BlucamonFactory & Distribution

Smart Contract Audit Report
Prepared for Bluca

Date Issued:	Jan 17, 2022
Project ID:	AUDIT2021047
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Public

Report Information

Project ID	AUDIT2021047
Version	v1.0
Client	Bluca
Project	BlucamonFactory & Distribution
Auditor(s)	Weerawat Pawanawiwat Natsasit Jirathammanuwat Puttimet Thammasaeng
Author(s)	Natsasit Jirathammanuwat Puttimet Thammasaeng
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Jan 17, 2022	Full report	Natsasit Jirathammanuwat Puttimet Thammasaeng

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Bluca, Inspex team conducted an audit to verify the security posture of the BlucamonFactory & Distribution smart contracts between Jan 5, 2022 and Jan 6, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of BlucamonFactory & Distribution smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 1 high, 3 medium, 1 low, 2 very low and 4 info-severity issues. With the project team's prompt response, 1 high, 3 medium, 1 low, 2 very low and 3 info-severity issues were resolved or mitigated in the reassessment, while 1 info-severity issue was acknowledged by the team. Therefore, Inspex trusts that BlucamonFactory & Distribution smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Bluca is a blockchain-based game that combines fun gamification and the quality of the NFT. At Bluca, players can enjoy collecting varieties of Blucamon, breeding, evolving, and exploring the world. A lot of unique NFT can be found along the journey.

BlucamonFactory & Distribution smart contracts are responsible for the distribution of Blucamon NFT to the users via the airdrop, exclusive sale, and standard sale campaigns. These contracts also provide functionalities to summon a Blucamon NFT from an egg to obtain the elemental and rarity.

Scope Information:

Project Name	BlucamonFactory & Distribution
Website	https://bluca.io/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Jan 5, 2022 - Jan 6, 2022
Reassessment Date	Jan 14, 2022

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 5ef5e677b35740bc8685e3877a80809b26f7fced)

Contract	Location (URL)
BlucamonOwnership	https://github.com/BlucaOrganization/public-smart-contracts/blob/5ef5e677b3/contracts/BlucamonOwnership.sol
BlucamonFactory	https://github.com/BlucaOrganization/public-smart-contracts/blob/5ef5e677b3/contracts/BlucamonFactory.sol
BlucamonAirdrop	https://github.com/BlucaOrganization/public-smart-contracts/blob/5ef5e677b3/contracts/BlucamonAirdrop.sol
BlucaDependency	https://github.com/BlucaOrganization/public-smart-contracts/blob/5ef5e677b3/contracts/BlucaDependency.sol
BlucamonSummoning	https://github.com/BlucaOrganization/public-smart-contracts/blob/5ef5e677b3/contracts/BlucamonSummoning.sol
StandardSale	https://github.com/BlucaOrganization/public-smart-contracts/blob/5ef5e677b3/contracts/StandardSale.sol
ExclusiveSale	https://github.com/BlucaOrganization/public-smart-contracts/blob/5ef5e677b3/contracts/ExclusiveSale.sol

Reassessment: (Commit: e09a5ee3276c22b5bfa1cd36fb3e07c49b6e0965)

Contract	Location (URL)
BlucamonOwnership	https://github.com/BlucaOrganization/public-smart-contracts/blob/e09a5ee327/contracts/BlucamonOwnership.sol
BlucamonFactory	https://github.com/BlucaOrganization/public-smart-contracts/blob/e09a5ee327/contracts/BlucamonFactory.sol
BlucamonAirdrop	https://github.com/BlucaOrganization/public-smart-contracts/blob/e09a5ee327/contracts/BlucamonAirdrop.sol
BlucaDependency	https://github.com/BlucaOrganization/public-smart-contracts/blob/e09a5ee327/contracts/BlucaDependency.sol
BlucamonSummoning	https://github.com/BlucaOrganization/public-smart-contracts/blob/e09a5ee327/contracts/BlucamonSummoning.sol
StandardSale	https://github.com/BlucaOrganization/public-smart-contracts/blob/e09a5ee327/contracts/StandardSale.sol

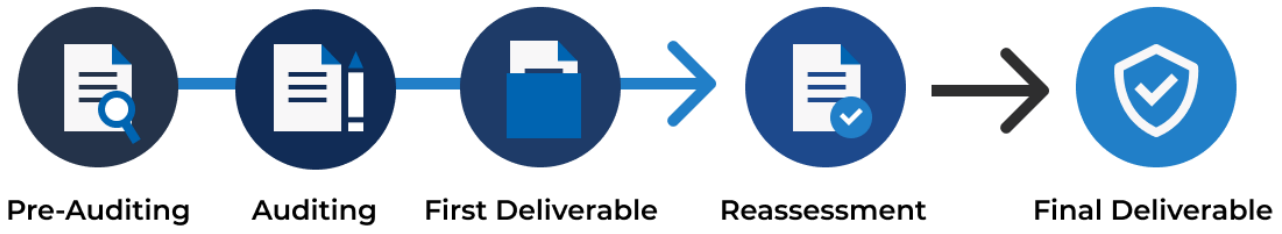
ExclusiveSale	https://github.com/BlucaOrganization/public-smart-contracts/blob/e09a5ee327/contracts/ExclusiveSale.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

