shieldify

Phi Material

SECURITY REVIEW

Date: 27 July 2023

CONTENTS

1. About Shieldify

We are Shieldify Security – a company on a mission to make web3 protocols more secure, cost-efficient and user-friendly. Our team boasts extensive experience in the web3 space as both smart contract auditors and developers that have worked on top 100 blockchain projects with multi-million dollars in market capitalization.

Book an audit and learn more about us at shieldify.org or @ShieldifySec

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Phi Material

Phi is a novel web3 metaverse formed directly through ENS and wallet operations, facilitating effortless representation of on-chain identity. It motivates individuals to engage with diverse web3 protocols, fostering a positive feedback loop of benefits within the crypto ecosystem as a whole.

The Phi Material is an NFT metaverse game comprised of a number of smart contracts, each with a specific role in the game dynamics and user-generated content creation. Thus, the Phi ecosystem fosters an interactive environment where users are encouraged to regularly engage and produce content, driving the dynamics of the game while ensuring the continuous generation and distribution of assets.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired or even gas optimization techniques

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible incentivize
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Audit Summary

The audit duration lasted 10 days and a total of 240 hours have been spent by the three auditors – @ShieldifyMartin, @ShieldifyAnon and @ShieldifyGhost. This is the first audit for this specific part of the Phi protocol.

Overall, the codebase is well-written with the implementation of good gas optimization techniques (note, the report does not include a gas audit). Some foundational security mechanisms are also in use, for instance, the usage of OpenZeppelin's library – modifiers, smart contracts and others.

Despite this, this audit report adds and improves the security of Phi Land, as it contains findings that are detrimental to the overall health of the protocol. The most severe ones cover exploitation via the burning mechanism, signature malleability and insecure randomness generation, among other less severe findings. There is some room for improvement in the documentation, in a sense that in some places it is relatively scarce. The test coverage is very good and comprehensive.

We also wanted to dedicate a small section about Phi Land's blockchain of choice – Polygon. Its PoS chain is an incredible alternative to Ethereum, providing very cheap transaction executions and ultra-fast confirmation times. And although it might easily be the best blockchain for game/metaverse projects like Phi, it often experiences block reorganizations, also known as reorgs. They happen almost on a daily basis. As of the 11th of July, the average `reorgDepth` on Polygon is a little over 4.5 blocks.

While there is not much that can be done about this, we would like to suggest to the Phi team to incorporate some user protection against chain reorgs at least via their dApp's GUI dashboard. Specifically, they could require that every state-changing function call has at least five block confirmations. And while this is still not a 100% reorg-protected approach (because chain reorgs deeper than 5 blocks are also frequent), it will handle the larger portion of them, while still not impacting the UX that much (more confirmations equal more waiting time for the user).

We would also like to thank the Phi Land team for being very responsive and for providing clarifications and detailed responses to all of our questions. They are an amazing project and Shieldify is happy to be part of it.

5.1 Protocol Summary

Project Name	**PHI Material**
Repository	PHIMaterial
Type of Project	NFT Metaverse Game
Audit Timeline	10 days
Review Commit Hash	355376812ba1e2eeed97d5447c2afea83a3ca8f1
Fixes Review Commit Hash	1f0f591138a4b545e5c1d6221326cbdbc5b14f23

5.2 Scope

The following smart contracts were in the scope of the audit:

File	nSLOC
src/PhiDaily.sol	157
src/EmissionLogic.sol	74
src/object/MaterialObject.sol	70
src/object/CraftObject.sol	71
src/CraftLogic.sol	127
src/UGCCraftLogic.sol	132
src/object/UGCObject.sol	58
src/UGCObjectFactory.sol	27
src/utils/BaseCraft.sol	37
src/utils/BaseObject.sol	147
src/utils/BaseUGC.sol	9
src/utils/MultiOwner.sol	25
src/interfaces/ICatalyst.sol	3
src/interfaces/ICraftLogic.sol	3
src/interfaces/ICraftObject.sol	8
src/interfaces/IEmissionLogic.sol	3
src/interfaces/IERC20.sol	3
src/interfaces/IMaterialObject.sol	8
src/interfaces/IPhiDaily.sol	8
src/interfaces/IUGCObject.sol	3
Total	**973**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **1**
- **Medium** issues: **3**
- **Low** issues: **2**
- **Informational** issues: **15**

