Highlight, Inc.

Smart Contract Security Assessment

December 10, 2021

Prepared for:

Kevin Matthews

Highlight, Inc.

Prepared by:

Stephen Tong and Jasraj Bedi

Zellic Inc.

Contents

1. Introduction

1.1. About Zellic

Zellic was founded in 2019 by a team of blockchain specialists with more than a decade of combined industry experience. They are leading experts in smart contract development, Web3 technologies, cryptography, web security, and reverse engineering. Before Zellic, they founded perfect blue, the top competitive hacking team in the world. Since then, they have won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than to simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible.

To keep up to date with our latest news and announcements, check out our website https://zellic.io or follow @zellic_io on Twitter. To engage with us directly, email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1.2. About Highlight

Highlight is a platform for content creators to build custom-branded, members-only clubs that engage, connect, and reward their most loyal supporters. Using low-carbon NFTs, creators can fully own their audience, and monetize transactions, forever.

1.3. Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but most of the time is be spent on a thorough manual review of the entire scope.

Alongside a variety of open-source tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review.
We analyze the scoped smart contract code using automated tools to quickly sieve out and catch these "shallow" bugs. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, etc. as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We manually review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents. We also thoroughly examine the specifications and designs themselves for inconsistencies, flaws, and vulnerabilities. This involves use-cases that open the opportunity for abuse, such as flawed tokenomics or share pricing, arbitrage opportunities, etc.

Complex integration risks. Several high-profile exploits have been the result of not any bug within the contract itself, but rather an unintended consequence of its interaction with the broader DeFi ecosystem. We perform a meticulous review of all of the contract's possible external interactions, and summarize the associated risks; for example: flash loan attacks, oracle price manipulation, MEV/sandwich attacks, etc.

Code maturity. We review for possible improvements in the codebase in general. We look for violations of industry best practices and guidelines, or code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, etc.

For each finding, Zellic assigns it an *impact* rating based on its *severity* and *likelihood*. There is no hard-and-fast formula for calculating a finding's impact; we assign it on a case-by-case basis based on our professional judgement and experience. As one would expect, both the severity and likelihood of an issue affect its impact; for instance, a highly severe issue's impact may be attenuated by a very low likelihood. We assign the following impact ratings (ordered by importance): *Critical*, *High*, *Medium*, *Low*, and *Informational*.

Similarly, Zellic organizes its reports such that the most *important* findings come first in the document, rather than impact alone. Thus, we may sometimes emphasize a "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same *impact rating*, their *importance* may differ. This varies based on numerous soft factors, such as our clients' threat model, their business needs, project timelines, etc. We aim to provide useful and actionable advice to our clients that consider their long-term goals, rather than simply a list of security issues at present.

1.4. Scope

The engagement involved a review of the following targets:

highlight-contracts

Repository	https://github.com/highlightxyz/contracts
Versions	`4106ae8301d9ad9c65ff0f890b7daf4f1ad3a578`
Type	Solidity
Platform	Ethereum (Polygon chain)

1.5. Disclaimer

This assessment does not provide any warranties on finding all possible issues within its scope; i.e., the evaluation results do not guarantee the absence of any subsequent issues. We, of course, also cannot make guarantees on any additional code added to the assessed project after our assessment has concluded. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program. Finally, this assessment report should not be considered as financial nor investment advice.

2. Executive Summary

We audited the scoped contracts and we discovered 14 findings. **Fortunately, no critical/high impact issues were found.** We applaud Highlight for their attention to detail and diligence in maintaining high code quality standards. Of the 14 findings, 2 were of medium impact, and 4 were of low impact. The remaining findings were informational in nature.

For the purpose of this audit, Highlight's smart contracts are mostly centered around the ERC1155 standard. Thus, there is relatively low chance of complex integration risks (e.g., oracle attacks, flash loans attacks, etc.). Nevertheless, business logic errors and other general coding mistakes still pose a threat.

Highlight's platform is designed around a lay audience; their vision is to make NFTs accessible to the general public, like average content creators and fans. Content creators, naturally, need to exercise control over their communities, so Highlight's contracts are more centralized than a typical DeFi platform may be. Notwithstanding that, Highlight's tokens are still much less centralized than those of a conventional fan club. This invites the question of whether there are centralization risks, and if so, how they should be mitigated. We discuss this in Section 4.

In its most common use case, the Highlight platform subsidizes its users' transaction fees by making them on their behalf. This enables laypeople to access NFTs easily with little technical knowledge. However, this poses a notable business risk to Highlight in terms of gas costs. Thus, we treated gas optimizations more seriously in this assessment than in some other projects.

3. Detailed Findings

3.1. Tokens may be minted, burned, or transferred even when the contract is paused

- **Target:** BasicCommunity
- **Severity:** Medium
- **Impact:** Medium
- **Category:** Business Logic
- **Likelihood:** Low

Description

The functions `mintTokenManagerNew`, `mintTokenManagerExisting`, `safeTransferFromTokenManager`, and `burnTokenManager` do not check whether the contract is currently paused (`whenNotPaused`).

```
/**
 * @dev See {ICommunity-mintTokenManagerNew}.
 */
function mintTokenManagerNew(
    address[] calldata to,
    uint256[] calldata amounts,
    string[] calldata uris,
    bool[] calldata isMembership
) external virtual override nonReentrant tokenManagerRequired(msg.sender) returns (uint256[] memory tokenIds) {
    return _mintNew(msg.sender, to, amounts, uris, isMembership);
}

/**
 * @dev See {ICommunity-mintTokenManagerExisting}.
 */
function mintTokenManagerExisting(
    address[] calldata to,
    uint256[] calldata tokenIds,
    uint256[] calldata amounts
) external virtual override nonReentrant tokenManagerRequired(msg.sender) {
    for (uint256 i = 0; i < tokenIds.length; i++) {
        require(_tokensManager[tokenIds[i]] == msg.sender, "Unauthorized tokenManager");
    }
    _mintExisting(to, tokenIds, amounts);
}
```

Impact

The pausing feature was designed to mitigate off-chain security incidents such as a compromised key, etc. Pausing, therefore, is an important way to guarantee the integrity of the on-chain state. A compromised token manager could impact the tokenomics or even steal users' tokens, without any immediate remediations available.

