Jun 19, 2024

ZAN

# Security Assessment InfluencerBadge

—

Professional Service

# Table of Contents

# 1. Overview

## 1.1. Executive Summary

**InfluencerBadge** is a SocialFi project based on ERC1155 standard. This report has been prepared for Halo to discover vulnerabilities and issues in the source code of the project as well as any contract dependencies that were not part of an officially recognized library.

Conducted by Static Analysis, Formal Verification and Manual Review, we have identified **2 low and 7 informational issues** in the commit 4caa4b6.

In the commit e14d0d7, the project team **resolved the issue in L-03**, acknowledged the other issues and decided to keep no change. In this commit, we additionally discovered **1 high and 1 informational issues** in H-01 and I-04 respectively. The project team **acknowledged these two issues and provided solutions to address them**. For details, please refer to the alleviation of these two issues.

## 1.2. Project Summary

| | |
|---|---|
| **Project Name** | InfluencerBadge |
| **Platform** | BSC |
| **Language** | Solidity |
| **Codebase** | Audit 1:<br>• https://github.com/halowalletdev/halo-influencer-badge-contract/tree/4caa4b645991a50d38e75ecb0ec73c8a90ad8698<br>Final Audit:<br>• https://github.com/halowalletdev/halo-influencer-badge-contract/tree/e14d0d7b881f77c26125d26aa35c5563b8f90a63 |

## 1.3. Assessment Summary

| | |
|---|---|
| **Delivery Date** | Jun 19, 2024 |
| **Audit Methodology** | Static Analysis, Formal Verification, Manual Review |

## 1.4. Assessment Scope

| ID | File | File Hash |
|---|---|---|
| 1 | /contracts/InfluencerBadge.sol | 5f10e1e36dc65026dda2722ad5610df9 |
| 2 | /contracts/interfaces/IERC20WithDecimals.sol | 4c61194936cf51a0ba9d04247c0af5a9 |
| 3 | /contracts/interfaces/EventsAndErrors.sol | f120e0aa17b14b617c6ddb492066b64a |

| ID | File | File Hash |
| --- | --- | --- |
| 4 | /contracts/interfaces/IHaloMembershipPass.sol | 5b07c8577a14704fa4456388041b525a |

# 2. Checklist

## 2.1. Code Security

| | | |
|---|---|---|
| Reentrancy | DelegateCall | Integer Overflow |
| Input Validation | Unchecked this.call | Frozen Money |
| Arbitrary External Call | Unchecked Owner Transfer | Do-while Continue |
| Right-To-Left-Override Character | Unauthenticated Storage Access | Risk For Weak Randomness |
| TxOrigin | Missing Checks for Return Values | Diamond Inheritance |
| ThisBalance | VarType Deduction | Array Length Manipulation |
| Uninitialized Variable | Shadow Variable | Divide Before Multiply |
| Affected by Compiler Bug | | |

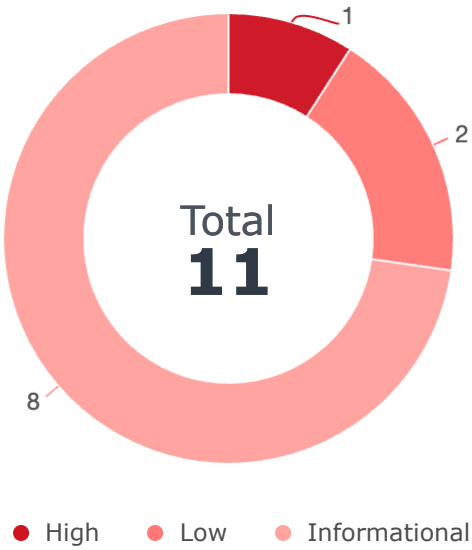
## 2.2. Optimization Suggestion

| | |
|---|---|
| Compiler Version | Improper State Variable Modification |
| Function Visibility | Deprecated Function |
| Externally Controlled Variables | Code Style |
| Constant Specific | Event Specific |
| Return Value Unspecified | Inexistent Error Message |
| State Variable Defined Without Storage Location | Import Issue |
| Compare With Timestamp/Block Number/Blockhash | Constructor in Base Contract Not Implemented |
| Delete Struct Containing the Mapping Type | Usage of '=+' |
| Paths in the Modifier Not End with "_" or Revert | Non-payable Public Functions Use msg.value |
| Lack of SafeMath | Compiler Error/Warning |
| Tautology Issue | Loop Depends on Array Length |
| Redundant/Duplicated/Dead Code | Code Complexity/Code Inefficiency |
| Undeclared Resource | Optimizable Return Statement |
| Unused Resource | |

## 2.3. Business Security

| |
|---|
| The Code Implementation is Consistent With Comments, Project White Papers and Other Materials |
| Permission Check |
| Address Check |

# 3. Findings




**Code Security**

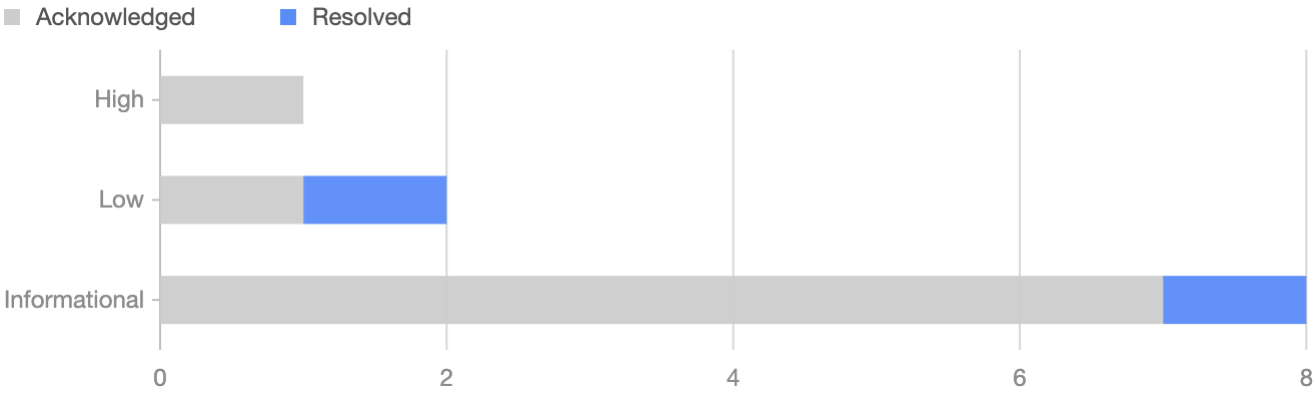
no vulnerabilities.