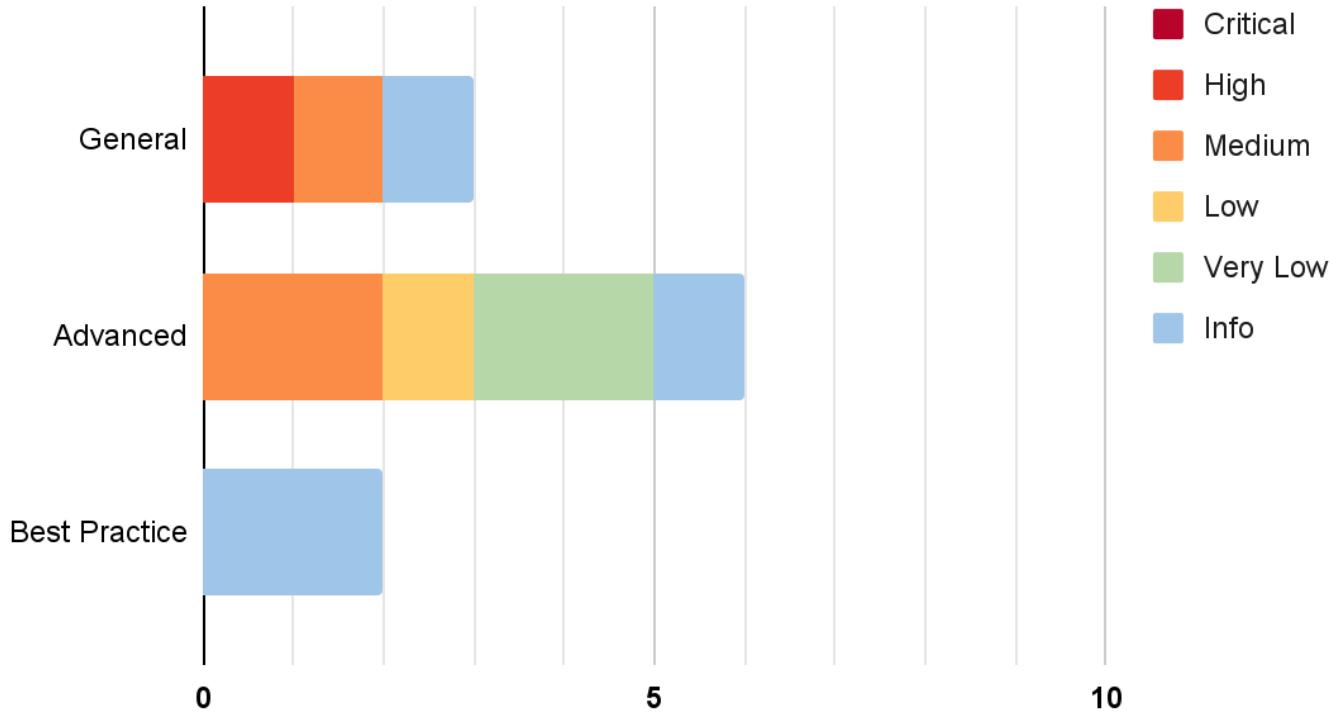
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

4. Summary of Findings

From the assessments, Inspex has found 11 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Centralized Control of State Variable	General	High	Resolved *
IDX-002	Manual Token Minting by Contract Owner	General	Medium	Resolved
IDX-003	Improper Sale Properties Modification During On-Going Sale Event	Advanced	Medium	Resolved
IDX-004	Design Flaw in blucamonId State Usage	Advanced	Medium	Resolved
IDX-005	Improper State Modification of BUSD Contract Address	Advanced	Low	Resolved
IDX-006	Design Flaw in ExclusiveSale and StandardSale Contracts	Advanced	Very Low	Resolved
IDX-007	Insufficient Logging for Privileged Functions	Advanced	Very Low	Resolved
IDX-008	Payment of Amount Exceeding the Purchase Price	Advanced	Info	Resolved
IDX-009	Inexplicit Solidity Compiler Version	Best Practice	Info	Resolved
IDX-010	Improper Function Visibility	Best Practice	Info	Resolved
IDX-011	Use of Low-Level Callings	General	Info	No Security Impact

* The mitigations or clarifications by Bluca can be found in Chapter 5.

5. Detailed Findings Information

5.1. Centralized Control of State Variable

ID	IDX-001
Target	BlucaDependency BlucamonAirdrop BlucamonFactory BlucamonSummoning ExclusiveSale StandardSale
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood: Medium** There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner.
Status	Resolved * The Bluca team has confirmed that a `Timelock` contract with 24-hour delay will be deployed and used for the affected roles. At the time of the reassessment, the contracts are not yet deployed. The platform users should confirm that only the `Timelock` contract has the privileged roles before using the platform.

