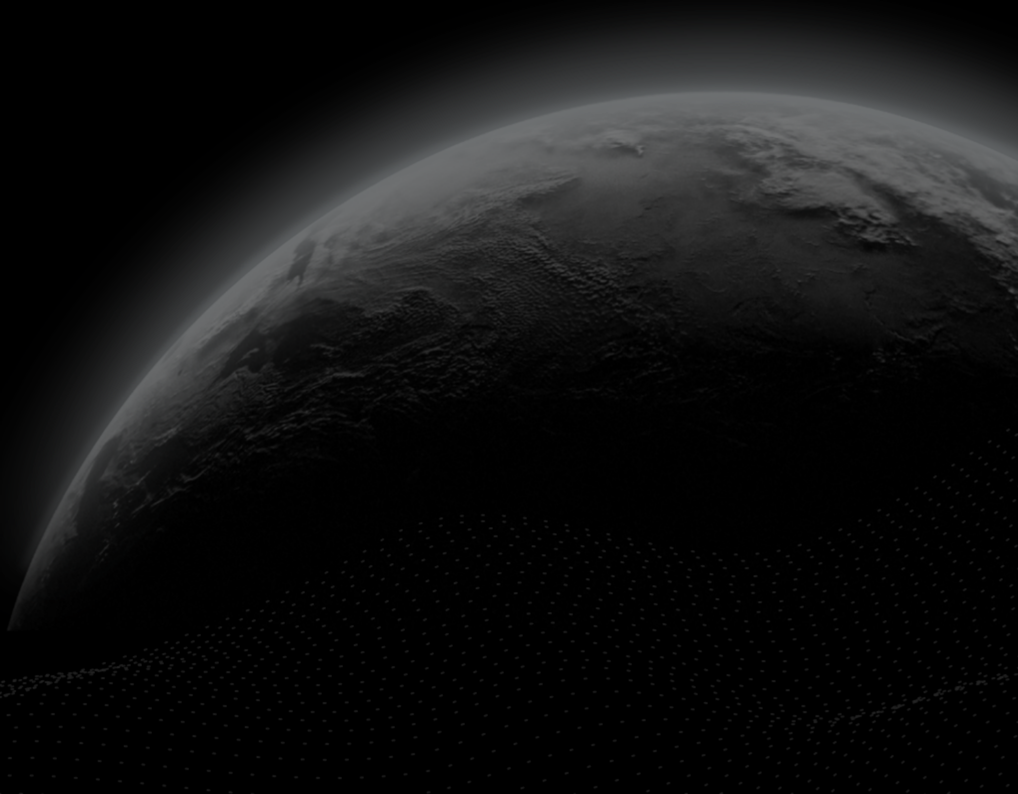
CERTIK

Security Assessment

BlackFort Group

CertiK Verified on Oct 13th, 2022



CertiK Verified on Oct 13th, 2022

BlackFort Group

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
ERC-721, NFT	Ethereum	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 10/13/2022	N/A

CODEBASE

https://github.com/BlackFortGroup/blackfort-network

...View All

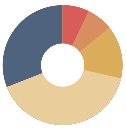
COMMITS

base: 92a1060005dd69ad2e63046cc6ff59d6eae4e1f7

update: d5706ad2c7550f481b9f480af76e20e2da57fbf1

...View All

Vulnerability Summary



42	37	0	4	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

🟥	3	Critical	3 Resolved	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	3	Major	2 Resolved, 1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨	6	Medium	5 Resolved, 1 Partially Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	17	Minor	15 Resolved, 2 Partially Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	13	Informational	12 Resolved, 1 Partially Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BLACKFORT GROUP

Summary

Findings

CODEBASE | BLACKFORT GROUP

Repository

https://github.com/BlackFortGroup/blackfort-network

Commit

base: 92a1060005dd69ad2e63046cc6ff59d6eae4e1f7

update: d5706ad2c7550f481b9f480af76e20e2da57fbf1

AUDIT SCOPE | BLACKFORT GROUP

33 files audited ● 11 files with Acknowledged findings ● 22 files with Resolved findings

ID	File	SHA256 Checksum
● BXA	contracts/BXP/BXP20Asset.sol	91eaf81501802aab0b4dd6367aa0ce9fed97302381902847185ae871c9d06fa7
● BXS	contracts/extensions/BXP20SystemRewardToken.sol	86ee21ee13ec3a5f39641530e843e8dec4476c6c9689e908b777a364bd58a430
● SAB	contracts/extensions/SystemAccess.sol	b3564c1391a184025aa5093ffb6328f6efd5c9871058ee0bddd3fab2ea689962
● ACH	contracts/AccessControlHub.sol	43ac4271533e065c7b8f73c2494f1f0586349bc800709aa3c7b12983954a073d
● CHB	contracts/CandidateHub.sol	9611aab885cd4125cc51e4cc122049604513d85934fdf0f1605cc3c9d4eec958
● NHB	contracts/NodeHub.sol	235bc745f852a4c425be9028ddd6b3021b057ff5a4acc6cc4ead769aefd952de
● PHB	contracts/PollHub.sol	112191b2daa1becc5a3af8b9ab7bc705e06e5c7766b161c4125d623c1ef846ed
● SHB	contracts/SlashingHub.sol	b8022459a840ac4aed9bfe6cce0ea30037a6b744cdc032bab89110dc84162800
● SBF	contracts/System.sol	b3309306a6b3fa4f1b7c8423572d4eb111ee7c0dec325c38d9ba0680a3f99f37
● VHB	contracts/VoteHub.sol	149464ae4821e5a4ce4f404dd93086c0349bd3d7432125e12298fe65490c7a16
● VHF	contracts/ValidatorHub.sol	5452fa69b5be29253a0eb9d38d9ab67e9595da1455a2a936df3202dd85ab751f
● IBX	contracts/BXP/interfaces/IBXP165.sol	46e3031b0934c54195519dffb311972d1abbdf4e3efad62d891d322e8dc2544f
● IBP	contracts/BXP/interfaces/IBXP20.sol	a080b437ed1f43e3ddf334de3366c2bff0d81641dc9931919c3176133227fee0
● IBM	contracts/BXP/interfaces/IBXP20Metadata.sol	78e894507f46f836a8920eebaa4da0476903b5cd36fc1442701b903d018d9fd4

ID	File	SHA256 Checksum
● IBB	contracts/BXP/interfaces/IBXP721.sol	ce889aa7229eb27a842bae079741ed95b71c447bce42f57405d1f13b33318936
● IBE	contracts/BXP/interfaces/IBXP721Enumerable.sol	f08164b44f09c9827a4a972fe81b3e9f8c6c9ec4a9f4113b573eec19bcd8b48c
● IBF	contracts/BXP/interfaces/IBXP721Metadata.sol	07fa4bef7c93d14738884229e0db351dfa9e52de8ada7b7d6f891e6421ec55f3
● IBR	contracts/BXP/interfaces/IBXP721Receiver.sol	c7d045ae89e980f94e8e824c0eee4683e2da75c5a1b37ad0f5f9093e856f394a
● BXX	contracts/BXP/BXP165.sol	a16c145ab5181cab12ea4ddfa65d4583e452d003733867ab8086e4c4ec494189
● BXF	contracts/BXP/BXP20.sol	a40f0822af198f39c6c74628fafc75d5b95836808281025b17b1a511794b234d
● BXG	contracts/BXP/BXP721.sol	93ebce44dd746466133d47e1ac9de548b33e4d5792f9f674be377e4e07dd8b0f
● BXE	contracts/BXP/BXP721Enumerable.sol	01b5a3ede3d9cd07d7508f67285d75d9c00a239599f55cdc035c92cae3d5fde2
● WBX	contracts/BXP/WBXN.sol	987a436b98306363e4514acc99939345e73f7604f5ce58dcf78b5a28f6c1e3e4
● EMB	contracts/extensions/ExtendedMath.sol	31aa3b882ac68c9bcc9c2c620c5214b87994870ddb75b9433e3ba253e77bcc0d
● IAC	contracts/interfaces/IAccessControlHub.sol	36f12685e948197e1e5230aaf55e687af9253ff3ef2cf48ae28e2149ccde369a
● IBS	contracts/interfaces/IBXP20SystemRewardToken.sol	36ae6b29c10b95e3d43ddd157650c2c5d7d5a32d4c88586dedb50e790b407c42
● IDH	contracts/interfaces/IDelegatorHub.sol	f3fdec93fc4b407ff2d75b1b0756d7ef62369b644b07471a4aa490d971e7431d
● INH	contracts/interfaces/INodeHub.sol	d793c1ca9cda322876da72411bc2635e78e164bccc58ee911cbf88058eeea55e
● ISH	contracts/interfaces/ISlashingHub.sol	d3dedf3c64bdf7765dcd614c2f521543b536e3eb3943bf5d3949c393dc2038d2
● ISB	contracts/interfaces/ISystem.sol	3a5fbcd3927e4a85287284282d9f683f7ceb4fb25715bb02e71ce43681c2edea

ID	File	SHA256 Checksum
● IVH	contracts/interfaces/IValidatorHub.sol	785cd121fb7d5a9403217237fd81febda5f77c155b8e604669846ab77f5f8689
● IVB	contracts/interfaces/IVoteHub.sol	7de61fd88f6fc231dd0f6d789279b6b8fc84634edeb8c9acac13f2155337940e
● DHB	contracts/DelegatorHub.sol	0b3370ed8aeefbfc23c055f6af7565efb2d413307820a2ecab1ebf85061c12f0

APPROACH & METHODS | BLACKFORT GROUP

This report has been prepared for BlackFort Group to discover issues and vulnerabilities in the source code of the BlackFort Group project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

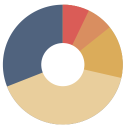
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | BLACKFORT GROUP



42 Total Findings

3 Critical

3 Major

6 Medium

17 Minor

13 Informational

This report has been prepared to discover issues and vulnerabilities for BlackFort Group. Through this audit, we have uncovered 42 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ACH-01	Function `init()` Can Be Called More Than Once	Inconsistency, Control Flow	Minor	● Resolved
BFG-01	No Bound On The Amount A `validatorAccount` Can Mint	Control Flow	Critical	● Resolved
BFG-02	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	● **Acknowledged**
BFG-03	Potential Reentrancy Attack	Volatile Code	Major	● Resolved
BFG-04	Entire `burntAmount` For Account Is Checked Against `minted` Amount For One `tokenId`	Logical Issue	Medium	● Resolved
BFG-05	Unused Return Value	Volatile Code	Minor	● Resolved
BFG-06	Missing Zero Address Validation	Volatile Code	Minor	● Resolved
BFG-07	Shadowing State Variable	Coding Style	Minor	● Resolved
BFG-08	Unchecked Value Of `address.call()`	Logical Issue	Minor	● Resolved
BFG-09	Check Effect Interaction Pattern Violated	Logical Issue	Minor	● Partially Resolved

