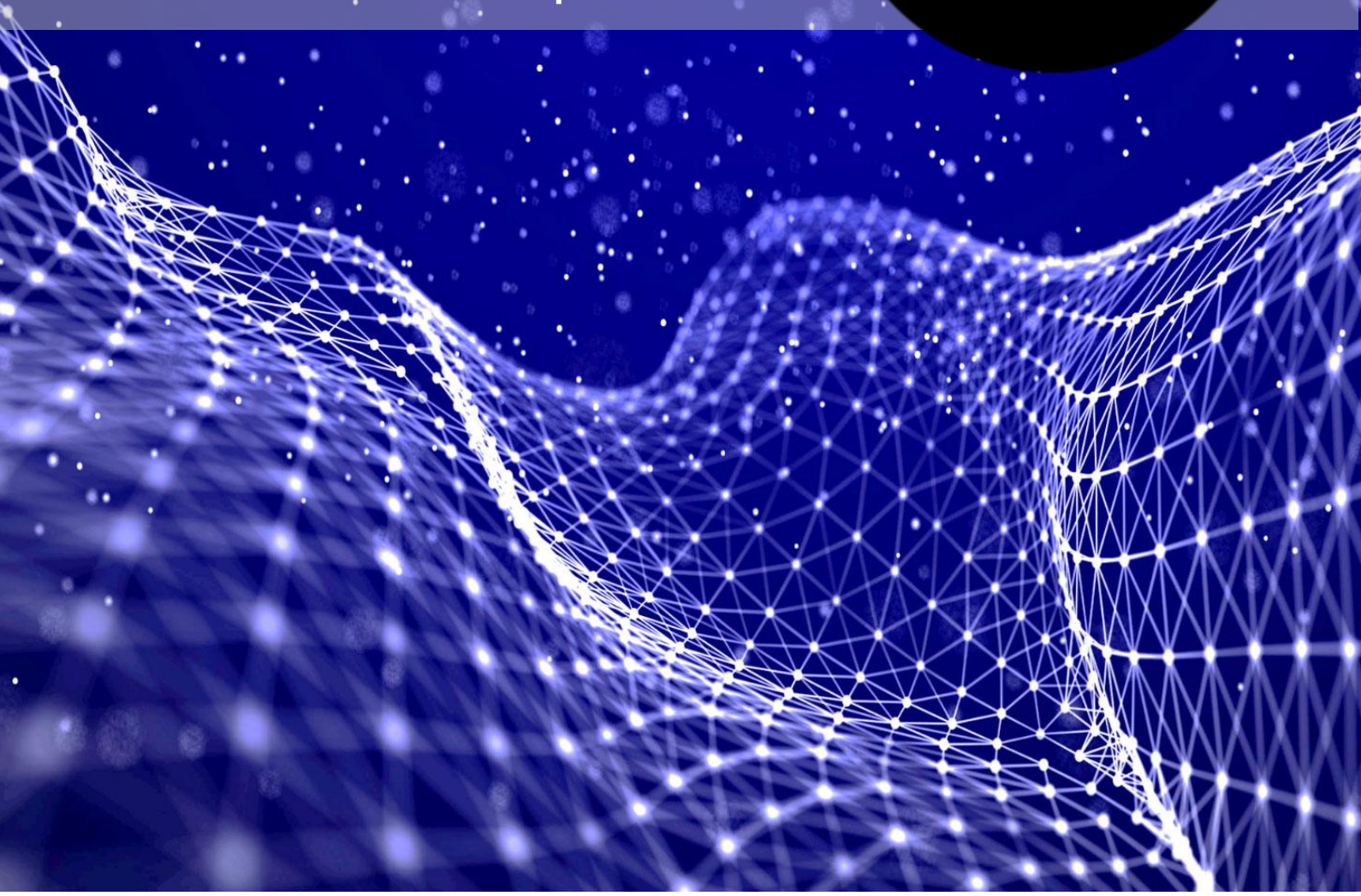
The Bigbang Theory

Smart Contract (Token & NFT)

Smart Contract Audit Report



Date Issued: 5 Sep 2023

Version: Final v1.0

ValiX Consulting

Public

Table of Contents

Executive Summary

Overview

Valix conducted a smart contract audit to evaluate potential security issues of the **Smart Contract (Token & NFT)**. This audit report was published on ***5 sep 2023***. The audit scope is limited to the **Smart Contract (Token & NFT).** Our security best practices strongly recommend that the **Big Bang Theory 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 Smart Contract (Token & NFT)

Big Bang Theory is the World's first Metaverse (Infrastructure) as a service. We are a metaverse builder tool where you can create and customize your own virtual world within 10 minutes and also integrate with business functions that support over 30 functions and also support Token & NFT on web3 without knowledge of smart contracts.