5.1.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

Target	Function	Modifier
BlucaDependency.sol (L:49)	setWhitelistSetter()	onlyWhitelistSetter

BlucaDependency.sol (L:54)	setSpawner()	onlyWhitelistSetter
BlucaDependency.sol (L:61)	setBreeder()	onlyWhitelistSetter
BlucaDependency.sol (L:68)	setAirdropSetter()	onlyWhitelistSetter
BlucaDependency.sol (L:75)	setFounder()	onlyWhitelistSetter
BlucaDependency.sol (L:82)	setSummoner()	onlyWhitelistSetter
BlucamonAirdrop.sol (L:33)	setWhitelist()	onlyAirdropSetter
BlucamonFactory.sol (L:35)	setBlucamonId()	onlySpawner
BlucamonFactory.sol (L:43)	setDefaultElementalFragments()	onlySpawner
BlucamonSummoning.sol (L:20)	setSummonFee()	onlyFounder
ExclusiveSale.sol (L:42)	setSetter()	onlySetter
ExclusiveSale.sol (L:46)	setFounder()	onlySetter
ExclusiveSale.sol (L:50)	setDefaultRarity()	onlySetter
ExclusiveSale.sol (L:54)	setEvent()	onlySetter
ExclusiveSale.sol (L:68)	setCurrentNumber()	onlySetter
ExclusiveSale.sol (L:72)	setPrefixTokenUri()	onlySetter
ExclusiveSale.sol (L:79)	disableEvent()	onlySetter
StandardSale.sol (L:34)	setSetter()	onlySetter
StandardSale.sol (L:38)	setFounder()	onlySetter
StandardSale.sol (L:42)	setEvent()	onlySetter
StandardSale.sol (L:54)	setBusdContract()	onlySetter
StandardSale.sol (L:58)	setCurrentNumber()	onlySetter
StandardSale.sol (L:62)	setPrefixTokenUri()	onlySetter
StandardSale.sol (L:69)	disableEvent()	onlySetter

5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a Timelock contract to delay the changes for a reasonable amount of time, e.g., 24 hours

Please note that if the timelock mechanism is decided to be used, from the discussion with the Bluca team, only the contracts responsible for minting the NFT, which are `StandardSale` and `ExclusiveSale`, will have the `spawner` role. Therefore, the affected functions with the `onlySpawner` modifier should be changed to another role modifier, so that the spawning function callings of the spawner contracts won't be affected by the timelock.

5.2. Manual Token Minting by Contract Owner

ID	IDX-002
Target	BlucamonOwnership
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** Medium The contract owner can arbitrarily mint the Blucamon NFT, but these tokens minted will have no Blucamon metadata in the `blucamons` array like other NFTs minted through the proper means. Furthermore, minting through the `safeMint()` function will make the `mintBlucamon()` function unusable, causing denial of service. **Likelihood:** Medium Only the contract owner can call the `safeMint()` function, and it is possible for the owner to profit from this action.
Status	Resolved The Bluca team has resolved this issue in commit `94b5ad94daa94db4e08f1e9a3646451576765b32` by removing the `safeMint()` function.

5.2.1. Description

The `safeMint()` function in `BlucamonOwnership` contract can be used to arbitrarily mint the Blucamon NFT as shown below.

BlucamonOwnership.sol

```
61 function safeMint(address to, uint256 tokenId) public onlyOwner {
62     _safeMint(to, tokenId);
63 }
```

The address with the `owner` role can mint a Blucamon NFT, but cannot set the metadata for it since the metadata is set when minted through the `spawn()` function.

In addition, the `safeMint()` function call can cause denial of service when the `blucamonId` reaches the `tokenId` of the token minted as in issue `IDX-004 Design Flaw in blucamonId State Usage`.

5.2.2. Remediation

Inspex suggests removing the `safeMint()` function to prevent the token from being minted manually.

5.3. Improper Sale Properties Modification During On-Going Sale Event

ID	IDX-003
Target	ExclusiveSale StandardSale
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** Medium The modification of the sale properties is unfair for the users, since the total number of tokens and the price can be changed from what is known by the users. This results in loss of reputation for the platform and monetary impact for the users. **Likelihood:** Medium Only the setter role can modify the states, and there is benefit for the owner in performing this action, so there is motivation for the attack.
Status	Resolved The Bluca team has resolved this issue in commit `33c64749e9f009727b03d3b87b45a77d10cfd959` by preventing the `setEvent()` function from being used during an on-going sale event.

5.3.1. Description

In the `StandardSale` and `ExclusiveSale` contracts, the sale event can be started using the `setEvent()` function as in the following example from the `StandardSale` contract.

StandardSale.sol

```
42 function setEvent(
43     uint256 _price,
44     uint256 _total,
45     uint256 _startTime,
46     uint256 _endTime
47 ) external onlySetter {
48     price = _price;
49     total = _total;
50     startTime = _startTime;
51     endTime = _endTime;
52 }
```

The users can purchase the Blucamon egg using the `purchaseEgg()` function.

StandardSale.sol

```
73  function purchaseEgg() external {
74      validatePurchasing();
75      (bool transferResult, ) = busdContract.call(
76          abi.encodeWithSignature(
77              "transferFrom(address,address,uint256)",
78              msg.sender,
79              founder,
80              price
81          )
82      );
83      require(transferResult, "S_STD_400");
84      currentNumber = currentNumber.add(1);
85
86      uint256 newBlucamonId = getBlucamonId().add(1);
87      string memory tokenUri = getTokenUri(newBlucamonId);
88      (bool mintResult, ) = blucamonOwnershipContract.call(
89          abi.encodeWithSignature(
90              "mintBlucamon(address,string,bool,uint8,uint256,uint8)",
91              msg.sender,
92              tokenUri,
93              false,
94              8,
95              0,
96              0
97          )
98      );
99      require(mintResult, "S_STD_500");
100     emit PurchaseStandardEgg(newBlucamonId);
101 }
```

The purchase can be done between the `startTime` and `endTime` as checked in the `validatePurchasing()` function.