**Optimization Suggestion**

Informational: 8

**Business Security**

High: 1, Low: 2




| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| H-01 | Arbitrage through Sandwich Attacks on Adding Bonuses | Business Security | High | Acknowledged |
| L-02 | Be Careful to Support payToken Extended from ERC-20 Standard | Business Security | Low | Acknowledged |
| L-03 | No Upper Limit for fee | Business Security | Low | Resolved |
| I-04 | Incorrect Price Formula in Whitepaper | Optimization Suggestion | Informational | Resolved |
| I-05 | Higher Costs of Purchasing Badges with Expensive Tokens | Optimization Suggestion | Informational | Acknowledged |
| I-06 | Event Should be Emitted When Critical State Variables Change | Optimization Suggestion | Informational | Acknowledged |
| I-07 | No Check of Address Params with Zero Address | Optimization Suggestion | Informational | Acknowledged |
| I-08 | Recommend to Follow Code Layout Conventions | Optimization Suggestion | Informational | Acknowledged |

| ID | Title | Category | Severity | Status |
| --- | --- | --- | --- | --- |
| I-09 | Parameters Should Be Declared as Calldata | Optimization Suggestion | Informational | Acknowledged |
| I-10 | Function Visibility Can Be External | Optimization Suggestion | Informational | Acknowledged |
| I-11 | Floating Pragma | Optimization Suggestion | Informational | Acknowledged |

# H-01: Arbitrage through Sandwich Attacks on Adding Bonuses

High: Business Security

File Location: /contracts/InfluencerBadge.sol (commit e14d0d7):384

## Description

Attackers can arbitrage by performing sandwich attacks on adding bonuses. Specifically, attackers can observe the adding bonuses operation and quickly buy badges from a badge pool before the adding bonuses operation, and then sell badges immediately after the adding bonuses operation to arbitrage. Because the price of badges before the adding bonuses operation is lower than the price after the adding bonuses operation, attackers can profit from buying low and selling high.

POC

1. User A calls the `createBadgePool` function to create a badge pool, specifying USDT as the `payToken`, `constA` as 210, `constB` as 2100, and `revenueSharingPercent` as 30.
2. User B calls the `buyFromPool` function and buys 10 badges for about 1.2 USDT.
3. An attacker observes that a funder intends to call the `addBonus` function to add 100 USDT of bonus funds to the pool created in step 1. The attacker initiates the purchase of 10 badges in advance of this transaction, spending about 2.3 USDT.
4. The funder executes the adding bonuses operation, adding 100 USDT of bonus into the aforementioned created pool.
5. The attacker promptly sells the 10 badges purchased in step 3, obtaining about 63.7 USDT.
6. Ultimately, the attacker successfully arbitraged about 61 USDT. The attacker can apply this attack method to multiple pools for multiple arbitrage opportunities.

/contracts/InfluencerBadge.sol (commit e14d0d7)

```
384   function addBonus(
385       uint256 poolId,
386       uint256 bonusERC20Amount
387   ) external payable nonReentrant {
388       // only specific addresses can contribute funds (e.g. treasury
      contract)
389       require(isWhitelistFunder[msg.sender], "NO_PERMISSION");
390
391       BadgePoolConfig storage poolConfig = badgePoolConfigs[poolId];
392       require(poolConfig.kol != address(0), "INV_ID");
393       require(poolConfig.tokenBalance > 0, "CANNOT_ADD"); // used as
      denominator below
394
395       address payToken = poolConfig.payToken;
396       uint256 actualBonusAmount;
397
398       // pay token
399       if (payToken == address(0)) {
400           require(msg.value > 0, "INV_VAL1");
```

```
401             actualBonusAmount = msg.value;
402         } else {
403             require(bonusERC20Amount > 0, "INV_VAL2");
404             actualBonusAmount = bonusERC20Amount;
405             SafeERC20.safeTransferFrom(
406                 IERC20(payToken),
407                 msg.sender,
408                 address(this),
409                 actualBonusAmount
410             );
411         }
412 
413         // update parameters
414         uint256 newCoef = Math.mulDiv(
415             poolConfig.tokenBalance + actualBonusAmount,
416             10 ** 18,
417             poolConfig.tokenBalance, // can not be zero
418             Math.Rounding.Floor
419         );
420         poolConfig.varCoef1 = (poolConfig.varCoef1 * newCoef) / (10 ** 18);
421 
422         poolConfig.tokenBalance += actualBonusAmount;
423 
424         emit AddBonus(
425             msg.sender,
426             poolId,
427             actualBonusAmount,
428             poolConfig.tokenBalance
429         );
430     }
```

## Recommendation

It is advisable for the project team to be aware of the above risks and to clarify whether the situation meets expectations.

## Alleviation

The project team acknowledged the issue and decided to keep no change. To mitigate sandwich attacks, the project team has implemented the following restrictions in the contract:

1. They have limited the maximum number of badges that can be bought or sold in a single transaction (currently set to 10) and have restricted the `buyFromPool` and `sellToPool` functions to only be callable by EOA.