ID	Title	Category	Severity	Status
BFG-10	`totalSupply` Is Not Updated With Every Token Burn	Language Specific	Minor	● Resolved
BFG-11	Bool Should Be Returned And Checked By Function `accept()` In `CandidateHub`	Inconsistency	Minor	● Resolved
BFG-12	Missing Input Validation	Logical Issue	Minor	● Resolved
BXA-01	`super.mint` Incorrectly Called In `burn()` Function	Logical Issue	Critical	● Resolved
BXS-01	Underflow Vulnerability Through Use Of Signed Integers	Logical Issue, Mathematical Operations	Minor	● Partially Resolved
DHB-01	Anyone Can Call `burnExtraFor()`	Logical Issue	Minor	● Resolved
NHB-01	`burntAmount` Not Updated Correctly For `to` And `from` Addresses	Logical Issue, Inconsistency	Medium	● Resolved
NHB-02	No Validation Check On The Function `unlock()` And `lock()`	Logical Issue	Minor	● Resolved
PHB-01	Wrong Index In The Loop Of Removing Option	Logical Issue	Major	● Resolved
PHB-02	Incorrect Condition In Modifier `pollNotOpened`	Logical Issue	Medium	● Resolved
PHB-03	`pollExists` Modifier Makes Functions Unusable If Tokens Are Burned	Logical Issue	Medium	● Resolved
PHB-04	Deadline Can Be Updated To A Block Before Previous Deadline	Logical Issue	Medium	● Resolved
PHB-05	Locked Ether	Language Specific	Minor	● Resolved
PHB-06	User May Overpay In `start()` Function	Logical Issue	Minor	● Resolved

ID	Title	Category	Severity	Status
SAB-01	Lack Of Access Control	Control Flow	Critical	Resolved
SAB-02	Hardcoded Address	Volatile Code	Minor	Resolved
SAB-03	Modifier Does Not Check For Intended Functionality	Coding Style, Language Specific	Minor	Resolved
SBF-01	No Validation For String Input In `approve()`	Data Flow	Medium	Partially Resolved
SHB-01	Lack Of Validation For `byBlock`	Volatile Code	Minor	Resolved
BFG-16	Unlocked Compiler Version	Compiler Error	Informational	Resolved
BFG-17	Missing Emit Events	Language Specific	Informational	Resolved
BXA-03	`burn()` And `destroy()` Have The Same Intended Utility For Two Distinct Parties	Coding Style, Inconsistency	Informational	Resolved
CHB-01	Logic Issue When Adding Users To Candidates	Logical Issue	Informational	Partially Resolved
EMB-01	Dead Code	Coding Style	Informational	Resolved
NHB-04	Unclear If Contract Is Upgradeable	Control Flow	Informational	Resolved
PHB-08	`mint()` Function Does Not Take A Fee	Language Specific	Informational	Resolved
SAB-04	Declaration Naming Convention	Coding Style	Informational	Resolved
SAB-05	Function `set_SYSTEM_CONTRACT_ADDRESS` Defined Before Modifiers	Coding Style	Informational	Resolved

ID	Title	Category	Severity	Status
SBF-02	No Refund If Caller Is Not Validator	Logical Issue	Informational	● Resolved
SBF-03	Race Condition For Third Party Addresses	Control Flow	Informational	● Resolved
SHB-02	`_timesSlashed` Updated To 2 The First Time `slash()` Called	Inconsistency	Informational	● Resolved
VHF-01	Validators Can Set Their Own Commission	Logical Issue	Informational	● Resolved

ACH-01 | FUNCTION `init()` CAN BE CALLED MORE THAN ONCE

Category	Severity	Location	Status
Inconsistency, Control Flow	● Minor	contracts/AccessControlHub.sol: 31~32	● Resolved

Description

The function `init()` can only be called by the constant address `DEFAULT_ADMIN_ROLE_ADDRESS` , in which case the internal function `_setupRole()` is directly called from the inherited `AccessControl` contract to transfer the role `DEFAULT_ADMIN_ROLE` to `DEFAULT_ADMIN_ROLE_ADDRESS` . If this role is ever renounced or granted to another address, `DEFAULT_ADMIN_ROLE_ADDRESS` can be reinstated to this role at any time.

Recommendation

We recommend considering if this is the intended effect. If so, no action is needed. Otherwise, consider using the `constructor()` to instead set `DEFAULT_ADMIN_ROLE` as `DEFAULT_ADMIN_ROLE_ADDRESS` or adding in extra validation that does not allow this function to be called at any time. A validation check that `DEFAULT_ADMIN_ROLE` is not already `DEFAULT_ADMIN_ROLE_ADDRESS` is recommended for optimization.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 162b66e39d2de2f8e5b1c38cb3e2d320cf128691.

BFG-01 NO BOUND ON THE AMOUNT A `validatorAccount` CAN MINT

Category	Severity	Location	Status
Control Flow	● Critical	contracts/DelegatorHub.sol: 76~77; contracts/ValidatorHub.sol: 66~67	● Resolved

Description

If `mint()` is called with an address `validatorAccount` that is in the set of `_validators` in `ValidatorHub` , then the caller can send as many `DelegatorHub` tokens as they want to the `validatorAccount` address. These tokens can then be burned by the `validatorAccount` , which calls the `transferTo()` function in the `System` contract. This function sends an equal amount of native BXN tokens to the `validatorAccount` . Note `mint()` can only be called by the `SYSTEM_CONTRACT_ADDRESS` ; since `SYSTEM_CONTRACT_ADDRESS` can be changed by anyone, this vulnerability is critical.

Recommendation

We recommend protecting this function so that external users may not call it themselves.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above by resolving finding SAB-01.

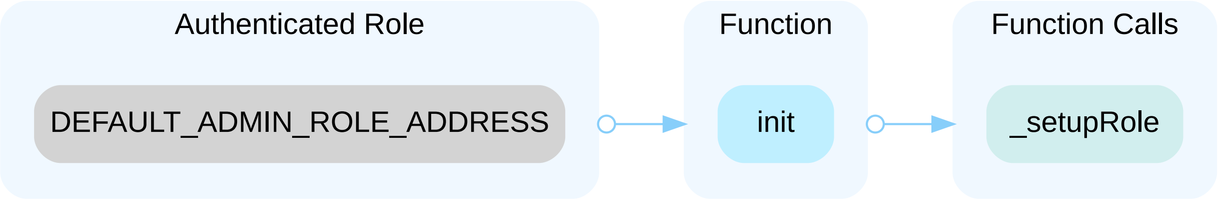
BFG-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	● Major	contracts/AccessControlHub.sol: 31, 44, 51; contracts/BXP/BXP20Asset.sol: 30, 37, 44, 56, 61, 71~72, 75~76, 83; contracts/CandidateHub.sol: 39, 50, 65; contracts/NodeHub.sol: 79, 191, 197; contracts/PollHub.sol: 73~74, 79~80, 84~85, 89~90, 94~95, 110, 118, 130~131, 135~136, 143, 147, 159; contracts/SlashingHub.sol: 30; contracts/System.sol: 34, 84~85; contracts/ValidatorHub.sol: 70~71, 77~78; contracts/VoteHub.sol: 24, 32, 38; contracts/extensions/BXP20SystemRewardToken.sol: 37	● Acknowledged

Description

In the contract `AccessControlHub` the role `DEFAULT_ADMIN_ROLE_ADDRESS` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE_ADDRESS` account may allow the hacker to take advantage of this authority and change the address for privileged roles such as `ACCESS_CONTROL_MANAGER_ROLE` , and any of the roles in contracts that inherit from `SystemAccess` contract.

In addition, any compromise to the `ACCESS_CONTROL_MANAGER_ROLE` account may allow the hacker to enable or disable the `transfer()` function for certain contracts.

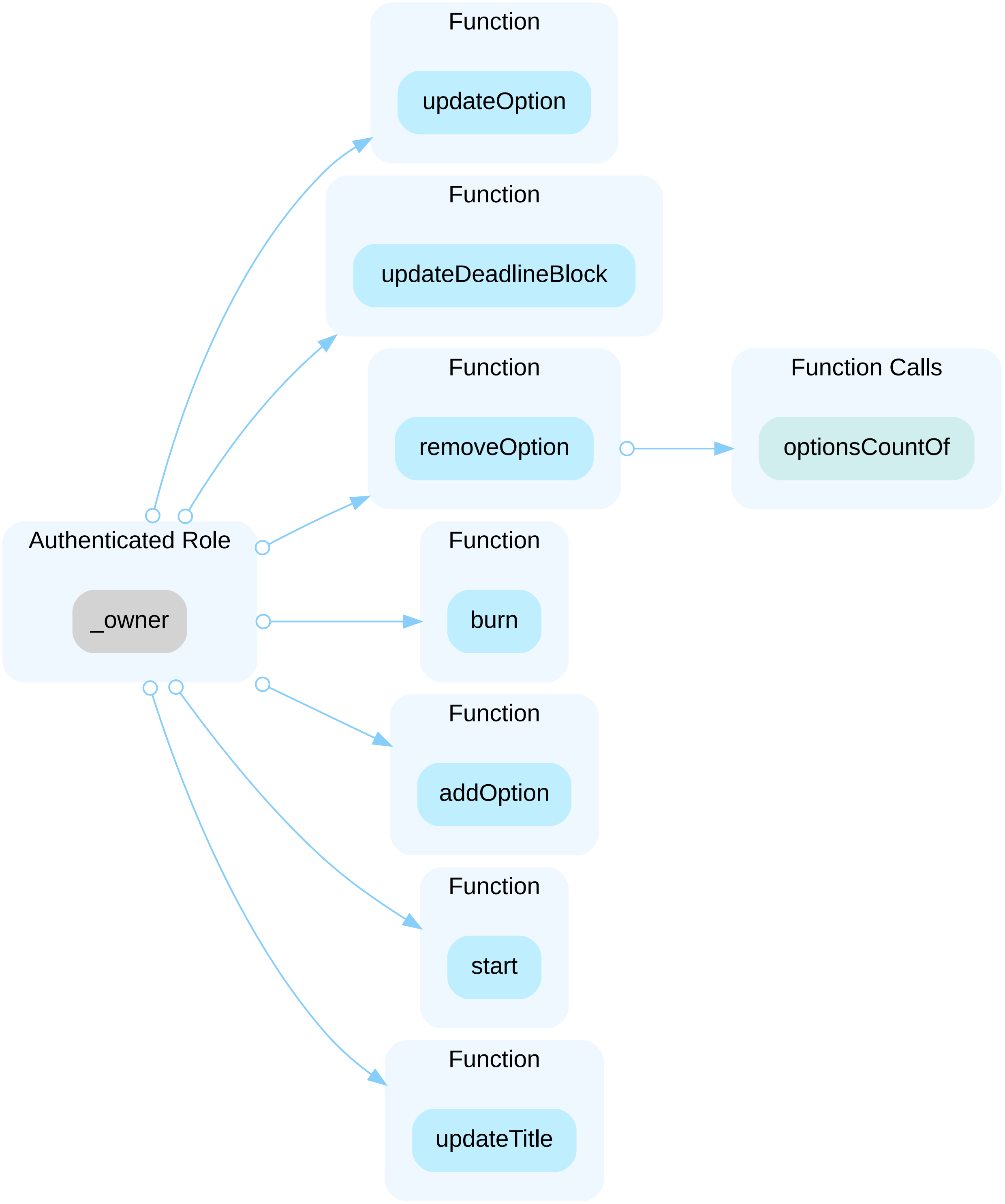


In the contract `System.sol` , any compromise to the `SYSTEM_MANAGER_ROLE` account may allow the hacker to approve any account to use the `transferTo()` function to send any amount of native token to an address of their choosing. Additionally, an address with the `VOTE_MINT_ROLE` can use this same function to mint any amount of `VoteHub` tokens to any address.