ID	Title	Severity
[C-01]	Anyone Can Burn Any NFTs of Other Users in the `Material`, `UGC` and `Craft` Objects	Critical
[M-01]	Signature Does Not Contain a Deadline, Making It Reusable	Medium
[M-02]	Insecure Generation of Randomness Used for Token Determination Logic	Medium
[M-03]	Centralization Risk in Multiple Places	Medium
[L-01]	Usage of `ecrecover` Should Be Replaced with Usage of OpenZeppelin's `ECDSA` Library	Low
[L-02]	Lack of Consistency and Misleading Naming in Checks	Low
[I-01]	No Check For the Existence of a Recipe With the Same Name May Cause Confusion About Which Recipe Should Be Used	Informational
[I-02]	Incomplete Parameters in Emitted Events	Informational
[I-03]	Not Using the Named `return` Variables in a Function	Informational
[I-04]	The `nonReentrant` Modifier Should Occur Before All Other Modifiers	Informational
[I-05]	Mixed Use of Custom Errors and Revert Strings	Informational
[I-06]	Create a Modifier Only if It Will Be Used in More than One Place	Informational
[I-07]	Interfaces & Functions in the Interface Not Used	Informational
[I-08]	Hardcoded `GelatoRelay1BalanceERC2771` Address	Informational
[I-09]	Redundant Functions	Informational
[I-10]	No Need to Initialize Variables with Default Values	Informational
[I-11]	Unused/Commented Constant	Informational
[I-12]	Misleading comment for `UGCCraftLogic.createRecipe` function	Informational
[I-13]	Use Inheritance Instead of Duplicated Code	Informational
[I-14]	Missing / Incomplete NatSpec	Informational
[I-15]	Function Ordering Does not Follow the Solidity Style Guide	Informational

7.Findings

[C-01] Anyone Can Burn Any NFTs of Other Users in the `Material`, `UGC` and `Craft` Objects

Severity

Critical Risk

Description

The `MaterialObject` and `CraftObject` contracts represent distinct digital assets and the corresponding materials required for their creation, respectively. These assets can take various forms,

such as in-game items or characters.

The same principle applies to the `UGCObject` contract, which enables users to generate their unique assets known as User Generated Content (UGC). These UGCObjects serve as materials and catalysts in `CraftLogic` recipes and are integrated into the Phi ecosystem.

The vulnerability permits arbitrary users to destroy NFTs from any account, potentially resulting in the unauthorized loss of these digital assets.

Based on the provided code, it appears that anyone can invoke the `burnObject` and `burnBatchObject` functions while specifying any account address from which to burn NFTs (digital assets). In a secure system, only the account holder should possess the ability to burn their own tokens. This creates a vulnerability whereby malicious users can burn NFTs belonging to other users without their consent.

Location of Affected Code

File: src/object/MaterialObject.sol#L145-L153

File: src/object/UGCObject.sol#L102-L110

File: src/object/CraftObject.sol#L158-L166

```
// Function to burn an object
function burnObject(address from, uint256 tokenid, uint256 amount)
    external override {
     super._burn(from, tokenid, amount);
}

// Function to burn a batch of objects
function burnBatchObject(address from, uint256[] memory tokenids, uint256
    [] memory amounts) external override {
     super._burnBatch(from, tokenids, amounts);
}
```

Recommendation

To address this vulnerability, it is crucial to implement a check in the code to verify that the caller is the rightful owner of the NFT intended for burning. This verification ensures that only the NFT holder has the authority to burn their own NFTs, effectively preventing unauthorized users from burning NFTs they do not possess.