2. The project team will also impose constraints on the amount and timing of reward tokens injected via the `addBonus` function. At the beginning when the pool has relatively little supply, they will not inject any reward tokens. Furthermore, each injection of reward tokens will be limited to a certain percentage to prevent a significant price increase following the rewards deposit.

# L-02: Be Careful to Support payToken Extended from ERC-20 Standard

Low: Business Security

File Location: /contracts/InfluencerBadge.sol:673

## Description

The owner of the `InfluencerBadge` contract can call the `addWhitelistPayTokens` function to support more `payTokens`. From the contract implementation, it can be seen that the project supports native token and tokens that are compatible with the ERC-20 standard. Users need to transfer in and out `payToken` when calling the `buyFromPool` and `sellToPool` functions to trade badges in the badge pool.

It should be noted that some tokens that extend the ERC-20 standard are quite unique, as they require contracts that receive or send tokens to implement additional functions for successful token transfer. For instance, tokens compatible with the ERC-777 standard require contracts that send or receive tokens to additionally implement `tokensToSend` and `tokensReceived` functions. Similarly, tokens compatible with the ERC-223 standard necessitate that contracts receiving tokens additionally implement a `tokenReceived` function. Otherwise, the token transfer will fail.

/contracts/InfluencerBadge.sol

```
673   function addWhitelistPayTokens(
674       address[] calldata tokens
675   ) external onlyOwner {
676       uint256 addLength = tokens.length;
677       address token;
678       for (uint i = 0; i < addLength; i++) {
679           token = tokens[i];
680           isWhitelistPayToken[token] = true;
681       }
682   }
```

## Recommendation

It is recommended that the project team clarify whether they will support some extended ERC-20 standard tokens. If necessary, it is recommended to carefully review the requirements of the respective tokens to avoid situations where token transfers may fail.

## Alleviation

The project team acknowledged the issue and decided to keep no change. The project team has indicated that they will only select some of the top ERC20 tokens as `payToken`, ensuring that these tokens can be transferred without the need for additional implemented functions.

# L-03: No Upper Limit for fee

Low: Business Security

File Location: /contracts/InfluencerBadge.sol:589,590

## Description

In the `InfluencerBadge` contract, the owner can set the `protocolFeePercent` and `kolFeePercent` to a very high fee percent by calling function `setFeePercent`, thus causing a loss to the users. For example, when a user sell badges to a badge pool, if the `protocolFeePercent` and `kolFeePercent` is sufficiently high, then the calculated `allFee` will be very large, to the extent that tokens are almost entirely transferred to the `protocolFeeTo` and `kol` of the pool, rather than the user.

/contracts/InfluencerBadge.sol

```
585   function setFeePercent(
586       uint256 newProtocolPercent,
587       uint256 newKolFeePercent
588   ) external onlyOwner {
589       protocolFeePercent = newProtocolPercent;
590       kolFeePercent = newKolFeePercent;
591   }
```

## Recommendation

We suggest to constrain the setting of the `protocolFeePercent` and `kolFeePercent` to ensure that there is a check on its upper limit, or introduce multi sign or governance module to avoid a single point of key management failure.

## Alleviation

Resolved in commit e14d0d7. The project team restricts the fee percent by adding the condition `require(newKolFeePercent + newProtocolPercent <= 20, "EX_LIM")`.

# I-04: Incorrect Price Formula in Whitepaper

Informational: Optimization Suggestion

File Location:

## Description

In the whitepaper, there are two issues with the description of the formula:

1. Upon calculation, the price of badges in USDT should be equal to `((N+1)^2 + A)*K/B` instead of `(N^2 + A)*K/B`.
2. When there are no badges in the pool, i.e., N is 0, the price of badges in USDT is not 0, so the curve described in the whitepaper should be shifted upwards.

## Recommendation

It is recommended to revise the description of the formula in the whitepaper.

## Alleviation

The project team acknowledged the issue and will revise the description of the formula in the whitepaper.

# I-05: Higher Costs of Purchasing Badges with Expensive Tokens

Informational: Optimization Suggestion

File Location: /contracts/InfluencerBadge.sol:170

## Description

When creating a new badge pool, users can specify the payment currency `payToken` of the new badge pool. Then users can buy badges from the pool. It will cost around 0.1 `payToken` per badge. When the `payToken` has a very high price, buying badges will cost a lot of money. This cost raises the barrier for user engagement in trading badges and might reduce the enthusiasm of users to buy badges, subsequently decreasing the liquidity of badges.

/contracts/InfluencerBadge.sol

```
132  function createBadgePool(
133      address payToken,
134      uint256 constA,
135      uint256 constB,
136      uint256 revenueSharingPercent
137  )
138      external
139      callerIsUser
140      nonReentrant
141      whenNotPaused
142      returns (uint256 poolId)
143  {
144      // verify parameters
145      if (isCheckCreator) {
146          require(isWhitelistKOL[msg.sender], "NOT_IN_WL"); // in the
         whitelist
147      }
148      require(!isBlacklistKOL[msg.sender], "IN_BL"); // in the blacklist
149
150      require(!hasCreatedPool[msg.sender], "HAS_CRED"); // has created
151
152      if (isCheckConstA) {
153          require(isWhitelistConstA[constA], "INV_CONSTA");
154      }
155      require(constB > 0, "INV_CONSTB");
156      if (isCheckConstB) {
157          require(isWhitelistConstB[constB], "INV_CONSTB");
158      }
159      // verify hmp
160      if (isCheckHMPInCreation) {
161          // check level
162          require(getHMPLevel(msg.sender) >= hpmLevelThreshold,
         "INV_LEVEL");
163      }
164      require(isWhitelistPayToken[payToken], "NS_TOKEN"); // not supported
         token
165
```

```
166         // get amountPerPayToken which is "10^n"
167         uint256 amountPerPayToken;
168         if (payToken == address(0)) {
169             // native token
170             amountPerPayToken = 10 ** 18;
171         } else {
172             uint256 decimals = IERC20WithDecimals(payToken).decimals();
173             amountPerPayToken = 10 ** decimals;
174         }
175 
176         ......
177     }
```

## Recommendation

It is recommended that the project team allows for a more flexible pricing of badge trading to reduce the barrier for user participation.