Recommendations

1. Add the `whenNotPaused` modifier to the affected functions.

Remediation

The issue has been acknowledged by Highlight, and a fix is pending.

3.2. Several functions lack a single, clear responsibility

- **Target:** BasicCommunity
- **Severity:** Medium
- **Impact:** Medium
- **Category:** Code Maturity/Gas Optimization
- **Likelihood:** n/a

Description

Several functions in the contract are heavily overloaded and perform several different tasks. The functions `_mintNew` and `_mintExisting` especially suffer from this.

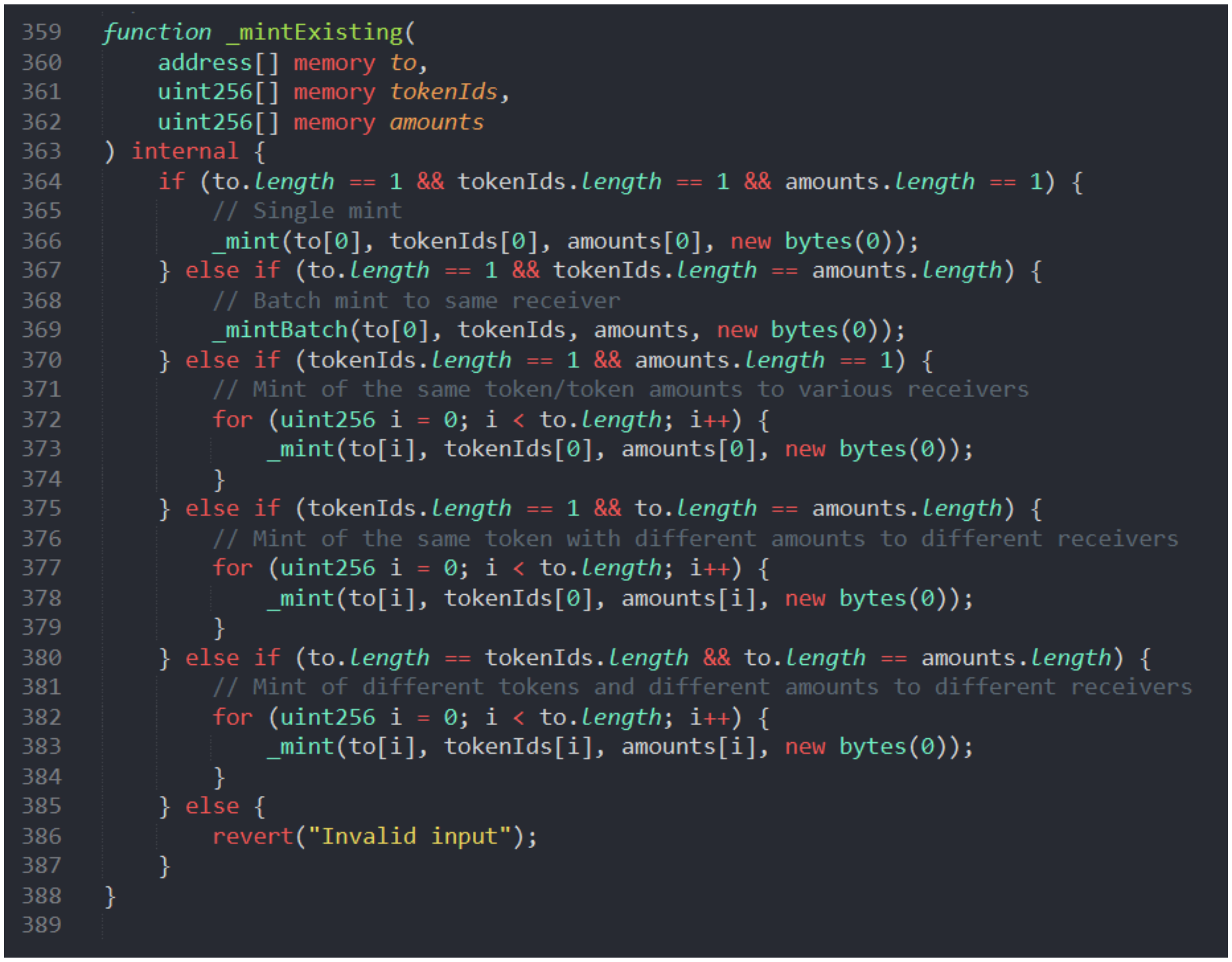
```
function _mintNew(
    address _tokenManager,
    address[] memory to,
    uint256[] memory amounts,
    string[] memory uris,
    bool[] memory isMembership
) internal returns (uint256[] memory tokenIds) {
    require(to.length > 0 && amounts.length > 0 && isMembership.length > 0, "Array cannot be empty");
    if (to.length > 1) {
        // Multiple receiver.  Give every receiver the same new token
        tokenIds = new uint256[](1);
        require(uris.length <= 1 && (amounts.length == 1 || to.length == amounts.length), "Invalid input");
    } else {
        // Single receiver.  Generating multiple tokens
        tokenIds = new uint256[](amounts.length);
        require(uris.length == 0 || amounts.length == uris.length, "Invalid input");
    }

    ...

    if (to.length == 1 && tokenIds.length == 1) {
        // Single mint
        _mint(to[0], tokenIds[0], amounts[0], new bytes(0));
    } else if (to.length > 1) {
        // Multiple receivers.  Receiving the same token
        if (amounts.length == 1) {
            // Everyone receiving the same amount
            for (uint256 i = 0; i < to.length; i++) {
                _mint(to[i], tokenIds[0], amounts[0], new bytes(0));
            }
        } else {
            // Everyone receiving different amounts
            for (uint256 i = 0; i < to.length; i++) {
                _mint(to[i], tokenIds[0], amounts[i], new bytes(0));
            }
        }
    } else {
        _mintBatch(to[0], tokenIds, amounts, new bytes(0));
    }

    return tokenIds;
}
```