- If the function `burnObject` is called only external. Remove the `address from` parameter from `burnObject` function and put `msg.sender` as a parameter of `super._burn` function. The same goes for the `burnBatchObject` function.

`Example:`

```
function burnObject(uint256 tokenid, uint256 amount) external override {
     super._burn(msg.sender, tokenid, amount);
}
```

- If the function `burnObject` is called only from other contracts like function `MaterialObject.burnObject` is called from `CraftLogic/UGCCraftLogic` contract.Add a modifier like `onlyCraftLogic` on `MaterialObject.burnObject` function and then in `CraftLogic/UGCCraftLogic` contracts call `MaterialObject.burnObject` with the parameter `from` as `msg.sender`. The same goes for the `burnBatchObject` function.

Example:

File: `src/object/MaterialObject.sol`

```
function burnObject(address from, uint256 tokenid, uint256 amount)
    external override onlyCraftLogic {
     super._burn(from, tokenid, amount);
}
```

File: `src/CraftLogic.sol`

```
IMaterialObject(material.tokenAddress).burnObject(_msgSender(), material.
    tokenid, material.amount);
```

Note:

If you implement our second suggestion, consider calling the **`burnBatchObject`** function in the **`CraftLogic/UGCCraftLogic`** contracts as it is currently only called externally because if a modifier is added to call this function only from the **`CraftLogic/UGCCraftLogic`** contracts there will be no other option to call it.

Or remove **`burnBatchObject`** function from the three contracts **`CraftObject.sol`**, **`MaterialObject.sol`** and **`UGCObject.sol`**, if you are not going to call this function in **`CraftLogic/UGCCraftLogic`** contracts.

Team Response

Acknowledged and fixed by creating a whitelist functionality for craft logic, restricting calling burn functions and removed **`burnBatchObject`** function from the three contracts **`CraftObject.sol`**, **`MaterialObject.sol`** and **`UGCObject.sol`**.

[M-01] Signature Does Not Contain a Deadline, Making It Reusable

Severity

Medium Risk

Description

The **`PhiDaily.sol`** contract has the **`claimMaterialObject`** function that allow a user to claim a material object directly. These then call the private **`_processClaim()`** function to check that the coupon has been signed by the admin signer.

The **`digest`** variable in **`_processClaim()`** accepts an **`eventId`**, **`logicId`** and **`msgSender()`**, but it does not accept any variable that defines a timeframe in which this signature is valid and does not check if such variable has passed a certain amount of time. Without such a variable and a corresponding check for it, the issued signature essentially becomes timeless and reusable again.

This is also applicabe in the other functions that call the **`_processClaim()`** function, namely - **`batchClaimMaterialObject`**, **`claimMaterialObjectByRelayer`** and **`batchClaimMaterialObjectByRelayer`** .This can negatively impact the business logic of the protocol, as one material object can be claimed more than once.

Location of Affected Code

File: src/PhiDaily.sol#L248-L258

```
// Function to claim a material object.
function claimMaterialObject(
    uint32 eventid,
    uint16 logicid,
    Coupon memory coupon
)
    external
    onlyIfAlreadyClaimed(eventid, logicid)
    nonReentrant
{

function _processClaim(uint32 eventid, uint16 logicid, Coupon memory
   coupon) private {
    // Check that the coupon sent was signed by the admin signer
    bytes32 digest = keccak256(abi.encode(eventid, logicid, _msgSender())
       );
```

Recommendation

Consider adding a deadline check in the `claimMaterialObject()` function and an `expiresIn` /`deadline` variable either as an argument that is passed to the `claimMaterialObject()` function or in the `Coupon` struct. The `claimMaterialObject()` function should also contain a check if the `expiresIn` / `deadline` variable is valid:

`Example:`

```
    function claimMaterialObject(
        uint32 eventid,
        uint16 logicid,
++      uint256 expiresIn
        Coupon memory coupon
    )
        external
        onlyIfAlreadyClaimed(eventid, logicid)
        nonReentrant
    {
++     if (expiresIn <= block.timestamp){
++        revert SignatureExpired()
++     }
        _processClaim(eventid, logicid, coupon, expiresIn);
    }

    function _processClaim(uint32 eventid, uint16 logicid, Coupon memory
       coupon, expiresIn) private {
    // Check that the coupon sent was signed by the admin signer
++   bytes32 digest = keccak256(abi.encode(eventid, logicid, _msgSender()
   , expiresIn));
```

Team Response

Acknowledged and fixed by setting expiration period `expiresIn` to `Coupon` and added/modified tests for it.

[M-02] Insecure Generation of Randomness Used for Token Determination Logic

Severity

Medium Risk

Description

`EmissionLogic.sol` contains the `determineTokenByLogic()` function that determines the rarity and `tokenId` of a `MaterialObject`. It generates a uint256 `random` value that relies on variables like `block.timestamp` and `tx.origin` as a source of randomness is a common vulnerability, as the outcome can be predicted by calling contracts or validators. In the context of blockchains, the best and most secure source of randomness is that which is generated off-chain in a verified manner.

The function also uses block.prevrandao whose random seed calculation is by epoch basis, which means that entropy within 2 epochs is low and sometimes even predictable. Users of PREVRANDAO would need to check that a validator has provided a block since the last time they called PREVRANDAO. Otherwise, they won't necessarily be drawing statistically independent random outputs on successive calls to PREVRANDAO.

In the context of Phi's business logic, improper insecure randomness generation could allow malicious actors to mint more rare/exclusive `MaterialObject` items.

Location of Affected Code

File: src/EmissionLogic.sol#L49

```
uint256 random = uint256(keccak256(abi.encodePacked(block.prevrandao,
    block.timestamp, tx.origin)));
```

Recommendation

Consider using a decentralized oracle for the generation of random numbers, such as `VRF` by Chainlink. It is important to take into account the `requestConfirmations` variable that will be used in the `VRFv2Consumer` contract when implementing VRF. The purpose of this value is to specify the minimum number of blocks you wish to wait before receiving randomness from the Chainlink VRF service. The inclusion of this value is motivated by the occurrence of chain reorganizations, which result in the alteration of blocks and transactions. Addressing this concern is crucial for the successful implementation of this application on the Polygon network because it is prone to block reorgs and they happen almost on a daily basis.

Shieldify recommends setting the `requestConfirmations` value to at least 5, so that the larger portion of the reorgs that happen are properly taken into account and won't impact the randomness generation.

Team Response

Acknowledged, but currently will not be mitigated as the team does not have plans to implement Chainlink functionality yet.

[M-03] Centralization Risk in Multiple Places

Severity

Medium Risk

Description

For `setEmissionLogic` and `setMaterialObject` in `PhiDaily.sol`, it's documented that `emissionLogic` and `materialObject` should be contracts but in reality, the admin can set any address.With `setTreasuryAddress`,`setMaxClaimed`,`setRoyalityFee` and `setSecondaryRoyalityFee` in `BaseObject.sol` owner can set not validated values which can result in loss of funds for users.

Location of Affected Code

File: src/PhiDaily.sol

```
function setEmissionLogic(address _emissionLogic) external onlyOwner {
function setMaterialObject(address _materialObject) external onlyOwner {
```

File: src/utils/BaseObject.sol

```
function setTreasuryAddress(address payable newTreasuryAddress) external
    onlyOwner {
function setMaxClaimed(uint256 tokenId, uint256 newMaxClaimed) public
    virtual onlyOwner {
function setRoyalityFee(uint256 newRoyalityFee) external onlyOwner {
function setSecondaryRoyalityFee(uint256 newSecondaryRoyalty) external
    onlyOwner {
```

Recommendation

Add proper address validation and upper-bound checks for the fees setter functions despite the fact all these functions are callable only by the owner.