Scope of Work

The security audit conducted does not replace the full security audit of the overall **Big Bang Theory** protocol. The scope is limited to the **Smart Contract (Token & NFT)** and their related smart contracts.

The security audit covered the components at this specific state:

Item	Description
Components	▪ *BigBangTheoryNFTV2, BigBangTheoryTokenV2, and BigBangTheoryTokenUnlimitV1* ▪ *Imported associated smart contracts and libraries*
Git Repository	▪ *https://git.multiverseexpert.io/bigbangtheory/web3/smart-contract*
Audit Commit	▪ *bb4ed3a0dc6b2076de5644f7ea6835cea6d0db0c (branch: main)*
Certified Commit	▪ *f3f502f504570724f99b453ea9b75d904bc99210 (branch: main)*

Audited Files	▪ *contracts/BBTNFT_V2.sol* ▪ *contracts/BBTToken_V2.sol* ▪ *contracts/BBTTokenUnlimit_V1.sol* ▪ *contracts/BBT/*.sol* ▪ *Other imported associated Solidity files*
Excluded Files/Contracts	▪ -

Remark: Our security best practices strongly recommend that the Big Bang Theory 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	**Kritsada Dechawattana (Lead Auditor)** **Anak Mirasing** **Parichaya Thanawuthikrai**
Authors	**Anak Mirasing** **Kritsada Dechawattana** **Parichaya Thanawuthikrai**
Reviewers	**Sumedt Jitpukdebodin**

Disclaimer

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

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

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

Audit Result Summary

From the audit results and the remediation and response from the developer, Valix trusts that the **Smart Contract (Token & NFT)** has sufficient security protections to be safe for use.

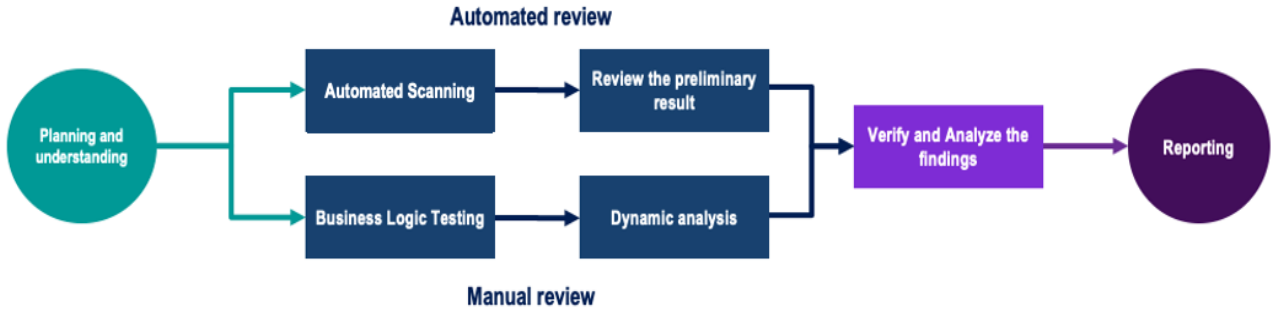
Initially, Valix was able to identify **10 issues** that were categorized from the "Critical" to "Informational" risk level in the given timeframe of the assessment. **Of these, the team was able to completely fix 9 issues, and acknowledge 1 issue**. Below is the breakdown of the vulnerabilities found and their associated risk rating for each assessment conducted.

Target	Assessment Result					Reassessment Result				
	C	H	M	L	I	C	H	M	L	I
Smart Contract (Token & NFT)	-	-	3	2	5	-	-	1	0	0

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

Methodology

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



Planning and Understanding

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

Automated Review

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

Manual Review

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

Reporting

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

Audit Items

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

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

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

Risk Rating

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

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

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

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

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

System Trust Assumptions

Trust assumptions

The trust assumption in this context is that the Big Bang Theory protocol allows the trusted operator of the BigBangTheoryNFTV2, BigBangTheoryTokenV2, and BigBangTheoryTokenUnlimitV1 contracts to perform actions to control the BBT721 or BBT20 tokens.