## Alleviation

The project team acknowledged the issue and decided to keep no change. The project team indicated that it will address this issue by offering more price options to users through the invocation of the `addWhitelistConstA` and `addWhitelistConstB` functions at a later stage.

# I-06: Event Should be Emitted When Critical State Variables Change

Informational: Optimization Suggestion

File Location: /contracts/InfluencerBadge.sol:576,580,585,593,597,606,610,617,621,625,629,633,637,646,655,664,673,684,695,699,703,707,711,716,720,724

## Description

When some critical variables in the contract, such as `protocolFeePercent`, `kolFeePercent` and `protocolFeeTo` change, an event should be emitted so that the changes of these variables can be tracked off-chain.

/contracts/InfluencerBadge.sol

```
576   function setURI(string memory newBaseUri) external onlyOwner {
577       _setURI(newBaseUri);
578   }
579
580   function setHMPLevelThreshold(uint8 newLevel) external onlyOwner {
581       require(newLevel <= 6, "EX_LEVEL");
582       hpmLevelThreshold = newLevel;
583   }
584
585   function setFeePercent(
586       uint256 newProtocolPercent,
587       uint256 newKolFeePercent
588   ) external onlyOwner {
589       protocolFeePercent = newProtocolPercent;
590       kolFeePercent = newKolFeePercent;
591   }
592
593   function setProtocolFeeTo(address newReceiver) external onlyOwner {
594       protocolFeeTo = newReceiver;
595   }
596
597   function setRevenueSharingPercent(
598       uint256 poolId,
599       uint256 newPercent
600   ) external onlyOwner {
601       require(newPercent <= maxPercentInRevenueSharing, "INV_PCT");
602       BadgePoolConfig storage poolConfig = badgePoolConfigs[poolId];
603       poolConfig.revenueSharingPercent = newPercent;
604   }
605
606   function setMaxLimitInBuyOrSell(uint256 newLimit) external onlyOwner {
607       maxLimitInBuyOrSell = newLimit;
608   }
609
610   function setMaxPercentInRevenueSharing(
611       uint256 newPercent
612   ) external onlyOwner {
```

```
613         require(newPercent <= SCALE_DECIMAL, "INV_PCT");
614         maxPercentInRevenueSharing = newPercent;
615     }
616 
617     function toggleIsCheckHMPInCreation() external onlyOwner {
618         isCheckHMPInCreation = !isCheckHMPInCreation;
619     }
620 
621     function toggleIsCheckHMPInBuyOrSell() external onlyOwner {
622         isCheckHMPInBuyOrSell = !isCheckHMPInBuyOrSell;
623     }
624 
625     function toggleIsCheckCreator() external onlyOwner {
626         isCheckCreator = !isCheckCreator;
627     }
628 
629     function toggleIsCheckConstA() external onlyOwner {
630         isCheckConstA = !isCheckConstA;
631     }
632 
633     function toggleIsCheckConstB() external onlyOwner {
634         isCheckConstB = !isCheckConstB;
635     }
636 
637     function addWhitelistKOLs(address[] calldata users) external onlyOwner {
638         uint256 addLength = users.length;
639         address user;
640         for (uint i = 0; i < addLength; i++) {
641             user = users[i];
642             isWhitelistKOL[user] = true;
643         }
644     }
645 
646     function removeWhitelistKOLs(address[] calldata users) external
    onlyOwner {
647         uint256 removeLength = users.length;
648         address user;
649         for (uint i = 0; i < removeLength; i++) {
650             user = users[i];
651             isWhitelistKOL[user] = false;
652         }
653     }
654 
655     function addBlacklistKOLs(address[] calldata users) external onlyOwner {
656         uint256 addLength = users.length;
657         address user;
658         for (uint i = 0; i < addLength; i++) {
659             user = users[i];
660             isBlacklistKOL[user] = true;
661         }
662     }
663 
664     function removeBlacklistKOLs(address[] calldata users) external
    onlyOwner {
665         uint256 removeLength = users.length;
666         address user;
667         for (uint i = 0; i < removeLength; i++) {
668             user = users[i];
```

```
669             isBlacklistKOL[user] = false;
670         }
671     }
672
673     function addWhitelistPayTokens(
674         address[] calldata tokens
675     ) external onlyOwner {
676         uint256 addLength = tokens.length;
677         address token;
678         for (uint i = 0; i < addLength; i++) {
679             token = tokens[i];
680             isWhitelistPayToken[token] = true;
681         }
682     }
683
684     function removeWhitelistPayTokens(
685         address[] calldata tokens
686     ) external onlyOwner {
687         uint256 removeLength = tokens.length;
688         address token;
689         for (uint i = 0; i < removeLength; i++) {
690             token = tokens[i];
691             isWhitelistPayToken[token] = false;
692         }
693     }
694
695     function addWhitelistPreminer(address newMiner) external onlyOwner {
696         isWhitelistPreminter[newMiner] = true;
697     }
698
699     function removeWhitelistPreminer(address removedMiner) external
    onlyOwner {
700         isWhitelistPreminter[removedMiner] = false;
701     }
702
703     function addWhitelistConstA(uint256 constA) external onlyOwner {
704         isWhitelistConstA[constA] = true;
705     }
706
707     function removeWhitelistConstA(uint256 constA) external onlyOwner {
708         isWhitelistConstA[constA] = false;
709     }
710
711     function addWhitelistConstB(uint256 constB) external onlyOwner {
712         require(constB > 0, "NOT_ZERO");
713         isWhitelistConstB[constB] = true;
714     }
715
716     function removeWhitelistConstB(uint256 constB) external onlyOwner {
717         isWhitelistConstB[constB] = false;
718     }
719
720     function addWhitelistFunder(address funder) external onlyOwner {
721         isWhitelistFunder[funder] = true;
722     }
723
724     function removeWhitelistFunder(address funder) external onlyOwner {
725         isWhitelistFunder[funder] = false;
```

```
726   }
```

## Recommendation

It is recommended to emit an event when critical variables change.