The function `_mintNew` really is responsible for performing 4 different tasks: (1) single mint, (2) multi-receiver mint, same amount, (3) multi-receiver mint, differing amounts, and (4) single-receiver, multi-token mint. Not only is this code complex, it is inefficient due to the required overhead of unnecessarily constructing and unwrapping arrays, as well as the branch comparison logic.



```
function _mintExisting(
    address[] memory to,
    uint256[] memory tokenIds,
    uint256[] memory amounts
) internal {
    if (to.length == 1 && tokenIds.length == 1 && amounts.length == 1) {
        // Single mint
        _mint(to[0], tokenIds[0], amounts[0], new bytes(0));
    } else if (to.length == 1 && tokenIds.length == amounts.length) {
        // Batch mint to same receiver
        _mintBatch(to[0], tokenIds, amounts, new bytes(0));
    } else if (tokenIds.length == 1 && amounts.length == 1) {
        // Mint of the same token/token amounts to various receivers
        for (uint256 i = 0; i < to.length; i++) {
            _mint(to[i], tokenIds[0], amounts[0], new bytes(0));
        }
    } else if (tokenIds.length == 1 && to.length == amounts.length) {
        // Mint of the same token with different amounts to different receivers
        for (uint256 i = 0; i < to.length; i++) {
            _mint(to[i], tokenIds[0], amounts[i], new bytes(0));
        }
    } else if (to.length == tokenIds.length && to.length == amounts.length) {
        // Mint of different tokens and different amounts to different receivers
        for (uint256 i = 0; i < to.length; i++) {
            _mint(to[i], tokenIds[i], amounts[i], new bytes(0));
        }
    } else {
        revert("Invalid input");
    }
}
```

The function `_mintExisting` suffers from the same problems mentioned above.

These functions are mostly inherited from Manifold codebase that inspired Highlight's code architecture.

Impact

Code complexity is overwhelmingly the primary source of bugs. It is incredibly difficult to accurately reason about, modify, or test complex functions. Thus, it is crucial to limit the complexity and scope of individual functions to a minimum.

A well-written function should perform exactly one task, and one task alone. It should have obvious pre-conditions and post-conditions that are specific to that function only. The aforementioned functions do not exemplify this best practice: they handle four different cases, each with their own conditions, all in the same function.

Functions that have many responsibilities are also difficult to document accurately and thoroughly. This is a major path to future bugs, as the code will be difficult to understand for future developers. Bugs can easily slip under the radar and evade detection during code review.

Complex functions are also gas-guzzlers. In practice, the most common use-case for these functions would likely be the single-token-single-receiver, or single-token-multi-receiver cases. These cases do not require the additional logic for handling multiple amounts, multiple tokens, etc. which become pure overhead under the current implementation: the calling function must pass scalar values as arrays, or even pass arrays that are completely unused.

Recommendations

1. Split the function `_mintNew` into 4 separate functions, that handles each of the individual cases separately. This will greatly simplify the code and save gas.
2. Split the function `_mintExisting` into 4 separate functions, that handles each of the individual cases separately. This will greatly simplify the code and save gas.
2. Modify the callers of the respective functions to use the new, specialized functions.

Remediation

The affected functions have been refactored.

3.3. A retired token manager can compromise community integrity

- **Target:** BasicCommunity
- **Severity:** Low
- **Impact:** Low
- **Category:** Business Logic
- **Likelihood:** Low

Description

The function `unregisterTokenManager` does not un-approve the token manager for existing users' tokens, which can be leveraged to call `safeTransferFrom`.

```
/**
 * @dev See {ICommunity-unregisterTokenManager}.
 */
function unregisterTokenManager(address _tokenManager) external override onlyRole(PLATFORM_ROLE) {
    _unregisterTokenManager(_tokenManager);
}

```

Impact

A retired token manager that is fully un-registered still has control over users' tokens.

Recommendations

1. Create a hash table or enumerable set of retired token managers to explicitly block unregistered token managers from calling `safeTransferFrom` and `safeBatchTransferFrom`. Remember to correctly update this block list if the retired token manager is ever re-registered by removing it from the block list; or:
3. Do not approve the current token manager when minting new tokens. Remove calls to `_approveTokenManager` in _mint and `_mintBatch`. The current token manager can already transfer users' token regardless, through the function `safeTransferFromTokenManager`.