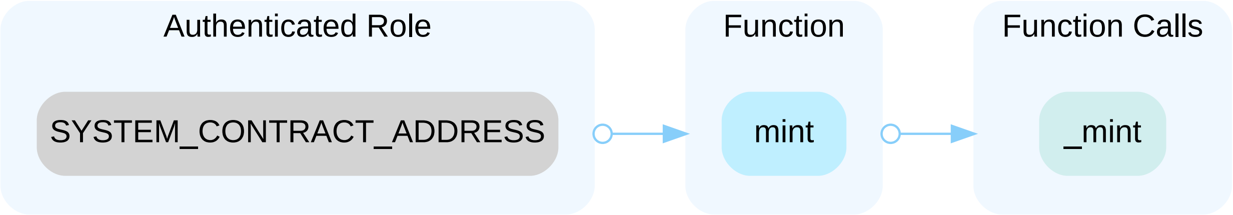
In the contract `NodeHub` , any compromise to the `NODE_MANAGER_ROLE` account may allow the hacker to set `_baseTokenURI` to any string and use the `lock()` and `unlock()` functions on the underlying token to prevent transfers.

In the contract `PollHub` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority, modify the information of poll such as its poll options and its deadline. Note that the `_owner` in this case refers to the owner of the specified `tokenId` representing a poll. Each instance of a poll represents a centralization risk.

In addition, any compromise to the `POLL_MANAGER_ROLE` account may allow the hacker to reset `_baseTokenURI` and change a poll's price and fee to an unwanted amount.



In the contract `BXP20SystemRewardToken` the role `SYSTEM_CONTRACT_ADDRESS` has authority over the functions shown in the diagram below. Any compromise to the `SYSTEM_CONTRACT_ADDRESS` account may allow the hacker to take advantage of this authority and mint any amount of tokens to any address within the contracts that inherit from this base; that is the `DelegatorHub` and `ValidatorHub` contracts



In the contract `BXP20Asset` , which inherits from the contract `BlackList` , any compromise to the `ASSET_BLACKLIST_MANAGER_ROLE` account may allow the hacker to add addresses to the blacklist. The contract `BXP20Asset` also inherits from the contract `Manageable` ; any compromise to the `ASSET_MANAGER_ROLE` account may allow the hacker to mint and burn any amount of tokens for any address.

The role `VALIDATOR_MANAGER_ROLE` has control over several contracts. In the contract `CandidateHub` , any compromise to the `VALIDATOR_MANAGER_ROLE` account may allow a hacker to add candidates to the validator set, remove candidates maliciously and set `requiredAmount` to any amount. In the contract `SlashingHub` , compromise to this role allows a hacker to use the `slash()` function to penalize rewards for a validator address. In the contract `ValidatorHub` , this may also allow the hacker to use the function `kick()` to remove any validator from the list.

In the contract `VoteHub` , any compromise to the `VOTE_MINT_ROLE` account may allow the hacker to mint any amount of `VoteHub` tokens to any account and any compromise to the `VOTE_BURN_ROLE` account may allow the hacker to burn any amount of `VoteHub` tokens. In addition, any compromise to the `VOTE_SPENDER_ROLE` may allow the hacker to spend all the `VoteHub` tokens belonging to any account.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

`[BlackFort Group]` : Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

BFG-03 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Volatile Code	● Major	contracts/CandidateHub.sol: 54, 56; contracts/NodeHub.sol: 149~150, 247 ~248; contracts/System.sol: 82~83	● Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Such an attack can come from the transfer of native tokens such as `ETH` but can also be a risk with token contracts conforming to the `ERC721` , `ERC777` and `ERC1155` standard in which a contract that is transferred these kinds of tokens must have a base that allows for holding such tokens. This base makes a callback to the token contracts which, should this callback function be modified, may contain malicious code.

Potential Reentrancy Involving Ether

External call(s)

```
54          (bool result,) = account.call{value:amount}("");
```

State variables written after the call(s)

```
56              _candidatesBonds[account] = _candidatesBonds[account].sub(amount);
```

External call(s)

```
82          (bool result,) = account.call{value:amount}("");
```

State variables written after the call(s)

```
83          _spentAmount[msg.sender] = _spentAmount[msg.sender].add(amount);
```

External call(s)

```
         _burnFrom(from, reward - burnedByFrom)
```

State variables written after the call(s)

```
        _burntAmount[from] -= reward;
        _burntAmount[to] += reward;
```

Potential Reentrancy Involving `ERC721` tokens (`BXP721`)

External call(s)

```
        _mint(owner, tokenId);
```

State variables written after the call(s)

```
        __delegators[tokenId] = address(0);
                _rewardShares[tokenId] = nodeTypes[i].rewardShare;
                _mintedAtBlock[tokenId] = block.number;
                _nodeType[tokenId] = i;

                nodeTypes[i].quantity = nodeTypes[i].quantity.sub(1);

                _depositedAmount[owner] = _depositedAmount[owner].add(nodePrice);
                amount = amount.sub(nodePrice);

                _tokenIdTracker.increment();
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 09d4d6a9dc5883cacd83fcd8053b2dc78196955a.

BFG-04 ENTIRE burntAmount FOR ACCOUNT IS CHECKED AGAINST minted AMOUNT FOR ONE tokenId

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/DelegatorHub.sol: 57~58, 82~83; contracts/NodeHub.sol: 232~233	● Resolved

Description

The function `burnExtraFor()` compares the entire amount a `delegatorAccount` (owner of a given `tokenId`) has burned of the `BXP20` `DelegatorHub` token to the amount that is minted corresponding to one `BXP721` token. If the `minted` amount corresponding to the `tokenId` is larger than the total `burntAmount` corresponding to the account, then the difference between the values is burned. These values do not necessarily correspond to one another.

As a simplified example, say that one token owner owns two tokens, token 1 and token 2. Token 1 has a corresponding reward amount of 10 `DelegatorHub` tokens, and token 2 has a corresponding reward of 20 `DelegatorHub` tokens. The owner burns the corresponding `DelegatorHub` tokens directly through the `burn()` function, and the `burntAmount` is now 10. If someone calls `burnExtraFor()` on token 2, the `minted` amount will be 20 while `burntAmount` will be 10, so the owner will now only have 10 tokens burned corresponding to token 2. After, `burntAmount` for the owner is 20, and `burnExtraFor()` will no longer burn `DelegatorHub` tokens.

Additionally, since the function `burnExtraFor()` relies on the return value of `mintedWith()` rather than the actual token balance corresponding to a given address, there is no way for the user to burn or transfer these tokens.

A similar issue occurs within the hook `_beforeTokenTransfer()` in the `NodeHub` contract.

Recommendation

We recommend either comparing a minted and burned amount for one given token ID or comparing a total minted and burned amount for one given address, dependent on context. Moreover, we recommend updating a token owner's balance to reflect the corresponding reward tokens associated with owning a `BXP721` `NodeHub` token.

Alleviation

`[CertiK]` : See below for the team's explanation on the validity of the mechanism.

`[BlackFort Group]` : "The idea of burnExtraFor method is to burn exceeding amount of not yet claimed DelegatorHub tokens for exact token before performing re/un-delegation. During redelegation burntAmount of current owner decreases by total value mintedWith the NodeHub token. On NodeHub token transfer for logic security NodeHub token is undelegated in order to return earned amount of tokens.

I can continue your example with further transfer example. We have total burntAmount for current owner 20 and he wants to transfer token #2 to someone else. As far burntAmount is 20 and mintedWith for token #2 is also 20, nothing will be burned by burnExtraFor method and NodeHub token #2 will be undelegated before the transfer. While undelegate total burntAmount will be decreased by mintedWith of #2 which is 20 so it makes burntAmount of current address on DelegatorHub equal to 0 which allows him to claim rest 10 he had with token #1

Concept relies on differences of income from multiple owned tokens and total spent amount by current owner. When we do manipulation with token, we have to reflect it on total burntAmount to keep calculations correct. It works same way on NodeHub, ValidatorHub and DelegatorHub."

BFG-05 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/CandidateHub.sol: 35, 47, 57; contracts/NodeHub.sol: 229; contracts/ValidatorHub.sol: 72, 81; contracts/VoteHub.sol: 34; contracts/extensions/BXP20SystemRewardToken.sol: 86	● Resolved

Description

The return value of an external call is not stored in a local or state variable, and there exists no check to ensure successful execution.

```
35          _candidates.add(account);
```

```
47          _candidates.remove(account);
```

```
57              _candidates.remove(account);
```

```
229          _getSystemContractInstance().transferTo(owner, amount);
```

```
72          _validators.add(account);
```

```
81          _validators.remove(account);
```

```
34          _getSystemContractInstance().transferTo(account, amount);
```

```
86          _getSystemContractInstance().transferTo(account, amount);
```

Recommendation

We recommend the client check the return values of all external function calls to ensure the correct outcome has taken effect.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 498a5f3f405e938cdaf4d5b0ba2373ce6036b378.

BFG-06 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/AccessControlHub.sol: 47, 53; contracts/System.sol: 82; contracts/extensions/SystemAccess.sol: 13, 48	● Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not `address(0)` .

```
82          (bool result,) = account.call{value:amount}("");
```

- `account` is not zero-checked before being used.

```
13          SYSTEM_CONTRACT_ADDRESS = name;
```

- `name` is not zero-checked before being used.

```
48          SYSTEM_CONTRACT_ADDRESS = addr;
```

- `addr` is not zero-checked before being used.

```
47          _transferAllowed[account] = true;
```

- `account` is not zero-checked before being used.