Team Response

Acknowledged, Timelock and Multisig will implemented.

[L-01] Usage of `ecrecover` Should Be Replaced with Usage of OpenZeppelin's `ECDSA` Library

Severity

Low Risk

Description

Signature malleability is one of the potential issues with `ecrecover`. The `_isVerifiedCoupon` function calls the Solidity `ecrecover()` function directly to verify the given signatures. However, the `ecrecover()` retrieves the address of the signer via a provided `v`, `r` and `s` signatures. However, due to the nature of the elliptic curve digital signing algorithm (ECDSA), there are two valid points (i.e. two

`s` values) on the elliptic curve with the exact same `r` value. An attacker can pretty easily compute the other valid `s` value on the elliptic curve that will return the same original signer, making the signature malleable.

`ecrecover()` allows malleable (non-unique) signatures and thus is susceptible to replay attacks. This means that multiple signatures will be considered valid, which will lead to wrong claim logic.

Location of Affected Code

File: src/PhiDaily.sol#L161

```
address signer = ecrecover(digest, coupon.v, coupon.r, coupon.s);
```

Recommendation

Use the `recover()` function from OpenZeppelin's ECDSA library to verify the uniqueness of the signature. Using this library implements a check on the value of the `s` variable to ensure that only one of its possible inputs is valid. Ensure that you are using a version `> 4.7.3` for there was a critical bug `>= 4.1.0 < 4.7.3`.

Team Response

Acknowledged and fixed by using `ECDSA` library.

[L-02] Lack of Consistency and Misleading Naming in Checks

Severity

Low Risk

Description

There is a different check implementation in `onlyIfAlreadyClaimed` and `onlyIfAlreadyClaimedMultiple` modifiers for the same check. Additionally, both modifier names ensure that the execution will revert if the user hasn't claimed already but actually, it is the opposite, it reverts if the user has claimed.

Location of Affected Code

File: src/PhiDaily.sol#L121

```
if (dailyClaimedStatus[_msgSender()][eventid][logicid] > 0) {
```

File: src/PhiDaily.sol#L134

```
if (dailyClaimedStatus[_msgSender()][eventids[i]][logicids[i]] ==
    _CLAIMED) {
```

Recommendation

Implement the comparison with the claimed status in both modifiers and correct the naming for sure.

Team Response

Acknowledged and fixed as proposed.

[I-01] No Check For the Existence of a Recipe With the Same Name May Cause Confusion About Which Recipe Should Be Used

Severity

Informational

Description

Both `UGCCraftLogic.sol` and `CraftLogic.sol` allow for the creation of recipes for new materials. The functions accept a `name` variable that allows for the naming of the recipe.

However, there is no check in the code if a recipe with the same name exists. This can cause confusion among users when they need to, later on, choose between two already existing recipes with the exact same names. This may lead to the crafting of non-desirable objects and confusion for the user.

Location of affected code

File: src/CraftLogic.sol#L155-L164

```
function createRecipe(
    string calldata name,
    Material[] calldata materials,
    Artifacts[] calldata artifacts,
    Catalyst calldata catalyst
)
    external
    override
    onlyOwner
{
```

File: src/UGCCraftLogic.sol#L171-L179

```
function createRecipe(
    string calldata name,
    Material[] calldata materials,
    Artifacts[] calldata artifacts,
    Catalyst calldata catalyst
)
    external
    override
{
```

Recommendation

Consider implementing a check for existing recipes with the same names or remove the `name` variable altogether.

Team Response

Acknowledged and fixed by removing the name variable in recipes altogether.