Remediation

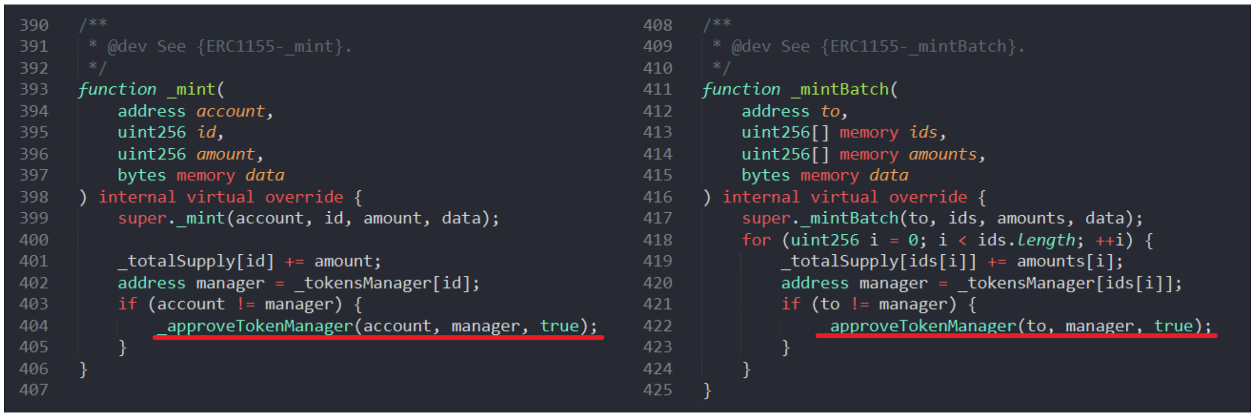
The issue has been acknowledged by Highlight, and a fix is pending.

3.4. Token managers seem to be approved unnecessarily

- **Target:** BasicCommunity
- **Severity:** Low
- **Impact:** Low
- **Category:** Code Maturity/Gas Optimization
- **Likelihood:** n/a

Description

The functions `_mint` and `_mintBatch` call `_approveTokenManager` to approve the current token manager for all newly minted tokens, although this is not actually necessary for the token manager to have control over the token supply. The current token manager can already transfer users' tokens regardless, through `safeTransferFromTokenManager`.



```
/**
 * @dev See {ERC1155-_mint}.
 */
function _mint(
    address account,
    uint256 id,
    uint256 amount,
    bytes memory data
) internal virtual override {
    super._mint(account, id, amount, data);

    _totalSupply[id] += amount;
    address manager = _tokensManager[id];
    if (account != manager) {
        _approveTokenManager(account, manager, true);
    }
}

```

```
/**
 * @dev See {ERC1155-_mintBatch}.
 */
function _mintBatch(
    address to,
    uint256[] memory ids,
    uint256[] memory amounts,
    bytes memory data
) internal virtual override {
    super._mintBatch(to, ids, amounts, data);
    for (uint256 i = 0; i < ids.length; ++i) {
        _totalSupply[ids[i]] += amounts[i];
        address manager = _tokensManager[ids[i]];
        if (to != manager) {
            _approveTokenManager(to, manager, true);
        }
    }
}
```

Impact

Unnecessary gas usage; approval uses storage, which is costly.

Recommendations

Do not approve the current token manager when minting new tokens. Remove calls to `_approveTokenManager` in _mint and `_mintBatch`. The current token manager can already transfer users' token regardless, through the function `safeTransferFromTokenManager`. In token manager code, replace `safeTransferFrom` with `safeTransferFromTokenManager`.

Remediation

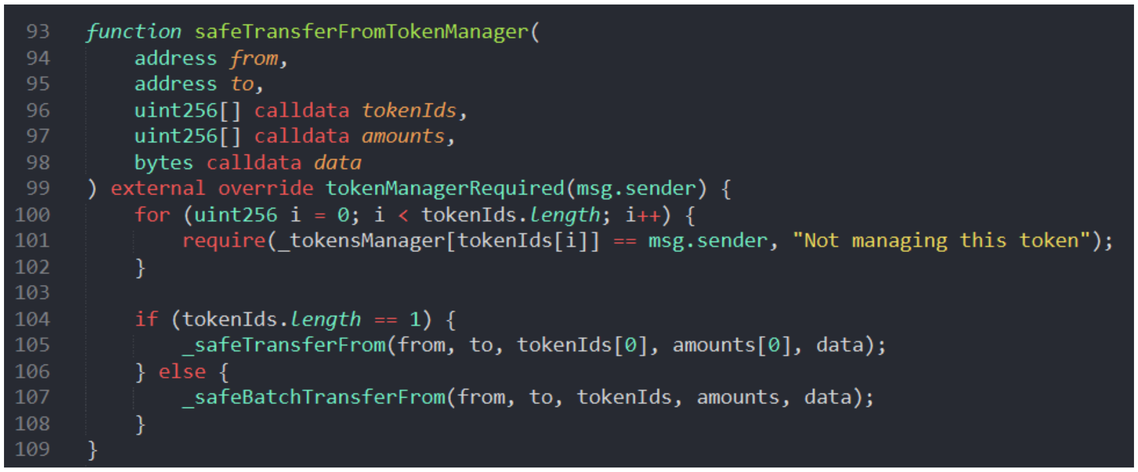
The contract has been modified so that token managers are not approved during minting.

3.5. Tokens may be transferred to an address the token manager is not approved for

- **Target:** BasicCommunity
- **Severity:** Low
- **Impact:** Low
- **Category:** Business Logic
- **Likelihood:** Medium

Description

The current token manager is only approved upon the minting of a new token. If a user has on-chain control over their tokens, they may transfer it to a new address that does not have the token manager approved.



```
function safeTransferFromTokenManager(
    address from,
    address to,
    uint256[] calldata tokenIds,
    uint256[] calldata amounts,
    bytes calldata data
) external override tokenManagerRequired(msg.sender) {
    for (uint256 i = 0; i < tokenIds.length; i++) {
        require(_tokensManager[tokenIds[i]] == msg.sender, "Not managing this token");
    }

    if (tokenIds.length == 1) {
        _safeTransferFrom(from, to, tokenIds[0], amounts[0], data);
    } else {
        _safeBatchTransferFrom(from, to, tokenIds, amounts, data);
    }
}
```

Impact

The current token manager will not be able to use `safeTransferFrom` to transfer tokens after a user transfers them to a different address. Although users do not currently have direct on-chain custody over their tokens in the current implementation of the Highlight platform, this is a planned feature for future iterations.