```
53          _transferAllowed[account] = false;
```

- `account` is not zero-checked before being used.

Recommendation

We recommend adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

`[BlackFort Group]` : " `SYSTEM_CONTRACT_ADDRESS = name;` and `SYSTEM_CONTRACT_ADDRESS = addr;` are only for development version. Not intended to be in production because it changing the SYSTEM_CONTRACT_ADDRESS may lead to break of smart contracts. Issue acknowledged. Changes have been reflected in the commit hash 83a7608ac20bea77220fa76a8e480cdbe6294341."

BFG-07 | SHADOWING STATE VARIABLE

Category	Severity	Location	Status
Coding Style	● Minor	contracts/BXP/BXP20.sol: 35, 39; contracts/extensions/BXP20SystemRewardToken.sol: 15, 19	● Resolved

Description

A state variable is shadowing another component defined in a parent contract.

Variable `_totalSupply` in `BXP20SystemRewardToken` shadows the variable `_totalSupply` in `BXP20` .

```
15      uint256 private _totalSupply;
```

```
39      uint256 private _totalSupply;
```

Variable `_balances` in `BXP20SystemRewardToken` shadows the variable `_balances` in `BXP20` .

```
19      mapping(address => int256) private _balances;
```

```
35      mapping(address => uint256) private _balances;
```

Recommendation

We recommend removing or renaming the state variable that shadows another definition.

Alleviation

`[BlackFort Group]` : "Partly resolved in commit a28ad84f992f28090b62274f392f5c754508816e. We don't have some way to change _totalSupply in BXP20 contract which leads to variable shadowing."

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit a28ad84f992f28090b62274f392f5c754508816e.

BFG-08 | UNCHECKED VALUE OF `address.call()`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/NodeHub.sol: 188; contracts/ValidatorHub.sol: 84	● Resolved

Description

The linked statement transfers the native token to the specified address. The `address.call()` function may return false if the aforementioned transaction is failed. If this return value is not checked, the receiving address is not transferred tokens, while the related variables have been set to zero or lost, and the tokens cannot be refunded.

```
188         (bool result,) = msg.sender.call{value:refund}("");
```

```
 84         (bool result,) = account.call{value:amount}("");
```

Recommendation

We recommend the client check the local variable `result`.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 498a5f3f405e938cdaf4d5b0ba2373ce6036b378.

BFG-09 | CHECK EFFECT INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/CandidateHub.sol: 45~46, 54~57; contracts/NodeHub.sol: 169~170, 208; contracts/PollHub.sol: 173~178	● Partially Resolved

Description

The order of external calls or transfers and storage manipulation must follow the check-effect-interaction pattern to keep contract logic safe from exploitation.

External call(s)

```
45          ValidatorHub.join{value:amount}(account);
```

State variables written after the call(s)

```
46          _candidatesBonds[account] = _candidatesBonds[account].sub(amount);
```

External call(s)

```
169          VoteHub.mint(owner, nodePrice.div(100));
```

State variables written after the call(s)

```
180          amount = amount.sub(nodePrice);
```

External call(s)

```
244          delegate(address(0), tokenId);
```

- This function call executes the following external call(s).
- In `NodeHub.delegate`,
    - `DelegatorHub.burnExtraFor(tokenId)`
- In `NodeHub.delegate`,

  - `DelegatorHub.decreaseDelegatedAmountFor(currentValidator,tokenId)`
- In `NodeHub.delegate` ,
  - `DelegatorHub.increaseDelegatedAmountFor(validatorAddress,tokenId)`
- This call sends Ether.

```
            _burnFrom(from, reward - burnedByFrom);
```

- This function call executes the following external call(s).
- In `NodeHub._burnFrom` ,
  - `_getSystemContractInstance().transferTo(owner,amount)`
- This call sends Ether.

State variables written after the call(s)

```
        super._beforeTokenTransfer(from, to, tokenId);
```

- This function call executes the following assignment(s).
- In `BXP721Enumerable._addTokenToAllTokensEnumeration` ,
  - `_allTokens.push(tokenId)`
- In `BXP721Enumerable._removeTokenFromAllTokensEnumeration` ,
  - `_allTokens[tokenIndex] = lastTokenId`
- In `BXP721Enumerable._removeTokenFromAllTokensEnumeration` ,
  - `_allTokens.pop()`

```
        super._beforeTokenTransfer(from, to, tokenId);
```

- This function call executes the following assignment(s).
- In `BXP721Enumerable._addTokenToAllTokensEnumeration` ,
  - `_allTokensIndex[tokenId] = _allTokens.length`
- In `BXP721Enumerable._removeTokenFromAllTokensEnumeration` ,
  - `_allTokensIndex[lastTokenId] = tokenIndex`

- In `BXP721Enumerable._removeTokenFromAllTokensEnumeration` ,
    - `delete _allTokensIndex[tokenId]`

```
        super._beforeTokenTransfer(from, to, tokenId);
```

- This function call executes the following assignment(s).
- In `BXP721Enumerable._addTokenToOwnerEnumeration` ,
    - `_ownedTokensIndex[tokenId] = length`
- In `BXP721Enumerable._removeTokenFromOwnerEnumeration` ,
    - `_ownedTokensIndex[lastTokenId] = tokenIndex`
- In `BXP721Enumerable._removeTokenFromOwnerEnumeration` ,
    - `delete _ownedTokensIndex[tokenId]`

External call(s)

```
        (bool result,) = account.call{value:amount}("");
```

State variables written after the call(s)

```
        _candidatesBonds[account] = _candidatesBonds[account].sub(amount);
            _candidates.remove(account);
```

External call(s)

```
        VoteHub.burn(msg.sender, amountOfVote.sub(fee));
        VoteHub.transferFrom(msg.sender, ownerOf(tokenId), fee);
        VoteHub.burn(ownerOf(tokenId), fee);
```

State variables written after the call(s)

```
        _pollVoteAmount[tokenId][optionId] = _pollVoteAmount[tokenId][optionId].add(amountOfVote);
        _pollTotalVoteAmount[tokenId] = _pollTotalVoteAmount[tokenId].add(amountOfVote);
```

Recommendation

We recommend the client always check the storage variables affected by an external call first, then update the storage variables affected by the external call, and finally make the external call itself.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash a4313c0864b2f69d73280d97ceab2e680da3c911.

`[CertiK]` : The issue still persists for the following locations:

- Function `accept()` in contract `CandidateHub` still makes an external call to function `join()` in `ValidatorHub before updating `_candidatesBonds[account]`.
- Function `delegate()` in contract `NodeHub` still makes external calls to `DelegatorHub` and `SlashingHub` before updating `_delegators[tokenId]` .

BFG-10 | `totalSupply` IS NOT UPDATED WITH EVERY TOKEN BURN

Category	Severity	Location	Status
Language Specific	● Minor	contracts/DelegatorHub.sol: 39~40, 49~50; contracts/extensions/BXP20SystemRewardToken.sol: 93~94, 97~98	● Resolved

Description

The functions `_increaseBurntAmountOf()` and `_decreaseBurntAmountOf()` are called in contracts which inherit contract `BXP20SystemRewardToken` as a base. Since these functions are called directly and not through `_burn()` in that circumstance, the value for `_totalSupply` is not updated to reflect the total number of tokens present in the contract.

Recommendation

We recommend updating the `_totalSupply` directly in the `_increaseBurntAmountOf()` and `_decreaseBurntAmountOf()` functions, since `_burn()` calls `_increaseBurntAmountOf()` .

Alleviation

`[CertiK]` : Please see the team's explanation below concerning the validity of mechanism.

`[BlackFort Group]` : "These methods update shares of distribution. When you undelegate NodeHub token the burnExtraFor method already processes changes in totalSupply, then decreaseDelegatedAmountFor and increaseDelegatedAmountFor change only shares in a way that it's not reflected on users' balances, so totalSupply is not changed"

BFG-11 | BOOL SHOULD BE RETURNED AND CHECKED BY FUNCTION `accept()` IN `CandidateHub`

Category	Severity	Location	Status
Inconsistency	● Minor	contracts/CandidateHub.sol: 39~40; contracts/ValidatorHub.sol: 72~73	● Resolved

Description

The bool return value for `_validators.add(account)` should be returned by the function `join()` and checked by the `accept()` function in the `CandidateHub` contract when called to ensure the correct outcome takes effect. Otherwise, in the `CandidateHub` contract, `_candidates` and `_candidatesBonds` may be updated without the account being added to the `_validators` list.

Recommendation

We recommend returning the bool value of `_validators.add(account)` at the end of the function call `join()`.

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash 778ab1b00fa0dce352b31e6b72fe372afe0b1fcb.

BFG-12 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/CandidateHub.sol: 65; contracts/PollHub.sol: 79~80, 84~85, 89~90	● Resolved

Description

The given input is missing a check for a nonzero amount.

Recommendation

We recommend the client add the necessary check for the mentioned functions. Ideally, each input would be checked against a chosen upper and lower bound.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash cac887bb16759e60f6df3fcdd516f73091ab8963.

BXA-01 | `super.mint` INCORRECTLY CALLED IN `burn()` FUNCTION

Category	Severity	Location	Status
Logical Issue	● Critical	contracts/BXP/BXP20Asset.sol: 76	● Resolved

Description

In the contract `BXP20Asset`, the override function `burn` calls the function `mint` of the next most derived contract.

