unused code present in the smart contracts. This unused code could potentially cause confusion or misunderstandings among users or developers when attempting to maintain or modify the source code. Unused code can also increase the complexity of the codebase and lead to unnecessary computational overhead.

List of unused codes.

- Unused Imported Interface
 - ***IXtatuzProject, IPresaled*** interfaces (*L8 - 9 in Property.sol*)
 - ***IProperty*** interface (*L5 in PresaledFactory.sol*)
 - ***IProperty*** interface (*L5 in PropertyFactory.sol*)
 - ***IERC20, IProperty, IXtatuzRouter*** interfaces (*L4, 6, 8 in XtatuzReroll.sol*)
 - ***IERC721, IXtatuzFactory, IPresaled, IProperty, IXtatuzReroll*** interfaces (*L5, 8, 10, 11, 12 in XtatuzReferral.sol*)

- Unused Imported Source/Library
 - **XtatuzProject** contract (*L4 in XtatuzFactory.sol*)
 - **Counters** library (*L6 in XtatuzReferral.sol*)
- Unused Event
 - **Buyback** event (*L57 in XtatuzRouter.sol*)
- Unused State
 - **_tokenIdCount** state variable (*L10 in Presaled.sol*)
 - **_xtatusFactory, _totalFee** state variables (*L11, 15 in XtatuzReroll.sol*)
- Unused Modifier and Function
 - **endedPresale** modifier (*L85 - 88 in XtatuzProject.sol*)
 - **_checkEndedPresale** function (*L362 - 364 in XtatuzProject.sol*)

Example of unused imported interfaces in the *Property* contract (L8 - 9 in code snippet below).

Property.sol

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.17;
3
4 import "@openzeppelin/contracts/access/Ownable.sol";
5 import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
6 import "@openzeppelin/contracts/utils/Strings.sol";
7 import "../interfaces/IProperty.sol";
8 import "../interfaces/IXtatuzProject.sol";
9 import "../interfaces/IPresaled.sol";
10
11 contract Property is ERC721Enumerable, Ownable {
12     // (...SNIPPED...)
145 }
```

Listing 47.1 The *Property* contract

Recommendations

We recommend removing unused code from the smart contracts as it can reduce the contract's complexity and also help to reduce confusion among users or developers when maintaining the source code.

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 48	Inconsistent Error Message With The Code		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	contracts/XtatuzProject.sol contracts/Property.sol		
Locations	XtatuzProject.sol L: 359 Property.sol L:71		

Detailed Issue

We found an error message inconsistent with the code in the functions `_checkIsAvailable` (L359 in code snippet 48.1) and `mintFragment` (L71 in code snippet 48.2). These inconsistencies can lead to misunderstanding among users or developers when maintaining the source code.

XtatuzProject.sol

```

357 function _checkIsAvailable() internal view {
358     uint256 timestamp = block.timestamp;
359     require(timestamp >= startPresale && timestamp <= endPresale, "PROJECT:
360 ENDED_PRESALE");
    }
```

Listing 48.1 The `_checkIsAvailable` function with an inconsistent error message

Property.sol

```

65 function mintFragment(address to, uint256[] memory tokenIdList) public
onlyOperator {
66     require(ownerOf(0) == address(this), "PROPERTY: NO_MASTER_NFT");
67     require(isMintedMaster == true, "PROPERTY: MASTER_NOT_MINTED");
68     uint256 amount = tokenIdList.length;
69     for (uint256 index = 0; index < amount; index++) {
70         uint256 tokenId = tokenIdList[index];
71         require(_exists(tokenId) == false, "PROJECT: ALREADY_EXISTS");
72         _safeMint(to, tokenId);
73     }
74     _setApprovalForAll(to, _routerAddress, true);
75     emit MintFragment(to, tokenIdList);
76 }
```

Listing 48.2 The *mintFragment* function with an inconsistent error message

Recommendations

We recommend revising the associated error message to reflect the actual code.

XtatuzProject.sol

```

357   function _checkIsAvailable() internal view {
358     uint256 timestamp = block.timestamp;
359     require(timestamp >= startPresale && timestamp <= endPresale, "PROJECT:
360     UNAVAILABLE");
360 }
```

Listing 48.3 The improved *_checkIsAvailable* function

Property.sol

```

65   function mintFragment(address to, uint256[] memory tokenIdList) public
66   onlyOperator {
67     require(ownerOf(0) == address(this), "PROPERTY: NO_MASTER_NFT");
68     require(isMintedMaster == true, "PROPERTY: MASTER_NOT_MINTED");
69     uint256 amount = tokenIdList.length;
70     for (uint256 index = 0; index < amount; index++) {
71       uint256 tokenId = tokenIdList[index];
72       require(!_exists(tokenId) == false, "PROPERTY: TOKEN_ALREADY_EXISTS");
73       _safeMint(to, tokenId);
74     }
75     _setApprovalForAll(to, _routerAddress, true);
76     emit MintFragment(to, tokenIdList);
76 }
```

Listing 48.4 The improved *mintFragment* function

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

Reassessment

The Xtatuz team adopted our recommended code to fix this issue.

No. 49	Misspelling State Variable		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	<i>contracts/XatuzRouter.sol</i>		
Locations	<i>XatuzRouter.sol L: 38</i>		

Detailed Issue

In the code snippet below, we found the misspelling crucial state variable at line 38. This misspelling can lead to misunderstanding among users or developers when maintaining the source code.

XatuzRouter.sol

```

14 contract XtatuzRouter {
    ...
38     mapping(address => uint256[]) private _memberdProject;

```

Listing 49.1 The misspell state variable

Recommendations

We recommend revising the misspelling state and associated function.

XatuzRouter.sol

```

14 contract XtatuzRouter {
    ...
38     mapping(address => uint256[]) private _memberedProject;

```

Listing 49.2 The improved state variable

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

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

No. 50	Inconsistent Interface With The Implementation		
Risk	Informational	Likelihood	Low
Functionality is in use	In use	Impact	Low
Associated Files	<i>interfaces/IPresaled.sol</i> <i>interfaces/IProperty.sol</i> <i>interfaces/IXtatuzRouter.sol</i>		
Locations	<i>IPresaled.sol L: 15</i> <i>IProperty.sol L: 36</i> <i>IXtatuzRouter.sol L: 47 and 51</i>		

Detailed Issue

We noticed the significant inconsistency between the functions listed in the interfaces of the following contracts and their contract implementation. This could result in compatibility issues with other contracts.

- The ***IPresaled*** interface
 - The *getPresaledPackage* function
- The ***IProperty*** interface
 - The *rerollData* function
- The ***IXtatuzRouter*** interface
 - The *referralAmount* function.
 - The *getMembershipAddress* function

Recommendations

We recommend reviewing and updating the associated interfaces to accurately reflect the implemented functions.

Reassessment

The Xtatuz team fixed this issue as per our suggestion.

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 the 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