## Alleviation

The project team acknowledged the issue and decided to keep no change. Because the project team thought that the functions involved in this issue are only callable by owner and will be invoked infrequently, so they are not considering adding event notifications for the time being.

# I-07: No Check of Address Params with Zero Address

Informational: Optimization Suggestion

File Location: /contracts/InfluencerBadge.sol:90,91,593

## Description

The input parameter of the address type in the function does not use the zero address for verification.

/contracts/InfluencerBadge.sol

```
85  constructor(
86      string memory name_,
87      string memory symbol_,
88      string memory uri_,
89      address usdt,
90      address protocolFeeTo_,
91      IHaloMembershipPass hmp_
92  ) ERC1155(uri_) Ownable(msg.sender) {
```

/contracts/InfluencerBadge.sol

```
593 function setProtocolFeeTo(address newReceiver) external onlyOwner {
594     protocolFeeTo = newReceiver;
595 }
```

## Recommendation

It is recommended to perform zero address verification on the input parameters of the address type.

## Alleviation

The project team acknowledged the issue and decided to keep no change. The project team indicated that they would meticulously check the parameters when deploying contracts and invoking the `setProtocolFeeTo` function, hence there will be no additional verification within the contract.

# I-08: Recommend to Follow Code Layout Conventions

Informational: Optimization Suggestion

File Location: /contracts/InfluencerBadge.sol:18

## Description

In the solidity document (https://docs.soliditylang.org/en/v0.8.17/style-guide.html), there are the following conventions for code layout:

Layout contract elements in the following order: 1. Pragma statements, 2. Import statements, 3. Interfaces, 4. Libraries, 5. Contracts.

Inside each contract, library or interface, use the following order: 1. Type declarations, 2. State variables, 3. Events, 4. Modifiers, 5. Functions.

Functions should be grouped according to their visibility and ordered: 1. constructor, 2. receive function (if exists), 3. fallback function (if exists), 4. external, 5. public, 6. internal, 7. private.

/contracts/InfluencerBadge.sol

```
16  /// @title Influencer Badge
17  /// @notice An ERC1155 contract, each tokenId(poolId) corresponds to a
    badge pool
18  contract InfluencerBadge is
19      ERC1155,
20      ERC1155Supply,
```

## Recommendation

It is recommended to follow the above code layout conventions to improve code readability.

## Alleviation

The project team acknowledged the issue and decided to keep no change.

# I-09: Parameters Should Be Declared as Calldata

Informational: Optimization Suggestion

File Location: /contracts/InfluencerBadge.sol:576

## Description

When the compiler parses the external or public function, it can directly read the function parameters from calldata. Setting it to other storage locations may waste gas. About 300-400 gas can be saved with optimization turned off while 120-150 gas can be saved vice versa.

/contracts/InfluencerBadge.sol

```
576     function setURI(string memory newBaseUri) external onlyOwner {
577         _setURI(newBaseUri);
578     }
```

## Recommendation

In external or public functions, the storage location of function parameters should be set to calldata to save gas.

## Alleviation

The project team acknowledged the issue and decided to keep no change.

# I-10: Function Visibility Can Be External

Informational: Optimization Suggestion

File Location: /contracts/InfluencerBadge.sol:542

## Description

Functions that are not called should be declared as external.

/contracts/InfluencerBadge.sol

```
540     }
541
542     function uri(uint256 tokenId) public view override returns (string
        memory) {
543         // require pool has created
544         require(tokenId > 0 && tokenId <= currentIndex, "INV_ID");
```

## Recommendation

Functions that are not called in the contract should be declared as external.

## Alleviation

The project team acknowledged the issue and decided to keep no change.

# I-11: Floating Pragma

Informational: Optimization Suggestion

File Location:
/contracts/InfluencerBadge.sol:2
/contracts/interfaces/EventsAndErrors.sol:2
/contracts/interfaces/IERC20WithDecimals.sol:2
/contracts/interfaces/IHaloMembershipPass.sol:2

## Description

Contracts should be deployed with fixed compiler version which has been tested thoroughly or make sure to lock the contract compiler version in the project configuration. Locked compiler version ensures that contracts will not be compiled by untested compiler version.

/contracts/InfluencerBadge.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.20;
3  import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
4  import "@openzeppelin/contracts/token/ERC1155/extensions/ERC1155Supply.
   sol";
```

/contracts/interfaces/EventsAndErrors.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.20;
3
4  interface EventsAndErrors {
```

/contracts/interfaces/IERC20WithDecimals.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.20;
3  import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
4
```

/contracts/interfaces/IHaloMembershipPass.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.20;
3
4  interface IHaloMembershipPass {
```

## Recommendation

Use a fixed compiler version, and consider whether the bugs in the selected compiler version (https://github.com/ethereum/solidity/releases) will affect the contract.

## Alleviation

The project team acknowledged the issue and decided to keep no change.