StandardSale.sol

```
114 function validatePurchasing() private view {
115     require(currentNumber < total, "S_STD_200");
116     require(block.timestamp >= startTime, "S_STD_300");
117     require(block.timestamp < endTime, "S_STD_301");
118 }
```

However, the contract owner can use the `setEvent()` function to change the sale properties, including the egg price, at any time. If this action is maliciously done during the sale, the users can unknowingly buy an egg with an exceedingly high price.

5.3.2. Remediation

Inspex suggests adding conditions to prevent the `setEvent()` function from being used during an on-going sale event, and prevent the new event time from being improperly set by making sure that the current timestamp does not exceed start time, and the start time is before end time, for example:

StandardSale.sol

```
42  function setEvent(
43      uint256 _price,
44      uint256 _total,
45      uint256 _startTime,
46      uint256 _endTime
47  ) external onlySetter {
48      require(
49          block.timestamp < startTime || block.timestamp >= endTime,
50          "Unable to set during sale"
51      );
52      require(
53          block.timestamp <= _startTime && _startTime < _endTime,
54          "Invalid time"
55      );
56      price = _price;
57      total = _total;
58      startTime = _startTime;
59      endTime = _endTime;
60  }
```

ExclusiveSale.sol

```
54  function setEvent(
55      uint8 _season,
56      uint256 _price,
57      uint256 _total,
58      uint256 _startTime,
59      uint256 _endTime
60  ) external onlySetter {
61      require(
62          block.timestamp < startTime || block.timestamp >= endTime,
63          "Unable to set during sale"
64      );
65      require(
66          block.timestamp <= _startTime && _startTime < _endTime,
67          "Invalid time"
68      );
69      season = _season;
70      price = _price;
71      total = _total;
72      startTime = _startTime;
```

```
73      endTime = _endTime;
74  }
```

5.4. Design Flaw in blucamonId State Usage

ID	IDX-004
Target	BlucamonOwnership
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The execution of `spawn()` function will be reverted when the `blucamonId` state overlaps the id of the minted NFT, causing disruption to the service, resulting in loss of reputation for the platform. **Likelihood:** Medium Only the spawner and airdrop setter can call the affected functions, but as the token can be continuously minted through the sale contracts, it is possible that improper id values are used for the tokens.
Status	Resolved The Bluca team has resolved this issue in commit `dc71611e4faf7cf68a0c2544c54470164c4e0ae3` by setting the id using the value from the `blucamonId` state in `mintBlucamons()` and `setWhitelist()` functions instead of inputting the value of id manually.

5.4.1. Description

In the `BlucamonOwnership` contract, the Blucamon NFT can be spawned through the `spawn()` function to set the metadata of the Blucamon NFT and mint the token. The id passed to the `_safeMint()` function cannot be duplicated, or else, the transaction will be reverted.

BlucamonOwnership.sol

```
161 function spawn(
162     uint256 _blucamonId,
163     bool _isSummoned,
164     address _address,
165     string memory _tokenUri,
166     uint8 _rarity,
167     uint256 _blucadexId,
168     uint8 _eggElement
169 ) private {
170     spawnBlucamon(
171         _blucamonId,
172         _isSummoned,
173         _rarity,
```

```
174             _blucadexId,
175             _eggElement
176         );
177         _safeMint(_address, _blucamonId);
178         _setTokenURI(_blucamonId, _tokenUri);
179     }
```

The `spawn()` function is called by the `mintBlucamon()`, `mintBlucamons()`, and `claimBlucamonEgg()` functions.

The `mintBlucamon()` function uses the `blucamonId` state to determine the id of the token as seen in line 149 and 151.

BlucamonOwnership.sol

```
141 function mintBlucamon(
142     address _address,
143     string memory _tokenUri,
144     bool _isSummoned,
145     uint8 _rarity,
146     uint256 _blucadexId,
147     uint8 _eggElement
148 ) external onlySpawner {
149     blucamonId = blucamonId.add(1);
150     spawn(
151         blucamonId,
152         _isSummoned,
153         _address,
154         _tokenUri,
155         _rarity,
156         _blucadexId,
157         _eggElement
158     );
159 }
```

However, the `mintBlucamons()` function uses the id defined from the input parameter, not the `blucamonId` state, as seen in line 199 and 207.

BlucamonOwnership.sol

```
198 function mintBlucamons(
199     uint256[] memory _idList,
200     address[] memory _addressesList,
201     string[] memory _tokenUriList,
202     uint8[] memory _rarityList,
203     uint8[] memory _eggElementList
204 ) external onlySpawner {
205     for (uint256 index = 0; index < _idList.length; index++) {
```

```
206         spawn(
207             _idList[index],
208             false,
209             _addressesList[index],
210             _tokenUriList[index],
211             _rarityList[index],
212             0,
213             _eggElementList[index]
214         );
215         blucamonId = blucamonId.add(1);
216     }
217 }
```

Likewise, the `claimBlucamonEgg()` function uses the id from the egg detail that is inherited from the `BlucamonAirdrop` contract, not the `blucamonId` state, as seen in line 124 and 126.