Recommendations

1. Add a call to `_approveTokenManager` when transferring tokens in `safeBatchTransferFrom` or `safeTransferFrom`.
2. Use `safeTransferFromTokenManager` instead of `safeTransferFrom` in token manager code.

Remediation

The contract has been modified so that token managers are not approved during minting.

3.6. Unnecessary use of storage

- **Target:** Community (and subclasses)
- **Severity:** Low
- **Impact:** Informational
- **Category:** Gas Optimization
- **Likelihood:** n/a

Description

The second element of `_tokenTypeCount`, the `membershipTokenLimit`, can be made into a compile-time constant, as it is never updated in Community or any of its subclasses.

```
// membershipTokenCount, membershipTokenLimit, benefitTokenCount
uint256[] internal _tokenTypeCount = [0, 100, 0];
```

Impact

This leads to a waste of gas in any function referencing `_tokenTypeCount`, such as `_unregisterTokenManager` or `BasicCommunity._mintNew`.

Recommendations

Eliminate the second element of `_tokenTypeCount` and replace it with a compile-time constant of a maximum of 100 member tokens.

Remediation

The code was refactored based on our recommendations.

3.7. Confusing use of array as a tuple of independent variables

- **Target:** Community (and subclasses)
- **Severity:** Medium
- **Impact:** Informational
- **Category:** Code Maturity/Gas Optimization
- **Likelihood:** n/a

Description

The array `_tokenTypeCount` is used as a 3-tuple of the independent variables `membershipTokenCount`, `membershipTokenLimit`, and `benefitTokenCount`. This appears to be the result of aggressive gas optimization.

Impact

Although reads from storage are indeed expensive, the complexity it introduces into the codebase is unjustified. This is a likely source of potential bugs in the future as it obfuscates code that uses `_tokenTypeCount`, making it hard to read or reason about.

Consider this code in the subclass BasicCommunity:

```
// reads from storage are expensive
uint256[2] memory tokenCountIncrement;
uint256[] memory tokenTypeCountTemporary = _tokenTypeCount;
uint256 _tokenId;
for (uint256 i = 0; i < tokenIds.length; i++) {
    if (isMembership[i]) {
        tokenCountIncrement[0]++;
        _tokenId = tokenTypeCountTemporary[0] + tokenCountIncrement[0];
        tokenIds[i] = _tokenId;
        _tokensManager[_tokenId] = _tokenManager;
    } else {
        tokenCountIncrement[1]++;
        _tokenId = tokenTypeCountTemporary[1] + tokenTypeCountTemporary[2] + tokenCountIncrement[1];
        tokenIds[i] = _tokenId;
        _tokensManager[_tokenId] = _tokenManager;
    }

    if (i < uris.length && bytes(uris[i]).length > 0) {
        _tokenURI[tokenIds[i]] = uris[i];
    }
}
uint256[2] memory tokenCountRes = [
    tokenTypeCountTemporary[0] + tokenCountIncrement[0],
    tokenTypeCountTemporary[2] + tokenCountIncrement[1]
];

_tokenTypeCount[0] = tokenCountRes[0];
_tokenTypeCount[2] = tokenCountRes[1];
```

The magic array indices 0, 1, and 2 are confusing and hard to keep track of. This can easily confuse developers, facilitating the generation of bugs. Considering the critical functionality that the three variables (`membershipTokenCount`, `membershipTokenLimit`, and `benefitTokenCount`) play in the codebase—allocating and managing the ERC1155 token ID space—these bugs would likely be very serious.

Recommendations

1. Replace the array `_tokenTypeCount` with three separate variables; or alternatively, use a struct. The following code excerpt shows a possible refactoring of the previous code from `_mintNew`:

```
uint256 newMembershipTokenId = membershipTokenCount;
uint256 newBenefitTokenId = MEMBERSHIP_TOKEN_LIMIT + benefitTokenCount;
for (uint256 i = 0; i < tokenIds.length; i++) {
    if (isMembership[i]) {
        newMembershipTokenId++;
        tokenIds[i] = newMembershipTokenId;
        _tokensManager[newMembershipTokenId] = _tokenManager;
    } else {
        newBenefitTokenId++;
        tokenIds[i] = newBenefitTokenId;
        _tokensManager[newBenefitTokenId] = _tokenManager;
    }

    if (i < uris.length && bytes(uris[i]).length > 0) {
        _tokenURI[tokenIds[i]] = uris[i];
    }
}

require(newMembershipTokenId <= MEMBERSHIP_TOKEN_LIMIT, "Too many membership tokens");
require(newMembershipTokenId >= membershipTokenCount, "Overflow");
require(newBenefitTokenId >= MEMBERSHIP_TOKEN_LIMIT + benefitTokenCount, "Overflow");
membershipTokenCount = newMembershipTokenId;
benefitTokenCount = newBenefitTokenId - MEMBERSHIP_TOKEN_LIMIT;
```

2. The gas consumption benefit of combining the three variables into a single array is unclear. To avoid any premature gas optimizations, we recommend that profiling be conducted to understand which code segments are the most expensive.