[I-02] Incomplete Parameters in Emitted Events

Severity

Informational

Description

The `setAdminSigner` event in `PhiDaily.sol` should also emit the old admin signer value that is being overwritten. The same goes for `SetEmissionLogic` and `SetMaterialObject`.

Location of Affected Code

File: src/PhiDaily.sol#L156

```
emit SetAdminSigner(adminSigner);
```

File: src/PhiDaily.sol#L179

```
emit SetEmissionLogic(emissionLogic);
```

File: src/PhiDaily.sol#L188

```
emit SetMaterialObject(materialObject);
```

Recommendation

Consider modifying the `SetAdminSigner`, `SetEmissionLogic` and `SetMaterialObject` events by emitting the old value.

Team Response

Acknowledged and fixed as proposed and also tests for events emission were added.

[I-03] Not Using the Named `return` Variables in a Function

Severity

Informational

Description

Named return variables are defined but never used which affects the code readability. Not using them wastes deployment gas as well.

Location of Affected Code

File: src/PhiDaily.sol#L205

```
function checkClaimCount(address sender) external view returns (uint256
    count) {
```

File: src/PhiDaily.sol#L210

```
function checkClaimEachCount(address sender, uint256 tokenid) external
    view returns (uint256 count) {
```

File: src/PhiDaily.sol#L215-L223

```
function checkClaimStatus(
    address sender,
    uint256 eventid,
    uint256 logicid
)
    external
    view
    returns (uint256 status)
{
```

File: src/utils/BaseObject.sol#L198-L206

```
function royaltyInfo(
    uint256,
    uint256 salePrice
)
    external
    view
    override
    returns (address receiver, uint256 royaltyAmount)
{
```

File: src/utils/BaseCraft.sol#L54

```
function getRecipe(uint256 recipeId) external view virtual returns (
    Recipe memory recipe) {
```

Recommendation

Consider either using these named return variables or removing them.

Team Response

Acknowledged and fixed as proposed.

[I-04] The `nonReentrant` Modifier Should Occur Before All Other Modifiers

Severity

Informational

Description

This is a best practice to protect against re-entrancy in other modifiers. It can additionally reduce gas costs if this modifier occurs before all others.

If a function has multiple modifiers they are executed in the order specified. If checks or logic of modifiers depend on other modifiers this has to be considered in their ordering. Some functions have multiple modifiers with one of them being **`nonReentrant`** which prevents reentrancy on the functions. This should ideally be the first one to prevent even the execution of other modifiers in case of reentrancies.

Location of Affected Code

File: src/CraftLogic.sol#L291

```
function craftByRelayer(uint256 recipeId) external onlyGelatoRelay
    nonReentrant {
```

File: src/PhiDaily.sol

```
function claimMaterialObject(
function batchClaimMaterialObject(
function claimMaterialObjectByRelayer(
function batchClaimMaterialObjectByRelayer(
```

File: src/UGCCraftLogic.sol#L306

```
function craftByRelayer(uint256 recipeId) external onlyGelatoRelay
    nonReentrant {
```

File: src/object/CraftObject.sol#L146

```
function craftObject(address to, uint256 tokenid, uint256 amount)
    external onlyCraftLogic nonReentrant {
```

File: src/object/UGCObject.sol#L93

```
function craftObject(address to, uint256 tokenid, uint256 amount)
    external override onlyCraftLogic nonReentrant {
```

Recommendation

Reorder the modifiers so that **`nonReentrant`** is first in line.

Team Response

Acknowledged and fixed as proposed.

[I-05] Mixed Use of Custom Errors and Revert Strings

Severity

Informational

Description

In some parts of the code, `custom errors` are declared and later used `Custom Errors`, while in other parts, classic revert strings are used in `Require Statements`.

Instead of using error strings, custom errors can be used, which would reduce deployment and run-time costs.

Location of Affected Code

Most contracts.

Recommendation

Consider using only custom errors as they are more gas efficient.

Team Response

Acknowledged, fixed and tested as proposed.