# 4. Disclaimer

No description, statement, recommendation or conclusion in this report shall be construed as endorsement, affirmation or confirmation of the project. The security assessment is limited to the scope of work as stipulated in the Statement of Work.

This report is prepared in response to source code, and based on the attacks and vulnerabilities in the source code that already existed or occurred before the date of this report, excluding any new attacks or vulnerabilities that exist or occur after the date of this report. The security assessment are solely based on the documents and materials provided by the customer, and the customer represents and warrants documents and materials are true, accurate and complete.

CONSULTANT DOES NOT MAKE AND HEREBY DISCLAIMS ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, REGARDING THE SERVICES, DELIVERABLES, OR ANY OTHER MATTER PERTAINING TO THIS REPORT.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR AND HEREBY DISCLAIMS MERCHANTABILITY, FITNESS FOR PURPOSE, TITLE, NON-INFRINGEMENT OR NON-APPROPRIATION OF INTELLECTUAL PROPERTY RIGHTS OF A THIRD PARTY, SATISFACTORY QUALITY, ACCURACY, QUALITY, COMPLETENESS, TIMELINESS, RESPONSIVENESS, OR PRODUCTIVITY OF THE SERVICES OR DELIVERABLES.

CONSULTANT EXCLUDES ANY WARRANTY THAT THE SERVICES AND DELIVERABLES WILL BE UNINTERRUPTED, ERROR FREE, FREE OF SECURITY DEFECTS OR HARMFUL COMPONENTS, REVEAL ALL SECURITY VULNERABILITIES, OR THAT ANY DATA WILL NOT BE LOST OR CORRUPTED.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR (A) ANY REPRESENTATIONS MADE BY ANY PERSON REGARDING THE SUFFICIENCY OR SUITABILITY OF SERVICES AND DELIVERABLES IN ANY ACTUAL APPLICATION, OR (B) WHETHER ANY SUCH USE WOULD VIOLATE OR INFRINGE THE APPLICABLE LAWS, OR (C) REVIEWING THE CUSTOMER MATERIALS FOR ACCURACY.

# 5. Appendix

## 5.1 Visibility

| Contract | FuncName | Visibility | Mutability | Modifiers |
|---|---|---|---|---|
| InfluencerBadge | _CTOR_ | public | Y | Ownable |
| InfluencerBadge | createBadgePool | external | Y | callerIsUser, nonReentrant, whenNotPaused |
| InfluencerBadge | buyFromPool | external | Y | callerIsUser, nonReentrant, whenNotPaused |
| InfluencerBadge | sellToPool | external | Y | callerIsUser, nonReentrant, whenNotPaused |
| InfluencerBadge | addBonus | external | Y | nonReentrant |
| InfluencerBadge | getBuyPrice | public | N | |
| InfluencerBadge | getSellPrice | public | N | |
| InfluencerBadge | uri | public | N | |
| InfluencerBadge | getHMPLevel | public | N | |
| InfluencerBadge | pause | external | Y | onlyOwner |
| InfluencerBadge | unpause | external | Y | onlyOwner |
| InfluencerBadge | setURI | external | Y | onlyOwner |
| InfluencerBadge | setHMPLevelThreshold | external | Y | onlyOwner |
| InfluencerBadge | setFeePercent | external | Y | onlyOwner |
| InfluencerBadge | setProtocolFeeTo | external | Y | onlyOwner |
| InfluencerBadge | setRevenueSharingPercent | external | Y | onlyOwner |
| InfluencerBadge | setMaxLimitInBuyOrSell | external | Y | onlyOwner |

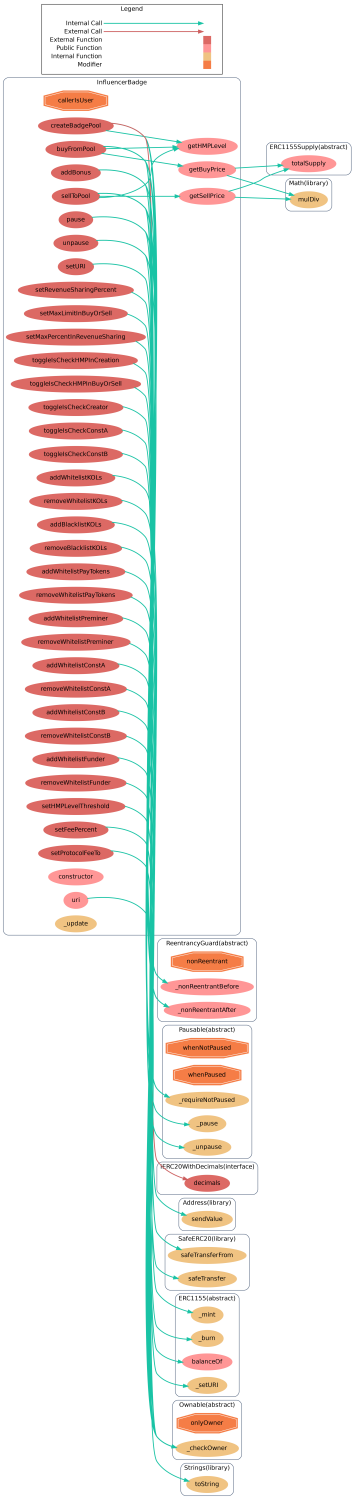
| Contract | FuncName | Visibility | Mutability | Modifiers |
|---|---|---|---|---|
| InfluencerBadge | setMaxPercentInRevenueSharing | external | Y | onlyOwner |
| InfluencerBadge | toggleIsCheckHMPInCreation | external | Y | onlyOwner |
| InfluencerBadge | toggleIsCheckHMPInBuyOrSell | external | Y | onlyOwner |
| InfluencerBadge | toggleIsCheckCreator | external | Y | onlyOwner |
| InfluencerBadge | toggleIsCheckConstA | external | Y | onlyOwner |
| InfluencerBadge | toggleIsCheckConstB | external | Y | onlyOwner |
| InfluencerBadge | addWhitelistKOLs | external | Y | onlyOwner |
| InfluencerBadge | removeWhitelistKOLs | external | Y | onlyOwner |
| InfluencerBadge | addBlacklistKOLs | external | Y | onlyOwner |
| InfluencerBadge | removeBlacklistKOLs | external | Y | onlyOwner |
| InfluencerBadge | addWhitelistPayTokens | external | Y | onlyOwner |
| InfluencerBadge | removeWhitelistPayTokens | external | Y | onlyOwner |
| InfluencerBadge | addWhitelistPreminer | external | Y | onlyOwner |
| InfluencerBadge | removeWhitelistPreminer | external | Y | onlyOwner |
| InfluencerBadge | addWhitelistConstA | external | Y | onlyOwner |
| InfluencerBadge | removeWhitelistConstA | external | Y | onlyOwner |
| InfluencerBadge | addWhitelistConstB | external | Y | onlyOwner |