Remediation

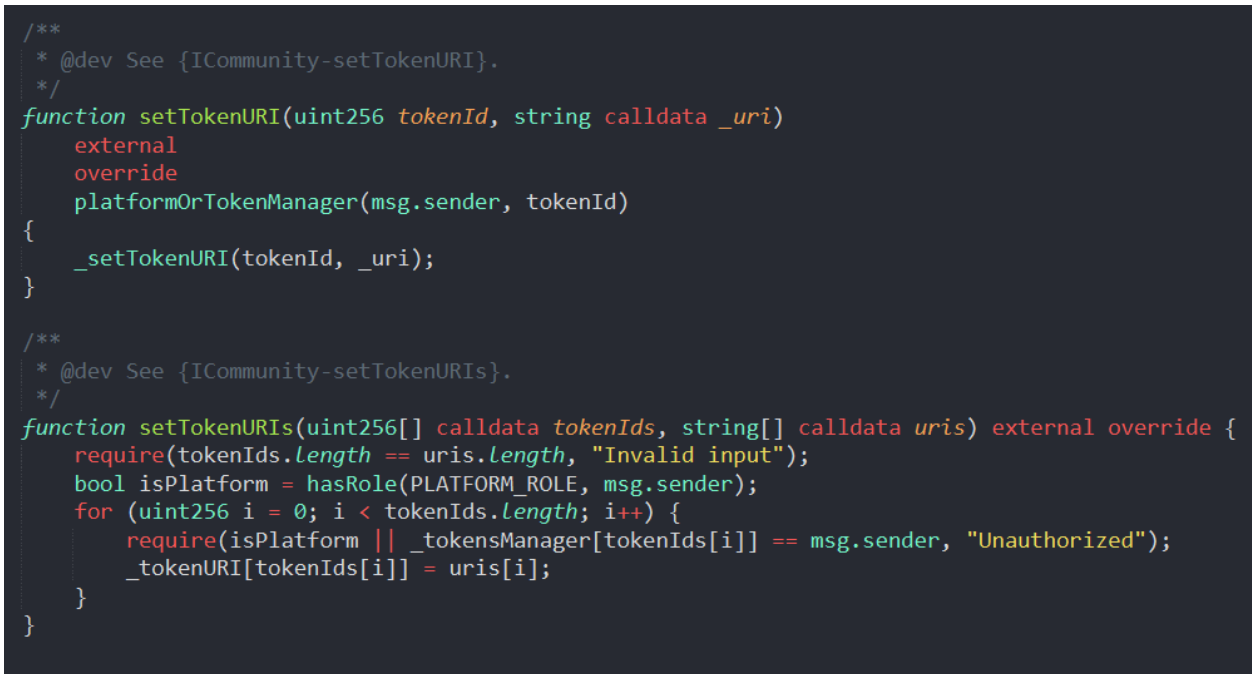
The code was refactored based on our recommendations.

3.8. Token URIs may be updated even when paused

- **Target:** BasicCommunity
- **Severity:** Low
- **Impact:** Low
- **Category:** Business Logic
- **Likelihood:** Low

Description

The functions `setTokenURIs`, `setTokenManagerBaseURI`, and `setTokenManagerBaseURIs` do not check whether the contract is currently paused (`whenNotPaused`).



```
/**
 * @dev See {ICommunity-setTokenURI}.
 */
function setTokenURI(uint256 tokenId, string calldata _uri)
    external
    override
    platformOrTokenManager(msg.sender, tokenId)
{
    _setTokenURI(tokenId, _uri);
}

/**
 * @dev See {ICommunity-setTokenURIs}.
 */
function setTokenURIs(uint256[] calldata tokenIds, string[] calldata uris) external override {
    require(tokenIds.length == uris.length, "Invalid input");
    bool isPlatform = hasRole(PLATFORM_ROLE, msg.sender);
    for (uint256 i = 0; i < tokenIds.length; i++) {
        require(isPlatform || _tokensManager[tokenIds[i]] == msg.sender, "Unauthorized");
        _tokenURI[tokenIds[i]] = uris[i];
    }
}
```

Impact

The pausing feature was designed to mitigate off-chain security incidents such as a compromised key, etc. Pausing, therefore, is an important tool for incident response. A compromised token manager could lead to Web2 vandalism by changing the URI.

Recommendations

Add the `whenNotPaused` modifier to the affected functions.

Remediation

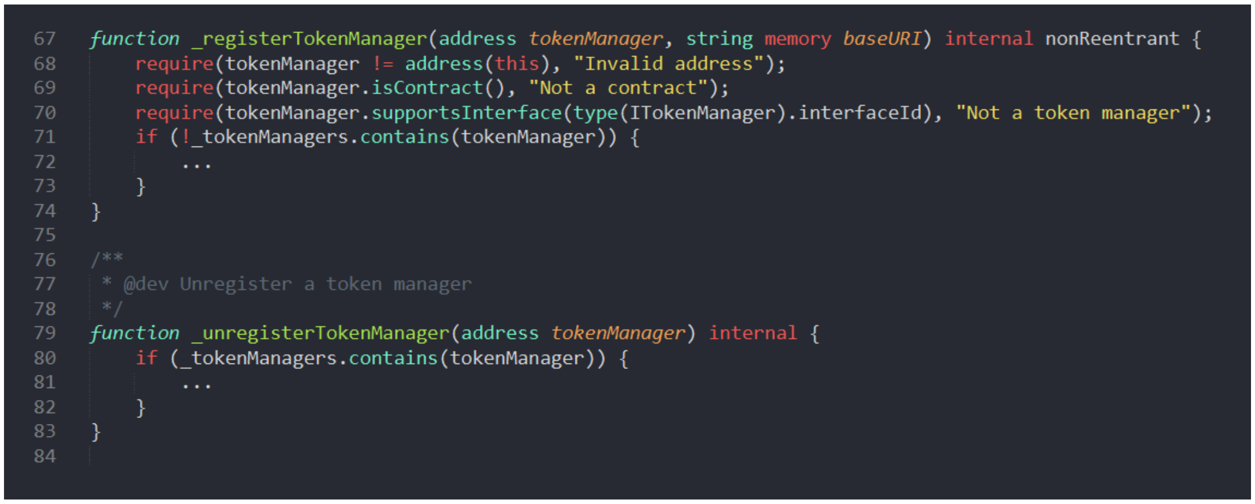
The affected methods were modified to include the `notWhenPaused` modifier.