[I-06] Create a Modifier Only if It Will Be Used in More than One Place

Severity

Informational

Description

There is no need to create a separate modifier unless it will be used in more than one place. If this is not the case, simply add the modifier code to the function instead.

Location of Affected Code

File: src/object/UGCObject.sol#L48-L51

```
modifier onlyCraftLogic() {
    if (_msgSender() != craftLogicAddress) revert UnauthorizedCaller();
    _;
}
```

File: src/object/CraftObject.sol#L69-L72

```
modifier onlyCraftLogic() {
    if (_msgSender() != craftLogic) revert UnauthorizedCaller();
    _;
}
```

Recommendation

Add the modifier logic into the function directly.

Team Response

Acknowledged, will not be mitigated.

[I-07] Interfaces & Functions in the Interfaces Not Used

Severity

Informational

Description

There are a few unused interfaces and functions on the codebase. These interfaces and functions should be cleaned up from the code if they have no purpose.

Location of Affected Code

Interfaces are never used:

- File: src/interfaces/IERC20.sol
- File: src/interfaces/ICraftLogic.sol
- File: src/interfaces/ICraftObject.sol
- File: src/interfaces/IPhiDaily.sol

Functions in the interface are never used:

File: src/interfaces/IUGCObject.sol

- function balanceOf
- function burnObject
- function burnBatchObject

File: src/interfaces/IMaterialObject.sol

- struct Size
- function getSize
- function balanceOf
- function safeTransferFrom
- function burnBatchObject

Recommendation

Consider removing the unnecessary interfaces and functions since are not used anywhere in the codebase.

Team Response

Acknowledged and fixed as proposed.

[I-08] Hardcoded `GelatoRelay1BalanceERC2771` Address

Severity

Informational

Description

Currently, the **`GELATO_RELAY`** address is hardcoded as a constant, the **`PhiDaily`**, **`CraftLogic`** and **`UGCCraftLogic`** smart contracts may encounter difficulties in interacting with the address, resulting in erroneous behavior and potential malfunctions.

Location of Affected Code

File: src/PhiDaily.sol#L287

File: src/CraftLogic.sol#L279

File: src/UGCCraftLogic.sol#L294

```
address constant GELATO_RELAY=0xd8253782c45a12053594b9deB72d8e8aB2Fca54c;
```

Recommendation

To mitigate this recommendation, it is advisable to pass the **`GELATO_RELAY`** as a constructor parameter when deploying the contract instead of relying on a fixed address. Additionally, add a function to change this address later, this function should be callable only by the owner. By allowing the addresses to be configured, smart contracts can accommodate changes in contract addresses.

Team Response

Acknowledged and fixed by implementing **`setGelatoRelay`** function.

[I-09] Redundant Functions

Severity

Informational

Description

There are already **`craftByRelayer()`**, **`claimMaterialObject()`** and **`batchClaimMaterialObject()`** functions that can be called externally by anyone and therefore there is no purpose to have functions doing the same that can be called only by the GelatoRelay address, as this only adds weight to the codebase.

Location of Affected Code

File: src/UGCCraftLogic.sol#L306

```
function craftByRelayer(uint256 recipeId) external onlyGelatoRelay
    nonReentrant {
```

File: src/PhiDaily.sol#L299

```
function claimMaterialObjectByRelayer(
function batchClaimMaterialObjectByRelayer(
```

Recommendation

Consider removing the `craftByRelayer()`, `claimMaterialObjectByRelayer()` and `batchClaimMaterialObjectByRelayer()` that is meant to be called only by the Gelato Relay, as they are redundant.

Team Response

The Phi team decided to add separate events for the different functions.

[I-10] No Need to Initialize Variables with Default Values

Severity

Informational

Description

If a variable is not set/initialized, the default value is assumed (0, false, 0x0 ... depending on the data type). Saves **8 gas** per instance.