| Contract | FuncName | Visibility | Mutability | Modifiers |
| --- | --- | --- | --- | --- |
| InfluencerBadge | removeWhitelistConstB | external | Y | onlyOwner |
| InfluencerBadge | addWhitelistFunder | external | Y | onlyOwner |
| InfluencerBadge | removeWhitelistFunder | external | Y | onlyOwner |
| InfluencerBadge | _update | internal | N | |

# 5. Appendix

## 5.2 Call Graph

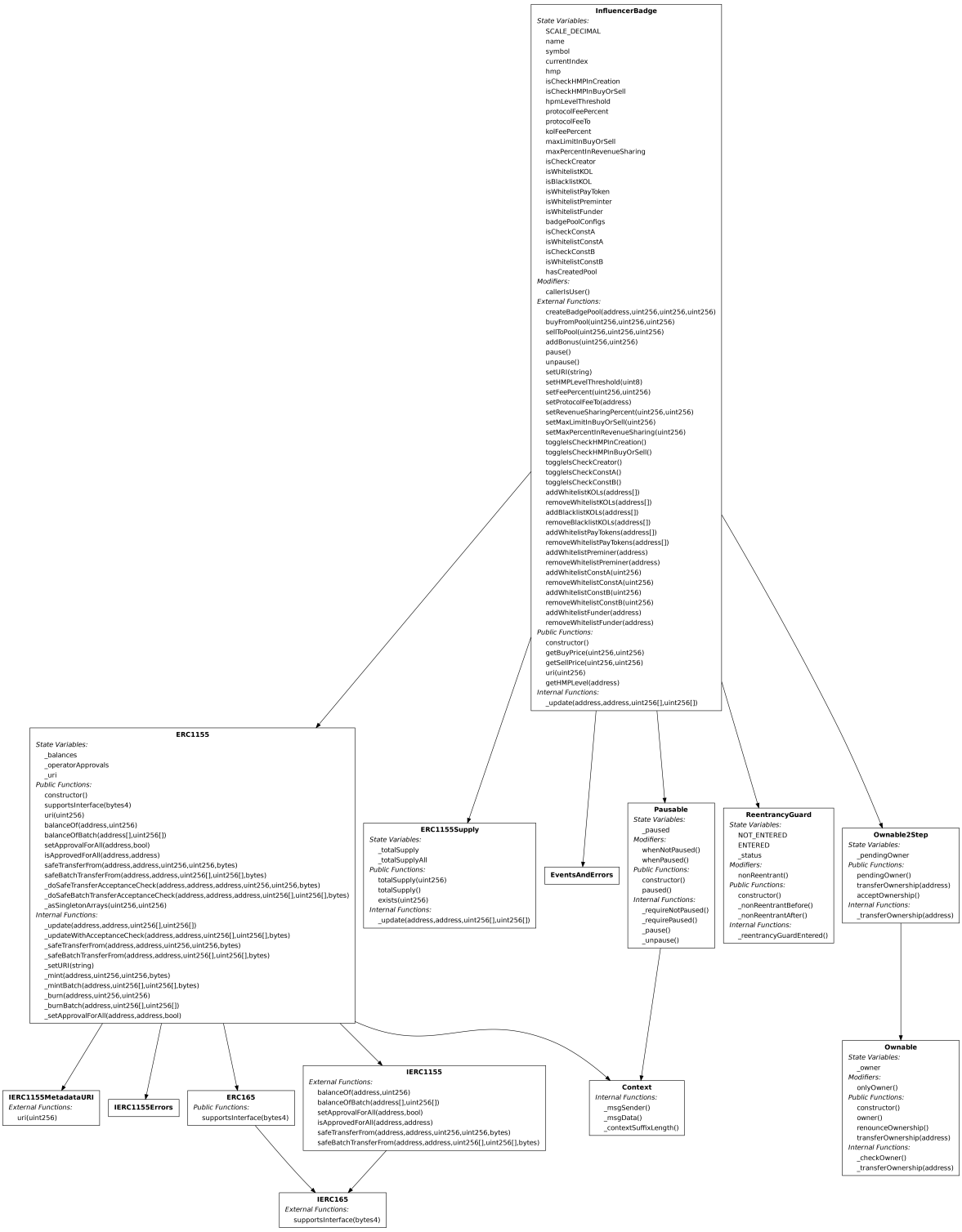
### InfluencerBadge



# 5. Appendix

## 5.3 Inheritance Graph

### InfluencerBadge



## 5.4 Formal Verification Metadata

**1. If the creator is checked in pool creation and the creator is whitelisted, the pool configuration should reflect that the premint phase has not finished.**

```
/// #if_succeeds {:msg "If the creator is checked in pool creation and the
creator is whitelisted, the pool configuration should reflect that the
premint phase has not finished."} always(isCheckCreator &&
isWhitelistKOL[msg.sender] ==> !badgePoolConfigs[poolId].hasFinishPremint);
function createBadgePool(
    address payToken,
    uint256 constA,
    uint256 constB,
    uint256 revenueSharingPercent
)
    external
    callerIsUser
    nonReentrant
    whenNotPaused
    returns (uint256 poolId)
{
```

Passed.