3.9. Several functions can fail silently when registering and unregistering token managers

- **Target:** Community (and subclasses)
- **Severity:** Low
- **Impact:** Informational
- **Category:** Code Maturity
- **Likelihood:** n/a

Description

The function `_registerTokenManager` fails silently if the specified token manager is already registered. Similarly, `_unregisterTokenManager` fails silently if the specified token manager is not registered.



```
function _registerTokenManager(address tokenManager, string memory baseURI) internal nonReentrant {
    require(tokenManager != address(this), "Invalid address");
    require(tokenManager.isContract(), "Not a contract");
    require(tokenManager.supportsInterface(type(ITokenManager).interfaceId), "Not a token manager");
    if (!_tokenManagers.contains(tokenManager)) {
        ...
    }
}

/**
 * @dev Unregister a token manager
 */
function _unregisterTokenManager(address tokenManager) internal {
    if (_tokenManagers.contains(tokenManager)) {
        ...
    }
}

```

Impact

Failing silently can hide the existence of other more serious bugs and amplify their severity.

It is also a waste of gas to attempt to register/unregister a token manager when the operation is effectively a no-op.

Recommendations

Broadly speaking, it is best practice to simply revert than to fail silently. In this situation, the caller should be responsible for checking the pre-conditions if required. The if-statements in the code excerpt should be replaced with equivalent `require` statements instead.

Remediation

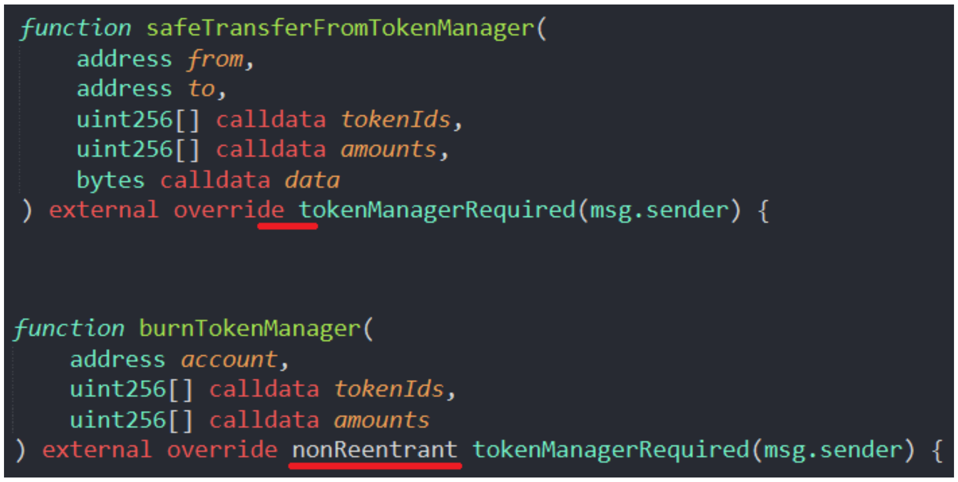
The affected functions were modified to revert instead of silently failing.

3.10. Internal inconsistency in marking functions non-reentrant

- **Target:** BasicCommunity
- **Severity:** Low
- **Impact:** Informational
- **Category:** Coding Mistakes
- **Likelihood:** n/a

Description

The function `safeTransferFromTokenManager` is not marked `nonReentrant`, whereas the similar functions `mintTokenManagerNew` and `burnTokenManager` are.



```
function safeTransferFromTokenManager(
    address from,
    address to,
    uint256[] calldata tokenIds,
    uint256[] calldata amounts,
    bytes calldata data
) external override tokenManagerRequired(msg.sender) {


function burnTokenManager(
    address account,
    uint256[] calldata tokenIds,
    uint256[] calldata amounts
) external override nonReentrant tokenManagerRequired(msg.sender) {
```

Impact

This currently has no impact, although it may be a sign of a minor coding oversight. A malicious or compromised token manager may be able to perform reentrancy attacks in the future if such opportunities or vulnerabilities arise. As it stands, however, this function is not currently vulnerable.

Recommendations

Consider adding the `nonReentrant` modifier to the affected function.

Remediation

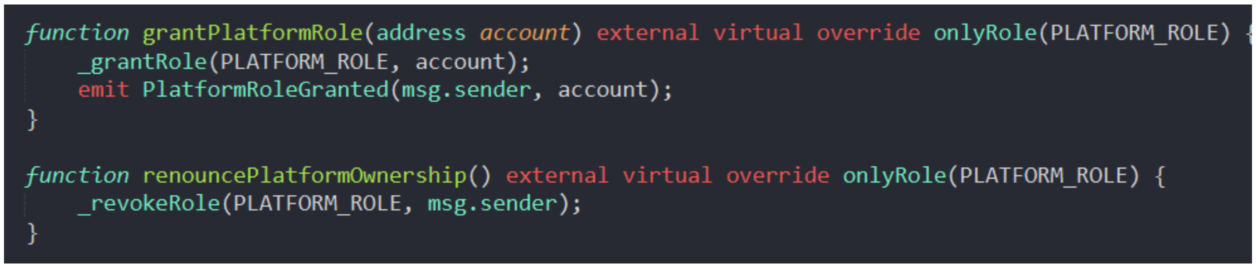
The issue has been acknowledged by Highlight, and a fix is pending.