```
    function burn(address account, uint256 amount) public override
notInBlackList(account) {
        super.mint(account, amount);
    }
```

Recommendation

We recommend the client call `super.burn()` function in the function `burn`.

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash 792ecdc81c99ebe5f4d25a4977a237d02de71e3f.

BXS-01 UNDERFLOW VULNERABILITY THROUGH USE OF SIGNED INTEGERS

Category	Severity	Location	Status
Logical Issue, Mathematical Operations	● Minor	contracts/extensions/BXP20SystemRewardToken.sol: 29~30	● Partially Resolved

Description

The `balanceOf()` function is overridden from the base `BXP20` contract, where the balance is calculated by first adding the amount minted and amount recorded in `_balances` for an account as signed integers. This value is converted to an unsigned integer and the amount burned by the address is subtracted. If the contract logic allows for the absolute value of `_balances[account]` to be larger than the value of `mintedBy(account)` while `_balances[account]` is a negative value, this will lead to an underflow. Consider the following set up:

Let `_balances[account]` = -2 and `mintedBy(account)` = 1. The sum of the two values is -1 and when that value is converted to an unsigned integer, it causes an underflow, reading the value as 2^256 - 1 instead. The account accomplishing this now has access to an amount larger than what they actually own.

Recommendation

During the audit, no clear path was found for executing the attack vector described above. However, such a path could still exist. We recommend removing the use of signed integers in calculating balances for a token in order to remove unnecessary risk from the contract. The original implementation for determining the balance and transferring tokens in the `BXP20` contract can be used while continuing to record the number of tokens minted and burned through the mappings `_mintedAmount` and `_burntAmount`.

Alleviation

`[CertiK]`: The team acknowledged the finding and took steps towards resolution in commit 9b58d7edff7a095c0ecf897eb8fbe6c925850c9f. However, the changes made appear to completely remove the use of the mapping `_balances` within the contract. The balanceOf() function and _transfer() function no longer reference or update this mapping.

DHB-01 | ANYONE CAN CALL `burnExtraFor()`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/DelegatorHub.sol: 82	● Resolved

Description

There is no access restriction on the function `burnExtraFor` in the contract `DelegatorHub` , allowing anyone to burn the extra amount for a specified token at any time.

Recommendation

We recommend the client re-examine this function and clarify whether everyone should have open access to make this update for any given address.

Alleviation

`[BlackFort Group]` : Functionality limited to NodeHub only at Commit dd4e4d7acc4df78a30bf8449fce1b94a2b8d1f56.

NHB-01 | `burntAmount` NOT UPDATED CORRECTLY FOR `to` AND `from` ADDRESSES

Category	Severity	Location	Status
Logical Issue, Inconsistency	● Medium	contracts/NodeHub.sol: 250~251	● Resolved

Description

In the hook `_beforeTokenTransfer()`, the internal function `_burnFrom()` is called for the `from` address to burn the amount `reward - burnedByFrom`, but after, the mapping `_burntAmount` is *decreased* by the amount `reward` instead of being increased by the difference `reward - burnedByFrom`. On the other hand, the mapping `_burntAmount` is updated for the `to` address by increasing by the amount `reward`. Since the `to` address is not burning tokens, their `_burntAmount` should not be increased at all.

Recommendation

We recommend *increasing* the `_burntAmount` for the `from` address by the value of `reward - burnedByFrom` and removing the updates to the `_burntAmount` for the `to` address.

Alleviation

`[CertiK]`: See the team's explanation of the design choice below.

`[BlackFort Group]`: "availableBalanceOf is difference between all rewards from the owned tokens and total burntAmount for the owner. If we transfer NodeHub token to someone else, we need to claim unclaimed rewards for exact token which is done by reward - burnedByFrom and then decrease burntAmount by reward for current owner and increase by same value burntAmount for new owner because if we'll not do that, new owner wiil have access to possibly already claimed tokens and previous one will not be able to withdraw his earned before tokens or even trap into negative availableBalanceOf

Current realization is a solution for, as I personally call, double-claim problem. BFG-04 is related to this case."

NHB-02 | NO VALIDATION CHECK ON THE FUNCTION `unlock()` AND `lock()`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/NodeHub.sol: 191~192, 197~198	● Resolved

Description

The functions `lock()` and `unlock()` should only be used on existing tokens. Moreover, `lock()` should only be executed if a token is currently unlocked, and `unlock()` should only be executed if a given token is currently locked.

Recommendation

We recommend the client add the necessary checks for the mentioned functions.

Alleviation

`[BlackFort]` : Issue acknowledged. Changes have been reflected in the commit hash 163729ffe51870229fdb570de136232977eba7c6.

PHB-01 | WRONG INDEX IN THE LOOP OF REMOVING OPTION

Category	Severity	Location	Status
Logical Issue	● Major	contracts/PollHub.sol: 137	● Resolved

Description

The function `removeOption` is used to remove the option `optionId` for the token `tokenId` by moving all subsequent options of this option in the array one place forward and removing the last one in the array. But the move operation (line 137) on state variable `_pollOptions` lacks token id, resulting in a removal of all poll options for a given `tokenId`.

```
25      mapping(uint256 => string[]) private _pollOptions;
```

```
135      function removeOption(uint256 tokenId, uint256 optionId) public
onlyOwner(tokenId) pollNotOpened(tokenId) optionExists(tokenId, optionId) {
136          for(uint i = optionId; i < optionsCountOf(tokenId) - 1; i++){
137              _pollOptions[i] = _pollOptions[i + 1];
138          }
139          _pollOptions[tokenId].pop();
140          _pollVoteAmount[tokenId].pop();
141      }
```

Recommendation

We recommend the client fix the wrong index on `_pollOptions` as below:

```
             _pollOptions[tokenId][i] = _pollOptions[tokenId][i + 1];
```

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash a092d35c9462270b12d2d9286a460fef2cf9265d.

PHB-02 | INCORRECT CONDITION IN MODIFIER `pollNotOpened`

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/PollHub.sol: 46	● Resolved

Description

The `_pollDeadline[tokenId]` is set in the function `start()`. Before opening the poll, the `_pollDeadline[tokenId]` is always equal to 0. However, the `start()` function uses the `pollNotOpened` modifier, so the function cannot be executed. Due to this failure, the following functions cannot be executed:

- `burn()`
- `updateTitle()`
- `addOption()`
- `removeOption()`
- `updateOption()`
- `start()`
- `updateDeadlineBlock()` (cannot be executed because poll cannot be opened)
- `vote()` (cannot be executed because poll cannot be opened)

Recommendation

We recommend the client re-examine this modifier as well as the functions that use this modifier, and consider whether the modifier require is meant to read as follows:

```
require(_pollDeadline[tokenId] == 0, "PollHub: poll is either opened or closed");
```

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash addfca7ac042d0867642e6f69c706c5373507ec1.

PHB-03 `pollExists` MODIFIER MAKES FUNCTIONS UNUSABLE IF TOKENS ARE BURNED

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/PollHub.sol: 61~62	● Resolved

Description

The `pollExists` modifier relies on the check that the `tokenId` value is strictly less than the value of `totalSupply()` which comes from `BXP721Enumerable` inheritance. The contract includes the ability to burn the contract non-fungible tokens, and when they are burned, the value for `totalSupply` decreases. However, each time a token is minted, it uses strictly increases values through incrementing `_tokenIdTracker` . As such, if one token is burned, then the most recently issued tokenId will appear to no longer exist under the logic of this modifier. As more tokens are burned, more of the recently minted tokens will appear to no longer exist, even if they have not been burned.

Recommendation

We recommend using the logic built in to `BXP721` contract to check if a `tokenId` exists in order to avoid this issue.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 9c719ccc522a90ab70dde69b7201fe0e05045010.

PHB-04 DEADLINE CAN BE UPDATED TO A BLOCK BEFORE PREVIOUS DEADLINE

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/PollHub.sol: 159~160	● Resolved

Description

The `newBlockDeadline` can be updated to a value that is less than the previous `_pollDeadline` value. In this way, a poll owner can end a poll at a time in which they are satisfied with the current poll results.

Recommendation

We recommend that the value `newBlockDeadline` be checked so that it is not less or equal to the current `_pollDeadline` value.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash a4ee6a3e17bbb5bbf358f4448207eebfd5121837.

PHB-05 | LOCKED ETHER

Category	Severity	Location	Status
Language Specific	● Minor	contracts/PollHub.sol: 147	● Resolved

Description

The contract has one payable function `start`, but does not have a function to withdraw the fund.

```
147      function start(uint256 tokenId, uint256 blockDeadline) public payable
onlyOwner(tokenId) pollNotOpened(tokenId) {
```

Recommendation

We recommend the client add a withdraw function.

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash 6ed1a80e7ba2d493a2469a8f4109e65bd52a3685.

PHB-06 | USER MAY OVERPAY IN `start()` FUNCTION

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/PollHub.sol: 148	● Resolved

Description

If a user calls the function `start()`, they must include a payment equal to `pollPrice`. Since the condition at line 148 for checking the sending value from caller is equal to or greater than `pollPrice`, the user may overpay for the function call without a refund.

```
147     function start(uint256 tokenId, uint256 blockDeadline) public payable onlyOwner(tokenId) pollNotOpened(tokenId) {
148         require(msg.value >= pollPrice, "PollHub: insufficient amount paid");
149         require(blockDeadline >= block.number, "PollHub: deadline block number must be in future");
150         _pollDeadline[tokenId] = blockDeadline;
151 
152         emit PollStarted(tokenId, blockDeadline);
153     }
```

Recommendation

We recommend the client change the statement to avoid overpaying as below:

```
require(msg.value == pollPrice, "PollHub: insufficient amount paid");
```

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash 6ed1a80e7ba2d493a2469a8f4109e65bd52a3685.

SAB-01 | LACK OF ACCESS CONTROL

Category	Severity	Location	Status
Control Flow	● Critical	contracts/extensions/SystemAccess.sol: 12	● Resolved

Description

The function `set_SYSTEM_CONTRACT_ADDRESS` can be called by anyone as it has no access restriction. This enables anyone to call this and set `SYSTEM_CONTRACT_ADDRESS` to a malicious contract that defines the same functions, with unexpected behavior. In addition, there is another test public function `TEST_setSystemContract` which has same functionality without any access restriction. As `SystemAccess` is a base contract for many of the project's contracts, this compromises all contracts that depend on this logic.

Recommendation

We recommend the client add a modifier or require statement to the function `set_SYSTEM_CONTRACT_ADDRESS()` restricting who can set `SYSTEM_CONTRACT_ADDRESS`. An alternative would be to declare a `constructor()` where the deployer can set the `SYSTEM_CONTRACT_ADDRESS`, and removing the vulnerable function completely. Additionally, we recommend removing the test function `TEST_setSystemContract` prior to deployment.

Alleviation

`[BlackFort Group]`: Development methods were removed in commit 83a7608ac20bea77220fa76a8e480cdbe6294341.

SAB-02 | HARDCODED ADDRESS

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/extensions/SystemAccess.sol: 10~11	● Resolved

Description

The `SystemAccess` contract serves as the base contract to several contracts in the scope of this audit. When one of these contracts is deployed, the initial state for `SYSTEM_CONTRACT_ADDRESS` is the address `address(0x0000000000000000000000000000000000001000)`. This address cannot be verified until the blockchain is launched. If this is not the correct address, any attempt to call functions within the derived contracts that depend on this variable may behave unexpectedly and could cause unforeseen issues.

Recommendation

We recommend the team carefully manages any changes to addresses before launch and updates them before deploying the contracts.

Alleviation

`[BlackFort Group]`: Must be always equal to address(0x0000000000000000000000000000000000001000). Development methods were removed in commit 83a7608ac20bea77220fa76a8e480cdbe6294341.

Update 10/13/22 `[BlackFort Group]`: "We'll use set of addresses from 0x0000000000000000000000000000000000000999 to 0x0000000000000000000000000000000000001007

for our core smart contracts. They're all predefined in genesis block."

SAB-03 | MODIFIER DOES NOT CHECK FOR INTENDED FUNCTIONALITY

Category	Severity	Location	Status
Coding Style, Language Specific	● Minor	contracts/extensions/SystemAccess.sol: 29~30	● Resolved

Description

The modifier `transfersAvailable()` reverts if the calling address *does* have transfers available. Since the default `bool` of a mapping is `false`, this will mean any calling contract will pass the modifier unless it is updated to `true`.

Recommendation

If this is the intended functionality, we recommend changing the naming of the related functions and modifiers to reflect this. Otherwise, we recommend removing the negation symbol "!".

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash ab51c879dc7be24d4b2644503139460dda5cd11c.

SBF-01 | NO VALIDATION FOR STRING INPUT IN `approve()`

Category	Severity	Location	Status
Data Flow	● Medium	contracts/System.sol: 34~35	● Partially Resolved

Description

In the `approve()` function, an address has an associated string input `name` that the `account` address is associated to in the mapping `_accounts`. There is no validation check that that the mapping `_accounts` for entry `name` is already occupied. From context, it appears the `approve()` function assigns a contract `name` to in-house contracts addresses, to be used for validation checks in other contracts.

If an externally owned address is approved through this function, the address can directly interact with the `transferTo()` address, which sends out native BXN tokens to specified addresses.

Moreover, any updates to the address corresponding to a given `name` using the `approve()` function causes the previous address to immediately lose privileged access.

Lastly, there is no check that the input `name` is a mapping that is used within other contracts. If the `name` is incorrectly input, then the corresponding checks using the `onlyContract` modifier will not allow a contract to interact.

Recommendation

We recommend adding a validation check that the `name` in the `_accounts` mapping is not currently occupied (and if it is, updating it to be unoccupied first). Further, we recommend that the options for the string input `name` be predetermined using an enum so that unusable strings are not updated in the mapping.

Alleviation

`[BlackFort Group]` : "Issue acknowledged. Changes have been reflected in the commit hash 70116c06e5470d2c80f70bc1a982f4a94c15a7f3.

Still having centralization risks issue."

`[CertiK]` : The new method of adding accounts may lead to a separate issue where an account is assigned an amount without setting the corresponding name, since these two operations are executed as two separated functions.

SHB-01 | LACK OF VALIDATION FOR `byBlock`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/SlashingHub.sol: 37	● Resolved

Description

There is no validation for the input `byBlock` in the function `slash()`. The role `VALIDATOR_MANAGER_ROLE` can update this value to a past block, effectively updating the address to no longer be slashed. However, `_timesSlashed` is still increased.