It's important to note that, while the trusted operator is granted specific privileges to oversee the BBT721 or BBT20 tokens within the Big Bang Theory protocol, special attention should be given to the _adminContract mapping account. This account possesses the authority to transfer or burn tokens across the Big Bang Theory protocol.

Furthermore, the trusted operator can execute actions without the need for a time-lock mechanism. This implies that any action within the scope of the trusted operator's authority will be carried out promptly.

The privileged roles

In the **Big Bang Theory** protocol, privileged roles have special access to perform sensitive actions, relying on the trust placed in these roles to ensure the proper functioning and security of the system.

The **BigBangTheoryNFTV2** contract:

- The **DEFAULT_ADMIN_ROLE** account:
  - Can grant new roles and manage existing roles
  - Can set the *_adminContract* mapping account using the *adminContractSet* function
- The **URI_ROLE** account:
  - Can set the base URI
- The **PAUSER_ROLE** account:
  - Can pause/unpause mint and transfer operations
- The **MINTER_ROLE** account:
  - Can mint NFTs using the *safeMint* function
- The **LOCKED_ROLE** account:
  - Can lock/unlock NFTs using the *lockNFT* and *unlockNFT* functions
- The **_adminContract** mapping account:
  - Can transfer NFTs using the *transferFrom* and *safeTransferFrom* functions
  - Can burn NFTs using the *burn* function

The **BigBangTheoryTokenV2** contract:

- The **DEFAULT_ADMIN_ROLE** account:
  - Can grant new roles and manage existing roles
  - Can set the *_adminContract* mapping account using the *adminContractSet* function
- The **PAUSER_ROLE** account:
  - Can pause/unpause mint and transfer operations
- The **MINTER_ROLE** account:
  - Can mint tokens using the *mint* function
- The **_adminContract** mapping account:
  - Can transfer tokens using the *transferFrom* function
  - Can burn tokens using the *burnFrom* function

The **BigBangTheoryTokenUnlimitV1** contract:

- The **DEFAULT_ADMIN_ROLE** account:
  - Can grant new roles and manage existing roles
- The **ADMIN_ROLE** account:
  - Can set the *_adminContract* mapping account using the *adminContractSet* function
- The **PAUSER_ROLE** account:
  - Can pause/unpause mint and transfer operations
- The **MINTER_ROLE** account:
  - Can mint tokens using the *mint* function
- The **_adminContract** mapping account:
  - Can transfer tokens using the *transferFrom* function
  - Can burn tokens using the *burnFrom* function

Review Findings Summary

The table below shows the summary of our assessments.

No.	Issue	Risk	Status	Functionality is in use
1	Potential Asset Loss By Compromised Administrator	Medium	Acknowledged	In use
2	Permanently Losing The Admin Role Through The renounceRole Function	Medium	Fixed	In use
3	Permanently Losing The Admin Role Through The revokeRole Function	Medium	Fixed	In use
4	Locking Unminted Tokens Causes Inability To Mint New Tokens	Low	Fixed	In use
5	Compiler Is Not Locked To Specific Version	Low	Fixed	In use
6	Compiler May Be Susceptible To Publicly Disclosed Bugs	Informational	Fixed	In use
7	Recommended Removing Unused Code	Informational	Fixed	In use
8	Inappropriate Role Authorization For The setBaseURI Function	Informational	Fixed	In use
9	Inconsistent Error Message With The Code	Informational	Fixed	In use
10	Recommended Event Emissions For Transparency And Traceability	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	Potential Asset Loss By Compromised Administrator		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Acknowledged
Associated Files	*contracts/BBT/BBT721.sol* *contracts/BBT/BBT20.sol*		
Locations	*BBT721.sol L: 272 - 281* *BBT20.sol L: 382 - 399*		

Detailed Issue

The *_isApprovedOrOwner* function of the *BBT721* contract is used to check whether the caller is the owner or spender (approved by the owner) that has the right to manage the given token ID (e.g., transfer, burn).

Besides the **owner** and **spender**, we noticed that the ***_adminContract*** mapping (L32 in code snippet 1.1) of the *BBT721* contract is used to store the admin contract list and could update via the *_adminContractSet* function (L383 - 396 in code snippet 1.1), which also has the permission to manage the given token ID (L280 in code snippet 1.1).

However, let's consider the case that one of the admin contracts has been compromised. An attacker can steal all assets on the platform suddenly.