3.11. Inconsistent emitting of events when roles are modified

- **Target:** CommunityAdmin
- **Severity:** n/a
- **Impact:** Informational
- **Category:** Coding Mistakes/Gas Optimization
- **Likelihood:** n/a

Description

Although the function `grantPlatformRole` emits an event when a new entity is granted the platform role, the dual function `renouncePlatformOwnership` does not emit an event when an entity renounces their platform role.



```
function grantPlatformRole(address account) external virtual override onlyRole(PLATFORM_ROLE) {
    _grantRole(PLATFORM_ROLE, account);
    emit PlatformRoleGranted(msg.sender, account);
}

function renouncePlatformOwnership() external virtual override onlyRole(PLATFORM_ROLE) {
    _revokeRole(PLATFORM_ROLE, msg.sender);
}
```

Note that the OpenZeppelin AccessControl implementation the class inherits from already emits an event for both granting and revoking roles.

Impact

This currently has no impact, although it may be a sign of a minor coding oversight. Emitting an event may also improve visibility into the platform, if this is desired. This may also harm the interoperability of the platform with off-chain applications.

Considering that AccessControl already emits events, this may also be a waste of gas.

Recommendations

1. Consider emitting an event when an entity renounces their platform role; or:
2. Do not emit a `PlatformRoleGranted` event in `grantPlatformRole`.

Remediation

The `PlatformRoleGranted` event was removed in favor of simply using the OpenZeppelin AccessControl events.

3.12. Redundant events are emitted

- **Target:** CommunityAdmin
- **Severity:** n/a
- **Impact:** Informational
- **Category:** Coding Mistakes/Gas Optimization
- **Likelihood:** n/a

Description

The functions pause and unpause functions emit CommunityPaused and CommunityUnpaused events, even though the OpenZeppelin Pausable implementation of _pause and _unpause already emits events. Although, those events are not parameterized by the message sender.

```
function pause() external virtual override onlyRole(PLATFORM_ROLE) {
    _pause();
    emit CommunityPaused(msg.sender);
}

function unpause() external virtual override onlyRole(PLATFORM_ROLE) {
    _unpause();
    emit CommunityUnpaused(msg.sender);
}
```

Impact

Considering that Pausable already emits events, this may also be a waste of gas.

Recommendations

1. Consider removing the CommunityPaused and CommunityUnpaused events.

Remediation

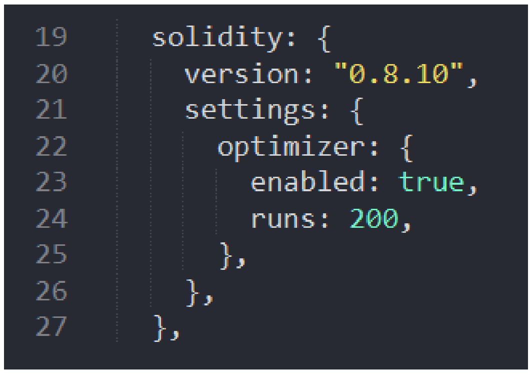
The affected functions no longer emit the `CommunityPaused` and `CommunityUnpaused` events, and instead rely on the OpenZeppelin Pausable events.

3.13. Solidity compiler optimizations can be problematic

- **Target:** hardhat.config.js
- **Severity:** Low
- **Impact:** Informational
- **Category:** Code Maturity
- **Likelihood:** n/a

Description

The Highlight contracts enable optional Solidity compiler optimizations. In the past, there have been several optimization bugs with security implications. Moreover, optimizations are actively being developed. Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised. This issue is a common finding in many audits.



```
  solidity: {
    version: "0.8.10",
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
```

Impact

High-severity security issues due to optimization bugs have occurred in the past. A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe. There is a possibility of latent bugs related to optimization, or new bugs introduced by future optimizations.

Recommendations

Consider disabling the optimizer if possible.

Remediation

The issue has been acknowledged by Highlight, and a fix is pending.

3.14. Potentially confusing variable names

- **Target:** Community
- **Severity:** Low
- **Impact:** Informational
- **Category:** Code Maturity
- **Likelihood:** n/a

Description

The variables `_tokenManagers` and `_tokensManager` are named very similarly while also frequently used together.

```
// Track tokenManagers
EnumerableSet.AddressSet internal _tokenManagers;

// Track which tokenManagers manage which tokens
mapping(uint256 => address) internal _tokensManager;
```

Impact

This is a potential source of developer confusion and a possible coding hazard. Otherwise, this is generally harmless.

Recommendations

Consider renaming one or more of the variables so the code is clearer.

Remediation

The issue has been acknowledged by Highlight, and a fix is pending.

4. Other interesting observations

In this section, we will discuss miscellaneous observations that we found during our audit that do not require an immediate remediation, but are nevertheless noteworthy and merit some consideration.

1. If the top-level creator suffers a private key disclosure, there is no mechanism in the current CommunityAdmin and BasicCommunity design to remediate this, short of a full migration to a new contract. There are several plausible designs for remediation protocols in the event of this possibility. For example: using a multi-signature contract for the top-level creator role to splits the key-bearing risk among multiple parties. Three such parties could be, for instance, the content creator, the Highlight platform, and a trusted third-party entity.

2. The platform is heavily centralized, and token managers exercise absolute control over users' tokens; although to our best understanding this is an intended design decision that enables content creators to better organize and lead their Communities. Of course, this bears the trade-off of increased centralization risk. We recommend that Highlight exercise careful control over private keys and other sensitive key material, or opt for professional key custody services.