BlucamonOwnership.sol

```
121 function claimBlucamonEgg() external onlyAirdropWhitelist(msg.sender) {
122     require(!isClaimedMapping[msg.sender], "S_ARD_103");
123     claim(msg.sender);
124     AirdropEggDetail memory eggDetail = getEggDetail();
125     spawn(
126         eggDetail.id,
127         false,
128         msg.sender,
129         eggDetail.tokenUri,
130         eggDetail.rarity,
131         0,
132         0
133     );
134 }
```

The egg detail is set through the `setWhitelist()` function, using the id from the input parameter as seen in line 34, 47, and 61.

BlucamonAirdrop.sol

```
33 function setWhitelist(
34     uint256[] memory _idList,
35     address[] memory _addresses,
36     string[] memory _tokenUriList,
37     uint8[] memory _rarityList
38 ) external onlyAirdropSetter {
39     validateWhitelistParameter(
40         _idList,
41         _addresses,
42         _tokenUriList,
```

```
43          _rarityList
44      );
45      for (uint256 idx = 0; idx < _addresses.length; idx++) {
46          whitelistEggDetail[_addresses[idx]] = setEggDetail(
47              _idList[idx],
48              _tokenUriList[idx],
49              _rarityList[idx]
50          );
51      }
52  }
53
54  function setEggDetail(
55      uint256 _id,
56      string memory _tokenUri,
57      uint8 _rarity
58  ) internal returns (AirdropEggDetail memory) {
59      blucamonId = blucamonId.add(1);
60      return
61          AirdropEggDetail({id: _id, tokenUri: _tokenUri, rarity: _rarity});
62  }
```

This means that if the Blucamon NFT is not spawned through a proper order, the minting function will be unusable for the overlapped id, resulting in denial of service on the contracts that use these functions, including `StandardSale` and `ExclusiveSale` contracts.

To demonstrate the impact, please consider the following scenario:

1. 5 Blucamons have been spawned, so the `blucamonId` state is 5.
2. Whitelist is set using the `setWhitelist()` function from id 11 to 15, causing the token id 11 to 15 to be saved to the eggs, while the `blucamonId` state is incremented to 10.
3. A Blucamon is spawned though the `mintBlucamon()` function, minting a Blucamon NFT with the id 11.
4. This causes the egg with the id 11 to be unclaimable, since the NFT with the id 11 is already minted.

5.4.2. Remediation

Inspex suggests setting the id using the value from the `blucamonId` state in `mintBlucamons()` and `setWhitelist()` functions instead of inputting the value of id manually, for example:

BlucamonOwnership.sol

```
198 function mintBlucamons(
199     address[] memory _addressesList,
200     string[] memory _tokenUriList,
201     uint8[] memory _rarityList,
202     uint8[] memory _eggElementList
203 ) external onlySpawner {
```

```
204     for (uint256 index = 0; index < _idList.length; index++) {
205         blucamonId = blucamonId.add(1);
206         spawn(
207             blucamonId,
208             false,
209             _addressesList[index],
210             _tokenUriList[index],
211             _rarityList[index],
212             0,
213             _eggElementList[index]
214         );
215     }
216 }
```

BlucamonAirdrop.sol

```
33  function setWhitelist(
34      address[] memory _addresses,
35      string[] memory _tokenUriList,
36      uint8[] memory _rarityList
37  ) external onlyAirdropSetter {
38      validateWhitelistParameter(
39          _addresses,
40          _tokenUriList,
41          _rarityList
42      );
43      for (uint256 idx = 0; idx < _addresses.length; idx++) {
44          blucamonId = blucamonId.add(1);
45          whitelistEggDetail[_addresses[idx]] = setEggDetail(
46              blucamonId,
47              _tokenUriList[idx],
48              _rarityList[idx]
49          );
50      }
51  }
```

As the parameter of the `setWhitelist()` function is changed, the `validateWhitelistParameter()` function should also be modified accordingly, for example:

BlucamonAirdrop.sol

```
68  function validateWhitelistParameter(
69      address[] memory _addresses,
70      string[] memory _tokenUriList,
71      uint8[] memory _rarityList
72  ) private pure {
73      uint256 addressCount = _addresses.length;
74      uint256 tokenUriCount = _tokenUriList.length;
75      uint256 rarityCount = _rarityList.length;
76      require(
77          addressCount > 0 &&
78              tokenUriCount > 0 &&
79              rarityCount > 0,
80          "S_ARD_101"
81      );
82      require(
83          addressCount == tokenUriCount &&
84              addressCount == rarityCount,
85          "S_ARD_102"
86      );
87  }
```

5.5. Improper State Modification of BUSD Contract Address

ID	IDX-005
Target	StandardSale
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Low **Impact:** Medium The $BUSD contract address can be changed to any token at any time by the setter role. The user may lose the token other than the $BUSD in order to purchase the egg. This results in loss of reputation for the platform, and monetary impact for the users. **Likelihood:** Low Only the setter role can modify this state, and performing an attack using this flaw is difficult as it requires prior token transfer approval from the users.
Status	Resolved The Bluca team has resolved this issue in commit `857e67ae0e85f9ee0e86d81406afab98646faeb3` by removing the `setBusdContract()` function and setting the `busdContract` state in the constructor.