A similar issue occurs in the *spendAllowance* function within the *BBT20* contract (code snippet 1.2).

BBT721.sol

```
32  mapping(address => bool) private _adminContract;

    // (...SNIPPED...)

272 function _isApprovedOrOwner(
273     address spender,
274     uint256 tokenId
275 ) internal view virtual returns (bool) {
276     address owner = BBT721.ownerOf(tokenId);
```

```
277     return
278         (spender == owner ||
279             isApprovedForAll(owner, spender) ||
280             getApproved(tokenId) == spender) || _adminContract[msg.sender];
281 }

    // (...SNIPPED...)

383 function _adminContractSet(
384     address _address,
385     bool _status
386 ) internal virtual {
387     require(
388         _address != address(0),
389         "BBT721: set admin contract to zero address"
390     );
391     require(
392         _adminContract[_address] != _status,
393         "BBT721: set admin contract status is invalid"
394     );
395     _adminContract[_address] = _status;
396 }
```

Listing 1.1 The *_adminContract* mapping of the *BBT721* contract which also has the permission to manage the given token ID

BBT20.sol

```
 42 mapping(address => bool) private _adminContract;

    // (...SNIPPED...)

334 function _adminContractSet(
335     address _address,
336     bool _status
337 ) internal virtual {
338     require(
339         _address != address(0),
340         "BBT20: set admin contract to zero address"
341     );
342     require(
343         _adminContract[_address] != _status,
344         "BBT20: set admin contract status is invalid"
345     );
346     _adminContract[_address] = _status;
347 }

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

```
382 function _spendAllowance(
383     address owner,
384     address spender,
385     uint256 amount
386 ) internal virtual {
387     if (!_adminContract[msg.sender]) {
388         uint256 currentAllowance = allowance(owner, spender);
389         if (currentAllowance != type(uint256).max) {
390             require(
391                 currentAllowance >= amount,
392                 "BBT20: insufficient allowance"
393             );
394             unchecked {
395                 _approve(owner, spender, currentAllowance - amount);
396             }
397         }
398     }
399 }
```

Listing 1.2 The *_adminContract* mapping of the *BBT20* contract which also has the permission to manage tokens on behalf of the owner

Recommendations

We cannot recommend code without breaking the contract functionalities. However, we recommend the *Big Bang Theory* team uses the *Multisig* and/or introduce *Timelock* for the admin contract to improve transparency and prepare the risk mitigation plan to protect user assets from this issue.

Moreover, the team has the flexibility to adopt and adapt these recommendations based on their specific business requirements.

Reassessment

The *Big Bang Theory* team has acknowledged this issue since the admin contract role is designed for use in emergency situations that may require asset retrieval. Therefore, The *Big Bang Theory* team has intended to maintain the existing code as per the defined business model.

No. 2	Permanently Losing The Admin Role Through The renounceRole Function		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBT/access/AccessControl.sol*		
Locations	*AccessControl.sol L: 194 - 204*		

Detailed Issue

The *BigBangTheoryNFTV2*, *BigBangTheoryTokenV2,* and *BigBangTheoryTokenUnlimitV1* contracts derive the *renounceRole* function from the *AccessControl* contract (L194 - 204 in code snippet 2.1). This function allows anyone to remove their specific role.

We consider the *renounceRole* function risky since it can remove privileged roles, including the *DEFAULT_ADMIN_ROLE*, which is the top-level role. Consider that the only account with the *DEFAULT_ADMIN_ROLE* role is removed by mistakenly calling the *renounceRole* function.

The *BigBangTheoryNFTV2*, *BigBangTheoryTokenV2,* and *BigBangTheoryTokenUnlimitV1* contracts will be dangled immediately since the contract will have no account with the *DEFAULT_ADMIN_ROLE* role anymore, and this is unrecoverable.

AccessControl.sol

```
194 function renounceRole(
195     bytes32 role,
196     address account
197 ) public virtual override {
198     require(
199         account == _msgSender(),
200         "AccessControl: can only renounce roles for self"
201     );
202
203     _revokeRole(role, account);
204 }


    // (...SNIPPED...)