```
30      function slash(address account, uint256 byBlock) public {
31          require(hasRole("VALIDATOR_MANAGER_ROLE", msg.sender), "SlashingHub:
only validator manager has right to perform that");
32          require(
33              IValidatorHub(_getAddressOf("VALIDATOR_HUB")).isValidator(account),
34              "SlashingHub: validator address is not valid"
35          );
36
37          _slashedBy[account] = byBlock;
38          if (_timesSlashed[account] == 0) {
39              _timesSlashed[account] = 1;
40          }
41          _timesSlashed[account] = _timesSlashed[account].mul(2);
42      }
```

Recommendation

We recommend the client add a requirement that the value for `byBlock` must exceed the current `block.number`, or change the logic of `slash()` to reflect the possibility of using a past block number accordingly.

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash 7a950a903a33d3ae78a2a7d61286220b2b28240d.

BFG-16 | UNLOCKED COMPILER VERSION

Category	Severity	Location	Status
Compiler Error	● Informational	contracts/AccessControlHub.sol: 3; contracts/BXP/BXP165.sol: 3; contracts/BXP/BXP20.sol: 3; contracts/BXP/BXP20Asset.sol: 3; contracts/BXP/BXP721.sol: 3; contracts/BXP/BXP721Enumerable.sol: 4; contracts/BXP/WBXN.sol: 3; contracts/BXP/interfaces/IBXP165.sol: 4; contracts/BXP/interfaces/IBXP20.sol: 3; contracts/BXP/interfaces/IBXP20Metadata.sol: 3; contracts/BXP/interfaces/IBXP721.sol: 3; contracts/BXP/interfaces/IBXP721Enumerable.sol: 3; contracts/BXP/interfaces/IBXP721Metadata.sol: 3; contracts/BXP/interfaces/IBXP721Receiver.sol: 3; contracts/CandidateHub.sol: 3; contracts/DelegatorHub.sol: 3; contracts/NodeHub.sol: 3; contracts/PollHub.sol: 3; contracts/SlashingHub.sol: 3; contracts/System.sol: 3; contracts/ValidatorHub.sol: 3; contracts/VoteHub.sol: 3; contracts/extensions/BXP20SystemRewardToken.sol: 3; contracts/extensions/ExtendedMath.sol: 3; contracts/extensions/SystemAccess.sol: 3; contracts/interfaces/IAccessControlHub.sol: 3; contracts/interfaces/IBXP20SystemRewardToken.sol: 3; contracts/interfaces/IDelegatorHub.sol: 3; contracts/interfaces/INodeHub.sol: 3; contracts/interfaces/ISlashingHub.sol: 3; contracts/interfaces/ISystem.sol: 3; contracts/interfaces/IValidatorHub.sol: 3; contracts/interfaces/IVoteHub.sol: 3	● Resolved

Description

The contracts listed have an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Moreover, the lowest compiler version declared is, at the time of this report, the newest compiler version available. Using the most recent compiler version may expose the contracts to unforeseen bugs not yet found in this compiler version.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

We additionally recommend considering the use of a compiler version lower than `v0.8.15`.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 2e67d871e7f4456e2cfe8ea90ec6a878807fe957.

BFG-17 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Language Specific	● Informational	contracts/AccessControlHub.sol: 31; contracts/CandidateHub.sol: 39, 50, 65; contracts/DelegatorHub.sol: 32, 42; contracts/NodeHub.sol: 79, 191, 197; contracts/PollHub.sol: 73, 79, 84, 89, 94, 135; contracts/SlashingHub.sol: 30; contracts/ValidatorHub.sol: 70, 77	● Resolved

Description

One or more state changes do not emit events to pass the changes out of chain.

Recommendation

We recommend declaring and emitting corresponding events for all the essential state variables that can possibly be changed during runtime.

Alleviation

`[CertiK]`: The team heeded the recommendation and made the changes outlined above in commit d5706ad2c7550f481b9f480af76e20e2da57fbf1.

BXA-03 | `burn()` AND `destroy()` HAVE THE SAME INTENDED UTILITY FOR TWO DISTINCT PARTIES

Category	Severity	Location	Status
Coding Style, Inconsistency	● Informational	contracts/BXP/BXP20Asset.sol: 44~45, 75~76	● Resolved

Description

The contract `BXP20Asset` inherits from both the contract `Manageable` and the contract `BlackList`. Through `BlackList`, the `BXP20Asset` contract has the function `destroyFunds()` available which the role `ASSET_BLACKLIST_MANAGER_ROLE` can call to burn the entire balance of a given address, for addresses that are included on the blacklist. Alternatively, the function `burn()` in the `BXP20Asset` contract allows the `ASSET_MANAGER_ROLE` to burn tokens for any address that is *not* on the blacklist. In this way, any address participating can have their tokens burned by a privileged role. If this is the case, it appears the modifier checking whether a user is in or is not in a blacklist is not needed.

Recommendation

We recommend clarifying this choice of design.

Alleviation

`[BlackFort Group]`: "We used same concept used in USDT token and we'd prefer to keep regular burn separately from destroying funds which were laundred for example. Regular burn will be used for exchanging tokenized asset (for example our wrapped BTC called BxBTC to native one)."

CHB-01 | LOGIC ISSUE WHEN ADDING USERS TO CANDIDATES

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/CandidateHub.sol: 25~37	● Partially Resolved

Description

A user can send the `requiredAmount` of BXN to the contract `CandidateHub` to become a candidate. Since the function `receive()` only checks the identity of a validator, candidates can trigger this function more than once. Although the validation on sent value `amount` is to check the cumulative total amount sent from a user, `_candidatesBonds[account]` is actually equal to 0 when a user triggers the function for the first time, meaning that a user has to pay at least the `requiredAmount` of BXN the first time. After that, they can send any amount they want to increase their bonds while adding a user to a candidate would be performed multiple times with `false` returned each time.

We speculate that this validation is supposed to allow the user to send tokens multiple times before becoming a candidate until the required number is reached at which point, they will be added to the candidates set.

```
  25      receive() external payable {
  26          address account = msg.sender;
  27          uint256 amount = msg.value;
  28          IValidatorHub ValidatorHub =
IValidatorHub(_getAddressOf("VALIDATOR_HUB"));
  29
  30          require(!ValidatorHub.isValidator(msg.sender), "CandidateHub: you're
already validator");
  31          require(
  32              _candidatesBonds[account].add(amount) >= requiredAmount,
  33              "CandidateHub: you don't have enough amount of tokens to become
candidate"
  34          );
  35          _candidates.add(account);
  36          _candidatesBonds[account] = _candidatesBonds[account].add(amount);
  37      }
```

Recommendation

We recommend the client re-examine the function `receive()` and change the logic accordingly.

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash d03278d963a712c7dfba2189d2d5f4390a481ae3.

`[CertiK]` : The function `receive()` still requires the user to send `requiredAmount` the first time. We encourage the team to consider whether the functionality of allowing the user to execute `receive()` multiple times after becoming a candidate is an intentional part of the design.

EMB-01 | DEAD CODE

Category	Severity	Location	Status
Coding Style	● Informational	contracts/extensions/ExtendedMath.sol: 7	● Resolved

Description

One or more internal functions are not used.

```
    function sqrt(uint256 x) internal pure returns(uint256) {
```

Recommendation

We recommend removing the unused functions.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 7074b7f14c59c56d1ad187b9a706c3f270d23a9d.

NHB-04 | UNCLEAR IF CONTRACT IS UPGRADEABLE

Category	Severity	Location	Status
Control Flow	● Informational	contracts/NodeHub.sol: 18~19	● Resolved

Description

It is unclear from the context if this contract is meant to be upgradeable and used with a proxy.

Recommendation

We recommend the client clarify if the intention is to use this contract as an implementation contract with a proxy contract.

Alleviation

`[BlackFort]` : "One-time setup. We put our contracts in genesis-block, so we can't run their constructors, we have to run init() methods for some contracts that require such action."

PHB-08 | `mint()` FUNCTION DOES NOT TAKE A FEE

Category	Severity	Location	Status
Language Specific	● Informational	contracts/PollHub.sol: 100~101	● Resolved

Description

The function `mint()` in `PollHub` checks that the `msg.sender` has the required amount of `VoteHub` tokens, and that they also have hold the required amount of `BXN` , however, these amounts of tokens are not withdrawn from the `msg.sender` during the execution of `mint()` . Please clarify whether this is the intention, that is that the amount of participation in the project is measured by the amount of `BXN` and `VoteHub` tokens owned by the address.

Recommendation

We recommend considering whether the `requiredAmountOfBXN` and `requiredAmountOfVote` should be withdrawn during execution of the `mint()` function.

Alleviation

`[BlackFort Group]` : "You pay with BXN only when you want to start the poll, Vote amount is just a requirement to have certain minimum of it.If we do payment in mint, we'd need to refund the payment if user decides to burn his token without starting the poll. In our case we give possibility to create polls for everyone, but you have to pay for start.

It's designed in the way that you can properly prepare your poll, before releasing it. VoteHub token used only to make minting available for users with necessary amount. Fee is taken only when poll is published."

SAB-04 | DECLARATION NAMING CONVENTION

Category	Severity	Location	Status
Coding Style	● Informational	contracts/extensions/SystemAccess.sol: 12, 47	● Resolved

Description

One or more declarations do not conform to the Solidity style guide with regards to its naming convention.

Particularly:

- `camelCase` : Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE` : Should be applied to `constant` variables
- `CapWords` : Should be applied to contract names, struct names, event names and enums

```
12      function set_SYSTEM_CONTRACT_ADDRESS(address name) public {
```

- Function `set_SYSTEM_CONTRACT_ADDRESS` is not in `camelCase` .