5.5.1. Description

The `setBusdContract()` function can be called by the setter role to change the $BUSD contract address as shown in the following example from `StandardSale` contract.

StandardSale.sol

```
54 function setBusdContract(address _newAddress) external onlySetter {
55     busdContract = _newAddress;
56 }
```

When the user calls the `purchaseEgg()` function in the `StandardSale` contract, the $BUSD contract will be called to transfer the token from the user to the founder address as follows:

StandardSale.sol

```
73 function purchaseEgg() external {
74     validatePurchasing();
75     (bool transferResult, ) = busdContract.call(
76         abi.encodeWithSignature(
77             "transferFrom(address,address,uint256)",
78             msg.sender,
79             founder,
```

```
80              price
81          )
82      );
83      require(transferResult, "S_STD_400");
84      currentNumber = currentNumber.add(1);
85
86      uint256 newBlucamonId = getBlucamonId().add(1);
87      string memory tokenUri = getTokenUri(newBlucamonId);
88      (bool mintResult, ) = blucamonOwnershipContract.call(
89          abi.encodeWithSignature(
90              "mintBlucamon(address,string,bool,uint8,uint256,uint8)",
91              msg.sender,
92              tokenUri,
93              false,
94              8,
95              0,
96              0
97          )
98      );
99      require(mintResult, "S_STD_500");
100     emit PurchaseStandardEgg(newBlucamonId);
101 }
```

It is possible for this contract to be used for different sale events with different token types. The setter can use the `setBusdContract()` function to change the token address to another token type with higher value during an on-going sale event, and cause the users to unknowingly pay with a different token type and gain more funds from the users.

5.5.2. Remediation

Inspex suggests removing the `setBusdContract()` function and setting the `busdContract` state in the constructor.

StandardSale.sol

```
11 constructor(address _ownershipContractAddress, address _busdContractAddress) {
12     blucamonOwnershipContract = _ownershipContractAddress;
13     setter = msg.sender;
14     busdContract = _busdContractAddress;
15 }
```

5.6. Design Flaw in ExclusiveSale and StandardSale Contracts

ID	IDX-006
Target	ExclusiveSale StandardSale
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Very Low **Impact:** Low The `setCurrentNumber()` function can set `currentNumber` to any number. The number of eggs bought may not match the `currentNumber`, and the total can be exceeded by setting the `currentNumber`. Thus, It can be damaging to the platform's reputation. **Likelihood:** Low Since, the `setCurrentNumber()` function can be called by `onlySetter` role only which is set by the contract owner. Hence, it is a low motivation to attack with this vulnerability as there is no direct profit.
Status	Resolved The Bluca team has resolved this issue in commit `c7e8291ec74c6d098be5045613fbeb0aa31a381b` by removing the `setCurrentNumber()` function and set the `currentNumber` to 0 in the execution of `setEvent()` function.

5.6.1. Description

The `ExclusiveSale` and `StandardSale` contracts have `validatePurchasing()` function to validate the purchasing for each event. For example, in the `ExclusiveSale` contract, the `currentNumber` state is used as a counting state to count the number of purchased eggs to ensure that no more than total eggs are purchased in a sale event.

ExclusiveSale.sol

```
125 function validatePurchasing(uint256 _value) private view {
126     require(_value >= price, "S_EXS_300");
127     require(currentNumber < total, "S_EXS_400");
128     require(block.timestamp >= startTime, "S_EXS_500");
129     require(block.timestamp < endTime, "S_EXS_501");
130 }
```

However, both contracts have the `setCurrentNumber()` function that can be used to set the `currentNumber` state as shown below.

ExclusiveSale.sol

```
58 function setCurrentNumber(uint256 _newNumber) external onlySetter {
59     currentNumber = _newNumber;
60 }
```

Therefore, The address with `onlySetter` role can set `currentNumber` to any number, so the number of eggs bought may not match the `currentNumber`, and the total can be exceeded by setting the `currentNumber` to a lower value.

5.6.2. Remediation

Inspex suggests removing the `setCurrentNumber()` function and set the `currentNumber` to 0 in the execution of `setEvent()` function, for example:

ExclusiveSale.sol

```
54 function setEvent(
55     uint8 _season,
56     uint256 _price,
57     uint256 _total,
58     uint256 _startTime,
59     uint256 _endTime
60 ) external onlySetter {
61     currentNumber = 0;
62     season = _season;
63     price = _price;
64     total = _total;
65     startTime = _startTime;
66     endTime = _endTime;
67 }
```

StandardSale.sol

```
42 function setEvent(
43     uint256 _price,
44     uint256 _total,
45     uint256 _startTime,
46     uint256 _endTime
47 ) external onlySetter {
48     currentNumber = 0;
49     price = _price;
50     total = _total;
51     startTime = _startTime;
52     endTime = _endTime;
53 }
```

5.7. Insufficient Logging for Privileged Functions

ID	IDX-007
Target	BlucaDependency BlucamonAirdrop BlucamonFactory BlucamonOwnership BlucamonSummoning ExclusiveSale StandardSale
Category	Advanced Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved The Bluca team has resolved this issue in commit `a0f987b681d4d45e2d138e88503082c00a8d6e2a` by emitting events for the execution of privileged functions.