262 function _revokeRole(bytes32 role, address account) internal virtual {
```

```
263     if (hasRole(role, account)) {
264         _roles[role].members[account] = false;
265         emit RoleRevoked(role, account, _msgSender());
266     }
267 }
```

Listing 2.1 The *renounceRole* function of the *AccessControl* contract

Recommendations

We recommend overriding and implementing the *renounceRole* function to the *BigBangTheoryNFTV2*, *BigBangTheoryTokenV2,* and *BigBangTheoryTokenUnlimitV1* contracts as the following code snippet.

This can avoid the case that the sole account with the *DEFAULT_ADMIN_ROLE* role is removed accidentally.

BBTNFT_V2.sol

```
130 function renounceRole(bytes32 role, address account) public virtual
131 override(AccessControl) {
132     require(role != DEFAULT_ADMIN_ROLE , "AccessControl: cannot renounce the
    DEFAULT_ADMIN_ROLE account");
133
134     super.renounceRole(role, account);
135 }
```

Listing 2.2 The example overridden *renounceRole* function of the *BigBangTheoryNFTV2* contract

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

Reassessment

The *Big Bang Theory* team has fixed this issue by deciding to apply the recommendation condition checks directly into the ***renounceRole*** function in the ***AccessControl*** contract, which is the base contract.

However, direct code modifications within a base contract can potentially lead to complications when future updates are introduced. We recommend thorough testing and documentation to ensure compatibility and maintainability in the long term.

AccessControl.sol

```
199 function renounceRole(
200     bytes32 role,
201     address account
202 ) public virtual override {
203     require(
204         role != DEFAULT_ADMIN_ROLE,
205         "AccessControl: cannot renounce the DEFAULT_ADMIN_ROLE account"
206     );
207     require(
208         account == _msgSender(),
209         "AccessControl: can only renounce roles for self"
210     );
211 
212     _revokeRole(role, account);
213 }
```

Listing 2.3 The updating of the *renounceRole* function of the *AccessControl* contract

No. 3	Permanently Losing The Admin Role Through The revokeRole Function		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBT/access/AccessControl.sol*		
Locations	*AccessControl.sol L: 171 - 176*		

Detailed Issue

The *BigBangTheoryNFTV2*, *BigBangTheoryTokenV2*, and *BigBangTheoryTokenUnlimitV1* contracts derive the *revokeRole* function function from the *AccessControl* contract (L171 - 176 in code snippet 3.1). This function permits solely the *DEFAULT_ADMIN_ROLE* account(s) to revoke roles from specific accounts.

We consider the *revokeRole* function risky since it can remove privileged roles, including the *DEFAULT_ADMIN_ROLE*, which is the top-level role. Consider that the only account with the *DEFAULT_ADMIN_ROLE* role is removed by mistakenly calling the *revokeRole* function.

The *BigBangTheoryNFTV2*, *BigBangTheoryTokenV2*, and *BigBangTheoryTokenUnlimitV1* contracts will be dangled immediately since the contract will have no account with the *DEFAULT_ADMIN_ROLE* role anymore, and this is unrecoverable.

AccessControl.sol

```
171 function revokeRole(
172     bytes32 role,
173     address account
174 ) public virtual override onlyRole(getRoleAdmin(role)) {
175     _revokeRole(role, account);
176 }

    // (...SNIPPED...)

262 function _revokeRole(bytes32 role, address account) internal virtual {
263     if (hasRole(role, account)) {
264         _roles[role].members[account] = false;
265         emit RoleRevoked(role, account, _msgSender());
266     }
267 }
```

Listing 3.1 The *revokeRole* function of the *AccessControl* contract

Recommendations

We recommend overriding and implementing the *revokeRole* function to the *BigBangTheoryNFTV2*, *BigBangTheoryTokenV2,* and *BigBangTheoryTokenUnlimitV1* contracts as the following code snippet.

The recommended code modification would prevent the *DEFAULT_ADMIN_ROLE* account from self-revoking, thereby, this can avoid the case that the sole account with the *DEFAULT_ADMIN_ROLE* role is removed accidentally.

BBTNFT_V2.sol

```
130 function revokeRole(bytes32 role, address account) public virtual
    override(AccessControl) {
131     require(
132         role != DEFAULT_ADMIN_ROLE || _msgSender() != account,
133         "AccessControl: the DEFAULT_ADMIN_ROLE account cannot self-revoke"
134     );
135
136     super.revokeRole(role, account);
137 }
```

Listing 3.2 The example overridden *revokeRole* function of the *BigBangTheoryNFTV2* contract

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

Reassessment

The *Big Bang Theory* team has fixed this issue by deciding to apply the recommendation condition checks directly into the ***revokeRole*** function in the ***AccessControl*** contract, which is the base contract.

However, direct code modifications within a base contract can potentially lead to complications when future updates are introduced. We recommend thorough testing and documentation to ensure compatibility and maintainability in the long term.

AccessControl.sol

```
171 function revokeRole(
172     bytes32 role,
173     address account
174 ) public virtual override onlyRole(getRoleAdmin(role)) {
175     require(
176         role != DEFAULT_ADMIN_ROLE || _msgSender() != account,
177         "AccessControl: the DEFAULT_ADMIN_ROLE account cannot self-revoke"
178     );
179
```

```
180         _revokeRole(role, account);
181     }
```

Listing 3.3 The updating of the *revokeRole* function of the *AccessControl* contract

No. 4	Locking Unminted Tokens Causes Inability To Mint New Tokens		
Risk	Low	Likelihood	Low
		Impact	Medium
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBTNFT_V2.sol* *contracts/BBT/BBT721.sol*		
Locations	*BBTNFT_V2.sol L: 91 - 94, L: 108 - 116* *BBT721.sol L: 325 - 347*		

Detailed Issue

The *lockNFT* function within the *BigBangTheoryNFTV2* contract allows the *LOCKED_ROLE* account to lock NFTs by providing the corresponding token ID. Upon execution, this function updates the *isLockeds* mapping state to *true* (L93 in code snippet 4.1), effectively preventing the mint, transfer or burning of the associated token that will be checked in the *_beforeTokenTransfer* function (L114 in code snippet 4.1).

However, an issue has been identified when the *LOCKED_ROLE* account mistakenly passes a token ID that hasn't been minted yet to the *lockNFT* function. This action incorrectly flags a non-existent NFT token as locked.

Consequently, a problem occurs when the *MINTER_ROLE* account attempts to mint an NFT token using the same ID that has already been locked, as the minting operations also trigger the *_beforeTokenTransfer* function (L329 in code snippet 4.2). This results in a transaction revert, preventing the successful minting of the NFT token.

BBTNFT_V2.sol

```
91  function lockNFT(uint256 _tokenId) public onlyRole(LOCKED_ROLE) {
92      require(!isLockeds[_tokenId], "This token is locked");
93      isLockeds[_tokenId] = true;
94  }

    // (...SNIPPED...)

108 function _beforeTokenTransfer(
109       address from,
110       address to,
111       uint256 tokenId,
112       uint256 batchSize
```

```
113 ) internal override(BBT721, BBT721Enumerable) whenNotPaused {
114     require(!isLockeds[tokenId], "This token is locked");
115     super._beforeTokenTransfer(from, to, tokenId, batchSize);
116 }
```

Listing 4.1 The *lockNFT* and *_beforeTokenTransfer* functions of the *BigBangTheoryNFTV2* contract

BBT721.sol

```
325 function _mint(address to, uint256 tokenId) internal virtual {
326     require(to != address(0), "BBT721: mint to the zero address");
327     require(!_exists(tokenId), "BBT721: token already minted");
328 
329     _beforeTokenTransfer(address(0), to, tokenId, 1);
330 
331     // Check that tokenId was not minted by `_beforeTokenTransfer` hook
332     require(!_exists(tokenId), "BBT721: token already minted");
333 
334     unchecked {
335         // Will not overflow unless all 2**256 token ids are minted to the same
    owner.
336         // Given that tokens are minted one by one, it is impossible in practice
    that
337         // this ever happens. Might change if we allow batch minting.
338         // The BBT fails to describe this case.
339         _balances[to] += 1;
340     }
341 
342     _owners[tokenId] = to;
343 
344     emit Transfer(address(0), to, tokenId);
345 
346     _afterTokenTransfer(address(0), to, tokenId, 1);
347 }
```

Listing 4.2 The *_mint* function of the *BBT721* contract

Recommendations

We suggest enhancing the *lockNFT* function by incorporating the existing *_requireMinted* utility function from the *BBT721* contract. This addition would ensure that only minted tokens can be locked, preventing the mistakenly locking of non-existent tokens.

BBTNFT_V2.sol