```
47      function TEST_setSystemContract(address addr) public {
```

- Function `TEST_setSystemContract` is not in `camelCase` .

Recommendation

We recommend adjusting those function names to properly conform to Solidity's naming convention.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash c3ecf3f8db79720bdd1a967a079b77eb38a7903c.

SAB-05 | FUNCTION `set_SYSTEM_CONTRACT_ADDRESS` DEFINED BEFORE MODIFIERS

Category	Severity	Location	Status
Coding Style	● Informational	contracts/extensions/SystemAccess.sol: 12~13	● Resolved

Description

For improved readability, functions should be defined after modifiers, conforming to the solidity style guide.

Recommendation

We recommend defining `set_SYSTEM_CONTRACT_ADDRESS()` after the modifiers are defined. Refer to the style guide for more information: https://docs.soliditylang.org/en/v0.8.15/style-guide.html#order-of-layout

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash c3ecf3f8db79720bdd1a967a079b77eb38a7903c.

SBF-02 | NO REFUND IF CALLER IS NOT VALIDATOR

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/System.sol: 61~63	● Resolved

Description

The contract `System` has a `receive()` function, so anyone can send BXN to this contract. For the validator user, the sent amount will be used to mint tokens in ValidatorHub and DelegatorHub. But for the non-validator user, the sent amount will be locked in this contract without refund.

Recommendation

We recommend the client re-examine this function and clarify whether the tokens sent from non-validator users should be refunded.

Alleviation

`[CertiK]` : "Please see the team's explanation of the design choice below."

`[BlackFort Group]` : "We have fixed supply so it would be better to collect "burned" tokens back to main storage which System is. My personal thought.

The idea is only that you can recycle them to System, not for 0x00..00 or 0xff..fff address, So they stay in System storage, but any time they can return back to the available supply."

SBF-03 | RACE CONDITION FOR THIRD PARTY ADDRESSES

Category	Severity	Location	Status
Control Flow	● Informational	contracts/System.sol: 34~35	● Resolved

Description

If external third party addresses are given an `_approvedAmount` via the `approve()` function, the external address can front run any changes made to the `_approvedAmount` by spending the previous approved amount before the update to the approved amount goes into effect.

Recommendation

We recommend a function is implemented to increase and decrease the `_approvedAmount` if approval is meant to be given to external contracts or addresses that are not controlled by the client.

Alleviation

`[CertiK]` : Since this functionality can only be executed by in-house contracts and privileged roles, the finding is considered resolved.

`[BlackFort Group]` : "Approval for System used only by our addresses and contracts and controlled by us, thought we might need change values there if anything like we integrate new contract for example"

SHB-02 | `_timesSlashed` UPDATED TO 2 THE FIRST TIME `slash()` CALLED

Category	Severity	Location	Status
Inconsistency	● Informational	contracts/SlashingHub.sol: 41~42	● Resolved

Description

When `slash()` is called for an address the first time, `_timesSlashed` is updated to value 2, instead of 1.

Recommendation

We recommend clarifying if this is the intended effect. If it is, no action is needed and this finding may be removed. Otherwise, please update the logic to reflect the correct value for `_timesSlashed` each time `slash()` is called.

Alleviation

`[CertiK]` : The team clarifies their choice of design below.

`[BlackFort Group]` : " `timesSlashed()` returns value by which reward is divided by. First time reward is divided by 2, which requires us to set 1 at first method call instead of 0."

VHF-01 | VALIDATORS CAN SET THEIR OWN COMMISSION

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/ValidatorHub.sol: 54	● Resolved

Description

When the Candidate is added to the `_validators` set in the function `join()` , the state variable `_validatorCommission[account]` is set to `100` by default. After, a validator can set their own commission to any value by the function `setCommission()` .

Recommendation

We recommend clarifying whether the Validators themselves should have the permission to set their own commission or only the `VALIDATOR_MANAGER_ROLE` should have this ability.

Alleviation

`[BlackFort Group]` : Validators set commission by themselves. Default equals to 10% from reward. Delegators can select validators which offer best conditions or whom they personally support.

Default should be 1000 which means 10%. Fixed in commit ed09661ece8f45ade62b5ae1f228249c05666249.

OPTIMIZATIONS | BLACKFORT GROUP

ID	Title	Category	Severity	Status
BFG-13	Improper Usage Of `public` And `external` Type	Gas Optimization	Optimization	● Resolved
BFG-14	Unnecessary Use Of SafeMath	Gas Optimization	Optimization	● Resolved
BFG-15	Non-Adherence To `AccessControl` Instructions	Gas Optimization, Control Flow	Optimization	● Acknowledged
BXA-02	Multiple Checks An Address Is Not In Blacklist	Gas Optimization	Optimization	● Resolved
DHB-02	Unused State Variable	Gas Optimization	Optimization	● Resolved
NHB-03	`constructor` Sets `_initialized` To `true` On Deploy	Gas Optimization	Optimization	● Resolved
PHB-07	Modifier `pollExists` Checked Twice In Function Call	Gas Optimization	Optimization	● Resolved

BFG-13 | IMPROPER USAGE OF `public` AND `external` TYPE

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/AccessControlHub.sol: 31, 44, 51; contracts/BXP/BXP20Asset.sol: 30, 37, 44, 56, 61, 71, 75; contracts/BXP/WBXN.sol: 13, 17; contracts/CandidateHub.sol: 39, 50, 65; contracts/DelegatorHub.sol: 32, 42, 76, 82; contracts/NodeHub.sol: 191, 197, 203; contracts/PollHub.sol: 79, 84, 89, 94, 100, 110, 130, 135, 147, 159, 168; contracts/SlashingHub.sol: 30; contracts/System.sol: 34; contracts/ValidatorHub.sol: 54, 70, 77; contracts/VoteHub.sol: 24, 28, 32, 45; contracts/extensions/BXP20SystemRewardToken.sol: 37, 41, 45; contracts/extensions/SystemAccess.sol: 12, 47	● Resolved

Description

The functions which are never called internally within the contract should have external visibility for gas optimization.

Recommendation

We recommend the client use the external attribute for public functions that are never called within the contract.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 2606185f7ed1952f25804b5da280546db8e28b77.

BFG-14 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/CandidateHub.sol: 31~34, 36, 46, 56; contracts/DelegatorHub.sol: 16~17, 38, 39, 50, 62, 71, 79; contracts/NodeHub.sol: 21, 109, 110, 111, 113, 119, 130, 140, 169, 177, 179, 180, 187, 228; contracts/PollHub.sol: 15, 171, 173, 177, 178; contracts/SlashingHub.sol: 41; contracts/System.sol: 69, 70, 71, 72, 73, 83, 86; contracts/ValidatorHub.sol: 14~15, 37, 49, 73, 82; contracts/extensions/BXP20SystemRewardToken.sol: 13~14, 30, 55, 56, 85, 94, 98	● Resolved

Description

The `SafeMath` library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow.

```
12      using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `CandidateHub` contract.

```
31          require(
32              _candidatesBonds[account].add(amount) >= requiredAmount,
33              "CandidateHub: you don't have enough amount of tokens to become 
candidate"
34          );
```

- `SafeMath.add` is called in `receive` function of `CandidateHub` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 5) are shown above.

```
13      using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `BXP20SystemRewardToken` contract.

```
38          _delegatedAmount[validatorAccount] = 
_delegatedAmount[validatorAccount].add(rewardShareAmount);
```

- `SafeMath.add` is called in `increaseDelegatedAmountFor` function of `DelegatorHub` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 10) are shown above.

```
using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `PollHub` contract.

```
            reward = reward.add(blockReward.mul(REWARD_HALVING_AFTER_BLOCKS));
```

- `SafeMath.mul` is called in `mintedWith` function of `NodeHub` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 17) are shown above.

```
using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `PollHub` contract.

Note: Only a sample of 1 `SafeMath` library usage in this contract (out of 2) are shown above.

```
using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `PollHub` contract.

```
        uint256 fee = amountOfVote.mul(pollCreatorFee).div(10000);
```

- `SafeMath.mul` is called in `vote` function of `PollHub` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 7) are shown above.

```
using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `SlashingHub` contract.

```
        _timesSlashed[account] = _timesSlashed[account].mul(2);
```

- `SafeMath.mul` is called in `slash` function of `SlashingHub` contract.

```
    using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `System` contract.

```
        uint256 validatorBondedReward =
amount.mul(selfBonded).div(SafeMath.add(selfBonded, delegated));
```

- `SafeMath.mul` is called in `receive` function of `System` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 11) are shown above.

```
    using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `BXP20SystemRewardToken` contract.

```
        _selfBondedAmount[msg.sender] =
_selfBondedAmount[msg.sender].add(msg.value);
```

- `SafeMath.add` is called in `receive` function of `ValidatorHub` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 7) are shown above.

```
    using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `BXP20SystemRewardToken` contract.

```
        return SafeMath.sub((uint256)((int256)(mintedBy(account)) +
_balances[account]), burnedBy(account));
```

- `SafeMath.sub` is called in `balanceOf` function of `BXP20SystemRewardToken` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 9) are shown above.

Recommendation

We recommend removing the usage of `SafeMath` library and using the built-in arithmetic operations provided by the Solidity programming language.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 842633cc720448c2cc25dbbd7d23649d39c49832.

BFG-15 | NON-ADHERENCE TO `AccessControl` INSTRUCTIONS

Category	Severity	Location	Status
Gas Optimization, Control Flow	● Optimization	contracts/AccessControlHub.sol: 40~41; contracts/extensions/SystemAccess.sol: 44~45	● Acknowledged

Description

The method for checking an address has a role is set up to use a string input in the function `hasRole()` within `SystemAccess` . This function calls into `AccessControlHub` which inherits from OpenZeppelin's `AccessControl` contract. The function `hasStringRole()` in `AccessControlHub` calls `hasRole()` from `AccessControl` by first taking the string input `role` , converting it to bytes, then applying `keccak256` hash to the outcome.

This method is inefficient since this conversion must take place each time a privileged role is checked in a contract that inherits from `SystemAccess` . Worse, there could be unforeseen vulnerabilities as a result of bypassing the instructions for set up in `AccessControl` .