**2. If the creator is not whitelisted or creator checking is disabled, the pool configuration should reflect that the premint phase has finished.**

```
/// #if_succeeds {:msg "If the creator is not whitelisted or creator
checking is disabled, the pool configuration should reflect that the
premint phase has finished."} always((!isCheckCreator ||
!isWhitelistKOL[msg.sender]) ==> badgePoolConfigs[poolId].hasFinishPremint);
function createBadgePool(
    address payToken,
    uint256 constA,
    uint256 constB,
    uint256 revenueSharingPercent
)
    external
    callerIsUser
    nonReentrant
    whenNotPaused
    returns (uint256 poolId)
{
```

Passed.

**3. The total token balance for the pool should correctly increase by the amount spent on buying the tokens minus administrative fees.**

```
/// #if_succeeds {:msg "The total token balance for the pool should
correctly increase by the amount spent on buying the tokens minus
```

```
administrative fees."} let _, protocolFee, kolFee :=
previously(getBuyPrice(poolId, buyAmount)) in
always((previously(badgePoolConfigs[poolId].tokenBalance) + (payInAddFee -
protocolFee - kolFee)) == badgePoolConfigs[poolId].tokenBalance);
function buyFromPool(
    uint256 poolId,
    uint256 buyAmount,
    uint256 maxPayIn
)
    external
    payable
    callerIsUser
    nonReentrant
    whenNotPaused
    returns (uint256 payInAddFee)
{
```

Passed.

**4. The payInAddFee exactly equals the sum of buyPrice, protocolFee, and kolFee.**

```
/// #if_succeeds {:msg "The payInAddFee exactly equals the sum of buyPrice,
protocolFee, and kolFee."} let buyPrice, protocolFee, kolFee :=
previously(getBuyPrice(poolId, buyAmount)) in always(payInAddFee ==
buyPrice + protocolFee + kolFee);
function buyFromPool(
    uint256 poolId,
    uint256 buyAmount,
    uint256 maxPayIn
)
    external
    payable
    callerIsUser
    nonReentrant
    whenNotPaused
    returns (uint256 payInAddFee)
{
```

Passed.

**5. Upon successful token transfer in the `sellToPool` function, the amount actually received by the seller `payOutSubFee` must be greater than or equal to `minPayOut` specified.**

```
/// #if_succeeds {:msg "Upon successful token transfer in the `sellToPool`
function, the amount actually received by the seller `payOutSubFee` must be
greater than or equal to `minPayOut` specified."} always(payOutSubFee >=
minPayOut);
function sellToPool(
    uint256 poolId,
    uint256 sellAmount,
```

```
    uint256 minPayOut
)
    external
    callerIsUser
    nonReentrant
    whenNotPaused
    returns (uint256 payOutSubFee)
{
```

Passed.

**6. The function `sellToPool` should result in the correct deduction of the seller's badge amount in the token balance of `poolId`.**

```
/// #if_succeeds {:msg "The function `sellToPool` should result in the
correct deduction of the seller's badge amount in the token balance of
`poolId`."} let _, protocolFee, kolFee := previously(getSellPrice(poolId,
sellAmount)) in always(previously(badgePoolConfigs[poolId].tokenBalance) ==
badgePoolConfigs[poolId].tokenBalance + (payOutSubFee + protocolFee +
kolFee));
function sellToPool(
    uint256 poolId,
    uint256 sellAmount,
    uint256 minPayOut
)
    external
    callerIsUser
    nonReentrant
    whenNotPaused
    returns (uint256 payOutSubFee)
{
```

Passed.

**7. The `sellToPool` function must correctly burn the `sellAmount` of badges from the seller's balance for the `poolId`.**

```
/// #if_succeeds {:msg "The `sellToPool` function must correctly burn the
`sellAmount` of badges from the seller's balance for the `poolId`."}
always(previously(balanceOf(msg.sender, poolId)) == balanceOf(msg.sender,
poolId) + sellAmount);
function sellToPool(
    uint256 poolId,
    uint256 sellAmount,
    uint256 minPayOut
)
    external
    callerIsUser
    nonReentrant
    whenNotPaused
```

```
    returns (uint256 payOutSubFee)
{
```

Passed.

**8. The updated balance of tokens in the badge pool should be the sum of the current balance and the actual amount added as a bonus.**

```
/// #if_succeeds {:msg "The updated balance of tokens in the badge pool
should be the sum of the current balance and the actual amount added as a
bonus."} always(previously(badgePoolConfigs[poolId].tokenBalance) +
actualBonusAmount == badgePoolConfigs[poolId].tokenBalance);
function addBonus(
    uint256 poolId,
    uint256 bonusERC20Amount
) external payable nonReentrant {
```

Passed.

**9. After a bonus has been added to a particular badge pool, both the buy and the sell price should experience an increase.**

```
/// #if_succeeds {:msg "After a bonus has been added to a particular badge
pool, both the buy and the sell price should experience an increase."}
always((previously(getBuyPrice(poolId, 1)) < getBuyPrice(poolId, 1)) &&
(previously(getSellPrice(poolId, 1)) < getSellPrice(poolId, 1)));
function addBonus(
    uint256 poolId,
    uint256 bonusERC20Amount
) external payable nonReentrant {
```

Passed.