```
91  function lockNFT(uint256 _tokenId) public onlyRole(LOCKED_ROLE) {
92      _requireMinted(_tokenId);
93      require(!isLockeds[_tokenId], "This token is locked");
94      isLockeds[_tokenId] = true;
95  }
```

Listing 4.3 The improved *lockNFT* function of the *BigBangTheoryNFTV2* contract

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

Reassessment

The *Big Bang Theory* team adopted our recommended code to fix this issue.

No. 5	Compiler Is Not Locked To Specific Version		
Risk	Low	Likelihood	Low
		Impact	Medium
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBTNFT_V2.sol* *contracts/BBTToken_V2.sol* *contracts/BBTTokenUnlimit_V1.sol*		
Locations	*BBTNFT_V2.sol L: 2* *BBTToken_V2.sol L: 2* *BBTTokenUnlimit_V1.sol L: 2*		

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 contract files that should lock to the specific version.

- ***BBTNFT_V2.sol***
- ***BBTToken_V2.sol***
- ***BBTTokenUnlimit_V1.sol***

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

BBTNFT_V2.sol

```
1 // SPDX-License-Identifier: MEX
2 pragma solidity ^0.8.4;
```

Listing 5.1 The *BBT721* contract

Recommendations

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

pragma solidity 0.8.19;
// or
pragma solidity =0.8.19;

Reference: *https://swcregistry.io/docs/SWC-103*

Reassessment

The *Big Bang Theory* team adopted our recommended code to fix this issue.

No. 6	Compiler May Be Susceptible To Publicly Disclosed Bugs		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBTNFT_V2.sol* *contracts/BBTToken_V2.sol* *contracts/BBTTokenUnlimit_V1.sol*		
Locations	*BBTNFT_V2.sol L: 2* *BBTToken_V2.sol L: 2* *BBTTokenUnlimit_V1.sol L: 2*		

Detailed Issue

The ***BBTNFT_V2***, ***BBTToken_V2***, and ***BBTTokenUnlimit_V1*** contract files use an outdated *Solidity* compiler version (v0.8.4) which may be susceptible to publicly disclosed vulnerabilities.

Most of the chains currently support *Solidity* version up to *0.8.19*, which contains the list of known bugs as the following link:

https://docs.soliditylang.org/en/v0.8.19/bugs.html

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

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

MintHelper.sol

```
1 // SPDX-License-Identifier: MEX
2 pragma solidity ^0.8.4;
```

Listing 6.1 An example smart contract that does not use the latest compatible patch version (v0.8.19)

Recommendations

We recommend using the latest compatible patch version, v0.8.19, that fixes all known bugs.

Reassessment

The *Big Bang Theory* team adopted our recommended code to fix this issue.

No. 7	Recommended Removing Unused Code		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBTNFT_V2.sol* *contracts/BBT/BBT20.sol*		
Locations	*BBTNFT_V2.sol L: 28* *BBT20.sol L: 108 - 113*		

Detailed Issue

We found unused code with the *BigBangTheoryNFTV2* and *BBT20* contracts. These codes are as follows:

- In the *BigBangTheoryNFTV2* contract, there is an inclusion of the *UNLOCK_ROLE*; however, this role is solely introduced and utilized during the contract's construction and is not employed in the rest of the implementation (as shown in code snippet 7.1)
- In the *BBT20* contract, there exists an inactive *_checkTransfer* function. It is indicated as a comment section of code (as shown in code snippet 7.2)

These unused codes could potentially cause confusion or misunderstandings among users or developers when attempting to maintain or modify the source code.

BBTNFT_V2.sol

```
28  bytes32 public constant UNLOCK_ROLE = keccak256("UNLOCK_ROLE");

    // (...SNIPPED...)

31  constructor(
32      string memory _name,
33      string memory _symbol
34  ) BBT721(_name, _symbol) {
35      _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
36      _grantRole(PAUSER_ROLE, msg.sender);
37      _grantRole(MINTER_ROLE, msg.sender);
38      _grantRole(LOCKED_ROLE, msg.sender);
39      _grantRole(UNLOCK_ROLE, msg.sender);
40  }
```

Listing 7.1 The unused code of the *BigBangTheoryNFTV2* contract

BBT20.sol

```
108 // function _checkTransfer(uint tokenId, address from) internal virtual {
109 //     require(
110 //         BBT721.ownerOf(tokenId) == from || _adminContract[msg.sender],
111 //         "BBT721: transfer from incorrect owner"
112 //     );
113 // }
```

Listing 7.2 The unused code of the *BBT20* 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 *Big Bang Theory* team adopted our recommended code to fix this issue.

No. 8	Inappropriate Role Authorization For The setBaseURI Function		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBTNFT_V2.sol*		
Locations	*BBTNFT_V2.sol L: 55 - 57*		

Detailed Issue

The *setBaseURI* function is designed within the *BigBangTheoryNFTV2* contract to facilitate the adjustment of a base URI.

However, the authorization of the *setBaseURI* function is restricted access to the *PAUSER_ROLE* account that might not be appropriately suited for the intended purpose of this function. The use of the *PAUSER_ROLE* to control access to the *setBaseURI* function may not align with its actual functionality.

In example, the *pause* and *unpause* functions are appropriately authorized with *onlyRole(PAUSER_ROLE)* (L59 and L63 in the code snippet below). These functions are restricted to only the *PAUSER_ROLE* account to manage the pausing and unpausing of the contract. The use of the *PAUSER_ROLE* for this case is appropriate and consistent with their intended functionalities.

As a result, the role authorization inappropriate could lead to confusion and mismanagement of function access.

BBTNFT_V2.sol

```
55 function setBaseURI(string memory _uri) public onlyRole(PAUSER_ROLE) {
56     uri = _uri;
57 }
58
59 function pause() public onlyRole(PAUSER_ROLE) {
60     _pause();
61 }
62
63 function unpause() public onlyRole(PAUSER_ROLE) {
64     _unpause();
65 }
```

Listing 8.1 The *setBaseURI* function of the *BigBangTheoryNFTV2* contract

Recommendations

We recommend updating the *setBaseURI* function's role authorization to *DEFAULT_ADMIN_ROLE* or the appropriate role to ensure the appropriate alignment between function purpose and authorized role.

BBTNFT_V2.sol

```
55 function setBaseURI(string memory _uri) public onlyRole(DEFAULT_ADMIN_ROLE) {
56     uri = _uri;
57 }
```

Listing 8.2 The improved *setBaseURI* function of the *BigBangTheoryNFTV2* contract

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

Reassessment

The *Big Bang Theory* team adopted our recommendation by introducing and applying a new role named ***URI_ROLE*** to handle updating token URI through the ***setBaseURI*** function.

No. 9	Inconsistent Error Message With The Code		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBTNFT_V2.sol*		
Locations	*BBTNFT_V2.sol L: 96 - 99*		

Detailed Issue

The *unlockNFT* function of the *BigBangTheoryNFTV2* contract contains an inconsistent error message displayed to users that does not correspond accurately with the actual statement as shown in the code snippet below.

This inconsistency between error messages and code behavior could result in inefficiencies, confusion, and difficulty in debugging issues.

BBTNFT_V2.sol

```
96  function unlockNFT(uint256 _tokenId) public onlyRole(LOCKED_ROLE) {
97      require(isLockeds[_tokenId], "This token is locked");
98      isLockeds[_tokenId] = false;
99  }
```

Listing 9.1 The inconsistent error message of the *unlockNFT* function

Recommendations

We recommend changing the error message of the *unlockNFT* function to align with the actual statement as follows. This consistency in error messages will contribute to enhancing code comprehension, and facilitate effective issue resolution.

BBTNFT_V2.sol

```
96  function unlockNFT(uint256 _tokenId) public onlyRole(LOCKED_ROLE) {
97      require(isLockeds[_tokenId], "This token is unlocked");
98      isLockeds[_tokenId] = false;
99  }
```

Listing 9.2 The improved error message of the *unlockNFT* function

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

Reassessment

The *Big Bang Theory* team adopted our recommended code to fix this issue.

No. 10	Recommended Event Emissions For Transparency And Traceability		
Risk	Informational	Likelihood	Low
		Impact	Low
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/BBTNFT_V2.sol* *contracts/BBTToken_V2.sol* *contracts/BBTTokenUnlimit_V1.sol* *contracts/BBT/BBT20.sol* *contracts/BBT/BBT721.sol*		
Locations	*Several functions throughout multiple contracts*		

Detailed Issue

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

- The ***BigBangTheoryNFTV2*** contract
  - The *setBaseURI* function
  - The *lockNFT* function
  - The *unlockNFT* function
  - The *adminContractSet* function
- The ***BigBangTheoryTokenV2*** contract
  - The *adminContractSet* function
- The ***BigBangTheoryTokenUnlimitV1*** contract
  - The *adminContractSet* function
- The ***BBT20*** contract
  - The *_adminContractSet* function
- The ***BBT721*** contract
  - The *_adminContractSet* function

Recommendations

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

Reassessment

The *Big Bang Theory* team adopted our recommended code to fix this issue.

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/