5.7.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the owner can set the spawner address by executing the `setSpawner()` function in the `BlucaDependency` contract, and no event is emitted.

BlucaDependency.sol

```
54 function setSpawner(address _spawner, bool _isWhitelisted)
55     external
56     onlyWhitelistSetter
57 {
58     whitelistedSpawner[_spawner] = _isWhitelisted;
59 }
```

The following table shows the privileged functions without any event emitted:

File	Contract	Function
BlucaDependency.sol (L:49)	BlucamonOwnership	setWhitelistSetter()
BlucaDependency.sol (L:54)	BlucamonOwnership	setSpawner()
BlucaDependency.sol (L:61)	BlucamonOwnership	setBreeder()
BlucaDependency.sol (L:68)	BlucamonOwnership	setAirdropSetter()
BlucaDependency.sol (L:75)	BlucamonOwnership	setFounder()
BlucaDependency.sol (L:82)	BlucamonOwnership	setSummoner()
BlucamonAirdrop.sol (L:33)	BlucamonOwnership	setWhitelist()
BlucamonFactory.sol (L:35)	BlucamonOwnership	setBlucamonId()
BlucamonFactory.sol (L:43)	BlucamonOwnership	setDefaultElementalFragments()
BlucamonOwnership.sol (L:61)	BlucamonOwnership	safeMint()
BlucamonOwnership.sol (L:136)	BlucamonOwnership	summon()
BlucamonOwnership.sol (L:141)	BlucamonOwnership	mintBlucamon()
BlucamonOwnership.sol (L:198)	BlucamonOwnership	mintBlucamons()
BlucamonSummoning.sol (L:20)	BlucamonSummoning	setSummonFee()
BlucamonSummoning.sol (L:37)	BlucamonSummoning	transfer()
ExclusiveSale.sol (L:42)	ExclusiveSale	setSetter()
ExclusiveSale.sol (L:46)	ExclusiveSale	setFounder()
ExclusiveSale.sol (L:50)	ExclusiveSale	setDefaultRarity()
ExclusiveSale.sol (L:54)	ExclusiveSale	setEvent()
ExclusiveSale.sol (L:68)	ExclusiveSale	setCurrentNumber()
ExclusiveSale.sol (L:72)	ExclusiveSale	setPrefixTokenUri()
ExclusiveSale.sol (L:79)	ExclusiveSale	disableEvent()
ExclusiveSale.sol (L:83)	ExclusiveSale	transfer()
StandardSale.sol (L:34)	StandardSale	setSetter()

StandardSale.sol (L:38)	StandardSale	setFounder()
StandardSale.sol (L:42)	StandardSale	setEvent()
StandardSale.sol (L:54)	StandardSale	setBusdContract()
StandardSale.sol (L:58)	StandardSale	setCurrentNumber()
StandardSale.sol (L:62)	StandardSale	setPrefixTokenUri()
StandardSale.sol (L:69)	StandardSale	disableEvent()

5.7.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

BlucaDependency.sol

```
54  event SetSpawner(address _spawner, bool _isWhitelisted);
55  function setSpawner(address _spawner, bool _isWhitelisted)
56      external
57      onlyWhitelistSetter
58  {
59      whitelistedSpawner[_spawner] = _isWhitelisted;
60      emit SetSpawner(_spawner, _isWhitelisted);
61  }
```

5.8. Payment of Amount Exceeding the Purchase Price

ID	IDX-008
Target	ExclusiveSale
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Bluca team has resolved this issue in commit `cbc0243b77d1c10b9d0c08fb681b2ad2244d65e8` by verifying that the native token value sent into the contract must be equal to the price.

5.8.1. Description

In the `ExclusiveSale` contract, the Blucamon egg NFT can be bought using native token through the `purchaseEgg()` function.

ExclusiveSale.sol

```
88  function purchaseEgg() external payable {
89      validatePurchasing(msg.value);
90      currentNumber = currentNumber.add(1);
91
92      uint256 newBlucamonId = getBlucamonId().add(1);
93      string memory tokenUri = getTokenUri(newBlucamonId);
94      uint8 rarity = getRarity();
95      (bool result, ) = blucamonOwnershipContract.call(
96          abi.encodeWithSignature(
97              "mintBlucamon(address,string,bool,uint8,uint256,uint8)",
98              msg.sender,
99              tokenUri,
100             false,
101             rarity,
102             0,
103             0
104         )
105     );
106     require(result, "S_EXS_600");
107     emit PurchaseExclusiveEgg(newBlucamonId);
108 }
```

The amount of native token sent into the contract is checked using the `validatePurchasing()` function in line 126, validating that the amount of native token sent into the contract is equal to or higher than the price.

ExclusiveSale.sol

```
125 function validatePurchasing(uint256 _value) private view {
126     require(_value >= price, "S_EXS_300");
127     require(currentNumber < total, "S_EXS_400");
128     require(block.timestamp >= startTime, "S_EXS_500");
129     require(block.timestamp < endTime, "S_EXS_501");
130 }
```

However, unlike in the `StandardSale` contract that transfers only the required amount from the user, more native token can be sent into the `ExclusiveSale` contract than the price, so it is possible for the users to unknowingly send in more native token into the contract than required.