Location of Affected Code

In all contracts where there is a for loop like this:

```
-- for (uint256 i = 0; ....
++ for (uint256 i; ....
```

Recommendation

Do not initialize variables with their default values.

Team Response

Acknowledged and fixed as proposed.

[I-11] Unused/Commented Constant

Severity

Informational

Description

Within the `PhiDaily.sol` contract, `_NOT_CLAIMED` constant seems unused and redundant since it is commented. It should be cleaned up from the code if it has no purpose as it affects code readability.

Location of Affected Code

File: src/PhiDaily.sol#L39

```
// uint256 private constant _NOT_CLAIMED = 0;
```

Recommendation

Consider removing the whole line.

Team Response

Acknowledged and fixed as proposed.

[I-12] Misleading comment for `UGCCraftLogic.createRecipe` function

Severity

Informational

Description

As you may notice, there is a comment that says that this function can only be called by the `owner` of the recipe, which is not the case, because the function does not have a limiting modifier `onlyRecipeCreator` like other functions:

- `UGCCraftLogic.updateRecipe`
- `UGCCraftLogic.changeRecipeStatus`

This affects code readability and makes confusing other users/auditors, as the comment does not correspond to the logic of the function.

Location of Affected Code

File: src/UGCCraftLogic.sol#L163

```
/*
Requirements:
* - The caller must be the owner.
* - The materials and artifacts arrays must not be empty.
*
* @param name the name of the recipe
* @param materials the materials required for the recipe
* @param artifacts the artifacts produced by the recipe
* @param catalyst the catalyst required for the recipe
*/
function createRecipe(
    string calldata name,
    Material[] calldata materials,
    Artifacts[] calldata artifacts,
    Catalyst calldata catalyst
)
    external
    override
{
```

Recommendation

Consider removing this comment from the createRecipe function.

```
/* * Requirements: */
-- /* - The caller must be the owner. */
/* - The materials and artifacts arrays must not be empty.
*
* @param name the name of the recipe
* @param materials the materials required for the recipe
* @param artifacts the artifacts produced by the recipe
* @param catalyst the catalyst required for the recipe
*/
function createRecipe(
    string calldata name,
    Material[] calldata materials,
    Artifacts[] calldata artifacts,
    Catalyst calldata catalyst
)
    external
    override
{
```

Team Response

Acknowledged and fixed by removing unnecessary comment.

[I-13] Use Inheritance Instead of Duplicated Code

Severity

Informational

Description

The implementation of `CraftLogic` and `UGCCraftLogic` is the same with the exception of the added access control in `UGCCraftLogic`.

Location of Affected Code

Files:

- `CraftLogic.sol`
- `UGCCraftLogic.sol`

Recommendation

Instead of duplicating code from `CraftLogic` in `UGCCraftLogic`, consider adding inheritance and then implementing the proper access control.

Team Response

Acknowledged, we will update comments, but not follow NetSpec.

[I-14] Missing / Incomplete NatSpec

Severity

Informational

Description

(`@notice`, `@dev`, `@param` and `@return`) are missing for some functions. Given that NatSpec is an important part of code documentation, this affects code comprehension, audibility, and usability.

This might lead to confusion for other auditors/developers that are interacting with the code.

Location of Affected Code

In some contacts.

Recommendation

Consider adding full NatSpec comments for all functions where missing to have complete code documentation for future use.

Team Response

Acknowledged and partially fixed.

[I-15] Function Ordering Does not Follow the Solidity Style Guide

Severity

Informational

Description

One of the guidelines mentioned in the style guide is to order functions in a specific way to improve readability and maintainability. By following this order, you can achieve a consistent and logical structure in your contract code.

Location of Affected Code

Most smart contracts.

Recommendation

It is recommended to follow the recommended order of functions in Solidity, as outlined in the Solidity style guide.

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

Team Response

Acknowledged, will not be mitigated.

shieldify

Thank you!