Recommendation

We recommend the client follow the outline for set up in the `AccessControl` base contract to ensure gas optimization and security for privileged functions. This includes setting up roles as public constants within the derived contract as follows:

```solidity
bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
```

Alleviation

`[CertiK]` : The team acknowledges the finding and opts to make no change.

`[BlackFort Group]` : "The idea of AccessControlHub is to manage roles across differentcontracts in one place without a mess. Also it makes easier to do role checks in the code while development"

BXA-02 | MULTIPLE CHECKS AN ADDRESS IS NOT IN BLACKLIST

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/BXP/BXP20Asset.sol: 71, 75, 83	● Resolved

Description

The hook `_beforeTokenTransfer()` is called inside transfers, mints, and burns of the token asset. Thus, when `mint()` or `burn()` is called for this function, the `notInBlackList()` modifier is checked twice for the `to` address and `from` address respectively.

Recommendation

We recommend removing the modifier `notInBlackList(account)` on the `mint()` and `burn()` functions since it is checked within the hook in the internal functions.

Alleviation

`[BlackFort Group]`: Issue acknowledged. Changes have been reflected in the commit hash a61e9bc0c2599257744398b682d0d695f448103b.

DHB-02 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/DelegatorHub.sol: 22	● Resolved

Description

Variable `_delegatorShares` in `DelegatorHub` is never used in `DelegatorHub` .

```
22      mapping (address => uint256) private _delegatorShares;
```

```
15  contract DelegatorHub is IDelegatorHub, BXP20SystemRewardToken {
```

Recommendation

We recommend the client remove the unused variables.

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash ffbfafacec44687b93b32db4e1c3c85fb324fd81.

NHB-03 | `constructor` SETS `_initialized` TO `true` ON DEPLOY

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/NodeHub.sol: 54~55, 59~60, 67~68	● Resolved

Description

On deployment of the contract, the function `init()` is called (which can only be executed when `_initialized = false`) and updates the storage variable `_initialized` to `true` . This storage variable cannot be updated after deployment. As such, there is no period of time after deployment of the contract in which `_initialized` is false, making the check from the modifier `isInitialized` unnecessary.

Recommendation

We recommend removing the modifier to optimize the code. Moreover, since `init()` is called in the `constructor` and cannot be called again, we recommend moving the function logic of `init()` to the `constructor` and removing the `init()` function.

Alleviation

`[CertiK]` : See the team's explanation of the design choice below.

`[BlackFort Group]` : "Constructor is reqiured for development purposes, while init() is used for initialization on chain when contract is deployed in genesis-block."

PHB-07 | MODIFIER `pollExists` CHECKED TWICE IN FUNCTION CALL

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/PollHub.sol: 122~123	● Resolved

Description

The modifier `pollExists` is checked twice: once directly in the function `optionOfPollByIndex` and a second time within the modifier `optionExists` when it calls `optionsCountOf()` .

Recommendation

We recommend removing the modifier `pollExists` from the function `optionOfPollByIndex()` since it will be checked in the modifier `optionExists` .

Alleviation

`[BlackFort Group]` : Issue acknowledged. Changes have been reflected in the commit hash 300c71671f1cfeeb408b4b2b4489ee35d1315e06.

FORMAL VERIFICATION | BLACKFORT GROUP

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

Verification of ERC-20 compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply` , which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-recipient-overflow	Function `transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-never-return-false	Function `transfer` Never Returns `false`
erc20-transfer-false	If Function `transfer` Returns `false` , the Contract State Has Not Been Changed
erc20-transferfrom-revert-to-zero	Function `transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-revert-from-zero	Function `transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-succeed-normal	Function `transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	Function `transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-amount-self	Function `transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-correct-amount	Function `transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-allowance	Function `transferFrom` Updated the Allowance Correctly
erc20-transferfrom-change-state	Function `transferFrom` Has No Unexpected State Changes

Property Name	Title

erc20-transferfrom-fail-exceed-allowance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-recipient-overflow	Function `transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-false	If Function `transferFrom` Returns `false` , the Contract's State Has Not Been Changed
erc20-transferfrom-never-return-false	Function `transferFrom` Never Returns `false`
erc20-totalsupply-succeed-always	Function `totalSupply` Always Succeeds
erc20-totalsupply-correct-value	Function `totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	Function `totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	Function `balanceOf` Always Succeeds
erc20-balanceof-correct-value	Function `balanceOf` Returns the Correct Value
erc20-balanceof-change-state	Function `balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	Function `allowance` Always Succeeds
erc20-allowance-correct-value	Function `allowance` Returns Correct Value
erc20-allowance-change-state	Function `allowance` Does Not Change the Contract's State
erc20-approve-revert-zero	Function `approve` Prevents Giving Approvals For the Zero Address
erc20-approve-succeed-normal	Function `approve` Succeeds for Admissible Inputs
erc20-approve-correct-amount	Function `approve` Updates the Approval Mapping Correctly
erc20-approve-change-state	Function `approve` Has No Unexpected State Changes
erc20-approve-false	If Function `approve` Returns `false` , the Contract's State Has Not Been Changed
erc20-approve-never-return-false	Function `approve` Never Returns `false`
erc20-transfer-revert-zero	Function `transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	Function `transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	Function `transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount	Function `transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	Function `transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-change-state	Function `transfer` Has No Unexpected State Changes
erc20-transfer-exceed-balance	Function `transfer` Fails if Requested Amount Exceeds Available Balance

Verification Results

For the following contracts, model checking established that each of the 38 properties that were in scope of this audit (see scope) are valid:

Contract DelegatorHub (Source File contracts/DelegatorHub.sol)

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-false	● True	
erc20-approve-change-state	● True	
erc20-approve-never-return-false	● True	

Contract ValidatorHub (Source File contracts/ValidatorHub.sol)

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract BXP20SystemRewardToken (Source File contracts/extensions/BXP20SystemRewardToken.sol)

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract BXP20 (Source File contracts/BXP/BXP20.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-change-state	● True	
erc20-allowance-correct-value	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-revert-zero	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract WBXN (Source File contracts/BXP/WBXN.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-change-state	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-false	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
  - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".

  - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.

- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
  - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
  - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Contract BXP20Asset (Source File contracts/BXP/BXP20Asset.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract BXP721 (Source File contracts/BXP/BXP721.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inapplicable	
erc20-transfer-succeed-normal	● Inapplicable	
erc20-transfer-succeed-self	● Inapplicable	
erc20-transfer-correct-amount	● Inapplicable	
erc20-transfer-correct-amount-self	● Inapplicable	
erc20-transfer-change-state	● Inapplicable	
erc20-transfer-exceed-balance	● Inapplicable	
erc20-transfer-recipient-overflow	● Inapplicable	
erc20-transfer-false	● Inapplicable	
erc20-transfer-never-return-false	● Inapplicable	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-succeed-normal	● Inapplicable	
erc20-transferfrom-succeed-self	● Inapplicable	
erc20-transferfrom-correct-allowance	● Inapplicable	
erc20-transferfrom-change-state	● Inapplicable	
erc20-transferfrom-fail-exceed-allowance	● Inapplicable	
erc20-transferfrom-false	● Inapplicable	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● Inapplicable	
erc20-totalsupply-correct-value	● Inapplicable	
erc20-totalsupply-change-state	● Inapplicable	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-change-state	● Inapplicable	
erc20-balanceof-correct-value	● True	
erc20-balanceof-succeed-always	● Inapplicable	Intended behavior

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● Inapplicable	
erc20-allowance-correct-value	● Inapplicable	
erc20-allowance-change-state	● Inapplicable	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-correct-amount	● Inapplicable	
erc20-approve-change-state	● Inapplicable	
erc20-approve-false	● Inapplicable	

Contract BXP721Enumerable (Source File contracts/BXP/BXP721Enumerable.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inapplicable	
erc20-transfer-succeed-normal	● Inapplicable	
erc20-transfer-succeed-self	● Inapplicable	
erc20-transfer-correct-amount	● Inapplicable	
erc20-transfer-correct-amount-self	● Inapplicable	
erc20-transfer-change-state	● Inapplicable	
erc20-transfer-exceed-balance	● Inapplicable	
erc20-transfer-recipient-overflow	● Inapplicable	
erc20-transfer-false	● Inapplicable	
erc20-transfer-never-return-false	● Inapplicable	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-succeed-normal	● Inapplicable	
erc20-transferfrom-succeed-self	● Inapplicable	
erc20-transferfrom-correct-allowance	● Inapplicable	
erc20-transferfrom-change-state	● Inapplicable	
erc20-transferfrom-fail-exceed-allowance	● Inapplicable	
erc20-transferfrom-false	● Inapplicable	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-correct-value	● Inapplicable	
erc20-totalsupply-change-state	● Inapplicable	
erc20-totalsupply-succeed-always	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-change-state	● Inapplicable	
erc20-balanceof-correct-value	● True	
erc20-balanceof-succeed-always	● Inapplicable	Intended behavior

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● Inapplicable	
erc20-allowance-correct-value	● Inapplicable	
erc20-allowance-change-state	● Inapplicable	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-correct-amount	● Inapplicable	
erc20-approve-change-state	● Inapplicable	
erc20-approve-false	● Inapplicable	

Contract PollHub (Source File contracts/PollHub.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inapplicable	
erc20-transfer-succeed-normal	● Inapplicable	
erc20-transfer-succeed-self	● Inapplicable	
erc20-transfer-correct-amount	● Inapplicable	
erc20-transfer-correct-amount-self	● Inapplicable	
erc20-transfer-change-state	● Inapplicable	
erc20-transfer-exceed-balance	● Inapplicable	
erc20-transfer-recipient-overflow	● Inapplicable	
erc20-transfer-false	● Inapplicable	
erc20-transfer-never-return-false	● Inapplicable	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-succeed-normal	● Inapplicable	
erc20-transferfrom-succeed-self	● Inapplicable	
erc20-transferfrom-correct-allowance	● Inapplicable	
erc20-transferfrom-change-state	● Inapplicable	
erc20-transferfrom-fail-exceed-allowance	● Inapplicable	
erc20-transferfrom-false	● Inapplicable	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-correct-value	● Inapplicable	
erc20-totalsupply-change-state	● Inapplicable	
erc20-totalsupply-succeed-always	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-change-state	● Inapplicable	
erc20-balanceof-correct-value	● True	
erc20-balanceof-succeed-always	● Inapplicable	Intended behavior

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● Inapplicable	
erc20-allowance-correct-value	● Inapplicable	
erc20-allowance-change-state	● Inapplicable	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-correct-amount	● Inapplicable	
erc20-approve-change-state	● Inapplicable	
erc20-approve-false	● Inapplicable	

Contract BlackList (Source File contracts/BXP/BXP20Asset.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract Manageable (Source File contracts/BXP/BXP20Asset.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

APPENDIX | BLACKFORT GROUP

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Data Flow	Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.
Compiler Error	Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.