5.8.2. Remediation

Inspex suggests returning the excess amount of native token value in the `purchaseEgg()` function if more native tokens are transferred into the `ExclusiveSale` contract than needed.

5.9. Inexplicit Solidity Compiler Version

ID	IDX-009
Target	BlucamonAirdrop BlucaDependency BlucamonFactory BlucamonOwnership BlucamonSummoning ExclusiveSale StandardSale
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	**Resolved** The Bluca team has resolved this issue in commit `e09a5ee3276c22b5bfa1cd36fb3e07c49b6e0965` by explicitly define the Solidity version to the latest stable version (v0.8.11).

5.9.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues.

Contract Name	Version
BlucamonAirdrop	>=0.7.0 <0.9.0
BlucaDependency	>=0.7.0 <0.9.0
BlucamonFactory	>=0.7.0 <0.9.0
BlucamonOwnership	>=0.7.0 <0.9.0
BlucamonSummoning	>=0.7.0 <0.9.0
ExclusiveSale	>=0.7.0 <0.9.0
StandardSale	>=0.7.0 <0.9.0

5.9.2. Remediation

Inspex suggests fixing the Solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.11[2].

5.10. Improper Function Visibility

ID	IDX-010
Target	BlucamonOwnership
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Bluca team has resolved this issue in commit `94b5ad94daa94db4e08f1e9a3646451576765b32` by removing the `safeMint()` function.

5.10.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

The following source code shows that the `safeMint()` function of the `BlucamonOwnership` contract is set to public and it is never called from any internal function.

BlucamonOwnership.sol

```
61 function safeMint(address to, uint256 tokenId) public onlyOwner {
62     _safeMint(to, tokenId);
63 }
```

5.10.2. Remediation

In this case, Inspex suggests changing the function visibility to external if it is not called from any internal function as shown in the following example:

BlucamonOwnership.sol

```
61 function safeMint(address to, uint256 tokenId) external onlyOwner {
62     _safeMint(to, tokenId);
63 }
```

5.11. Use of Low-Level Callings

ID	IDX-011
Target	BlucamonSummoning ExclusiveSale StandardSale
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact The Bluca team has acknowledged this issue as it affects the business logic and the return value checks are already implemented.

5.11.1. Description

The contracts use low-level function callings to call external contracts as shown below.

BlucamonSummoning.sol

```
24 function summon(uint256 _id) external payable {
25     (, bytes memory ownerData) = blucamonOwnershipContract.call(
26         abi.encodeWithSignature("getBlucamonOwner(uint256)", _id)
27     );
28     address owner = abi.decode(ownerData, (address));
29     require(owner == msg.sender, "S_ONS_100");
30     require(msg.value == summonFee, "S_SMN_101");
31     (bool result, ) = blucamonOwnershipContract.call(
32         abi.encodeWithSignature("summon(uint256)", _id)
33     );
34     require(result, "S_SMN_100");
35 }
```

The return value for a low-level function call is not checked by default, and the execution will not be reverted even when an exception is thrown. Unexpected behaviors can occur if function calling is required to be successful, and the return value is not properly checked. Casting the address to an interface and calling an external contract through the interface helps propagate the exception throws, reducing the risk for improper checking of return value.

The low-level function calls are done in the following locations:

Filename	Contract	Function Name
BlucamonSummoning.sol (L:25, 32)	BlucamonSummoning	summon()
ExclusiveSale.sol (L:96)	ExclusiveSale	purchaseEgg()
ExclusiveSale.sol (L:110)	ExclusiveSale	getBlucamonId()
StandardSale.sol (L:76)	StandardSale	purchaseEgg()
StandardSale.sol (L:89)	StandardSale	purchaseEgg()
StandardSale.sol (L:105)	StandardSale	getBlucamonId()

5.11.2. Remediation

Inspex suggests calling external contracts using the interfaces instead of the low-level function call, for example, calling functions in `BlucamonOwnership` contract can be done as follows:

IBlucamonOwnership.sol

```
1  interface IBlucamonOwnership {
   [...]
2      function getBlucamonOwner(uint256) external view returns (address);
3      function summon(uint256) external;
   [...]
4  }
```

BlucamonSummoning.sol

```
3  import "./IBlucamonOwnership.sol";
4
5  contract BlucamonSummoning {
6      IBlucamonOwnership blucamonOwnershipContract;
7      address payable founder;
8
9      constructor(address _ownershipContractAddress, address _founderAddress) {
10         blucamonOwnershipContract =
   IBlucamonOwnership(_ownershipContractAddress);
11         founder = payable(_founderAddress);
12     }
   [...]
24     function summon(uint256 _id) external payable {
25         address owner = blucamonOwnershipContract.getBlucamonOwner(_id);
26         require(owner == msg.sender, "S_ONS_100");
27         require(msg.value == summonFee, "S_SMN_101");
```

```
28              blucamonOwnershipContract.summon(_id);
29          }
    [...]
41  }
```

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] “Release Version 0.8.11 · ethereum/solidity” [Online]. Available: https://github.com/ethereum/solidity/releases/tag/v0.8.11. [Accessed: 11-January-2022]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE