P12 Code Review & Pre-Audit - Summary

> Prepared for the P12 team, by Yos Riady

I. Executive summary

This report presents the results of an initial code review and audit of the P12 protocol’ s smart contracts, specifically the airdrop and economy contracts. The review was conducted over multiple sub-projects from April 2022 to July 2022.

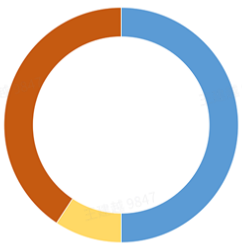
II. Project objectives

The review is focused on the following repos:

- The P12 Airdrop contracts: https://github.com/ProjectTwelve/p12-airdrop-contracts
- The P12 economy contracts: https://github.com/ProjectTwelve/contracts
- The P12 Badge Merge contracts: https://github.com/ProjectTwelve/Badge-Merge-Contract

The review focuses on **security best practices, gas efficiency, readability, and general Solidity conventions and design patterns.**

III. Findings



43	0	8	35
All Findings	Unresolved	Acknowledged	Resolved

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.

- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

3.A Major Be aware of hash collision risks in airdrop contract from the use of abi.encodePacked

Description
The airdrop contracts makes use of off-chain signatures and on-chain verification for users to claim airdropped tokens. Be aware that this is a critical operation that should be evaluated carefully for security risks. Malicious actors should not be able to exploit the airdrop mechanism to acquire more tokens than they should. Specifically, there is the risk hash collisions being used to potentially claim tokens belonging to someone else. *In other words, can someone with just a little bit of grinding, find any input that produces the right signature?* For reference, please review: · https://swcregistry.io/docs/SWC-133 · https://medium.com/swlh/new-smart-contract-weakness-hash-collisions-with-multiple-variable-length-arguments-dc7b9c84e493
Recommendation
Consider using abi.encode instead of encodePacked to mitigate this risk.
Code
https://github.com/ProjectTwelve/p12-airdrop-contracts/blob/main/contracts/airdrop/P12AirdropSteamDeveloper.sol#L44
Status

Fixed in https://github.com/ProjectTwelve/p12-airdrop-contracts/commit/341a6ac3e6522a0ce70fc38230fe2a3ad4f8516a

3.B Medium Missing zero address check in airdrop contract withdrawal function

Description

Once the airdrop claim period ends, it’ s possible to accidentally burn the remaining tokens when calling the `transfer(address dest)` function, by incorrectly sending to the zero address 0x0000....

require(address != address(0))

Recommendation

To improve safety, add a require statement to ensure the destination address is not the zero address.

Code

https://github.com/ProjectTwelve/p12-airdrop-contracts/blob/main/contracts/airdrop/P12AirdropSteamDeveloper.sol#L100-L109

Status

Fixed in https://github.com/ProjectTwelve/p12-airdrop-contracts/commit/341a6ac3e6522a0ce70fc38230fe2a3ad4f8516a

3.C Minor Mark variables as immutable to improve gas efficiency

Description

For variables that are only set during contract creation, marking them as immutable makes them significantly cheaper to read.

For more details, see: https://blog.soliditylang.org/2020/05/13/immutable-keyword/

SLOAD evm opcode -> very expensive

Recommendation

Mark variables as *immutable* where relevant.

Code

https://github.com/ProjectTwelve/p12-airdrop-contracts/blob/main/contracts/airdrop/P12AirdropSteamDeveloper.sol#L14-L15

Status

Fixed in https://github.com/ProjectTwelve/p12-airdrop-contracts/commit/341a6ac3e6522a0ce70fc38230fe2a3ad4f8516a

3.D Minor Consider renaming airdrop contract withdrawal function

Description

Consider renaming the `transfer(address dest)` function to `withdraw(address dest)` to improve readability. The term transfer is overloaded and is frequently used to refer to ERC20 transfers. It is recommended to not re-use this term in non-token contracts to reduce confusion.

Recommendation

Rename the airdrop withdrawal function to `withdraw()` for better clarity.

Code

https://github.com/ProjectTwelve/p12-airdrop-contracts/blob/main/contracts/airdrop/P12AirdropSteamDeveloper.sol#L100

Status

Fixed in https://github.com/ProjectTwelve/p12-airdrop-contracts/commit/341a6ac3e6522a0ce70fc38230fe2a3ad4f8516a

3.E Minor Consider labelling storage variables and function arguments differently

Description

Consider adding an underscore (`_`) prefix or suffix consistently for either storage variables or function

arguments. Sometimes this is done (like here) but it's not done consistently across all the contracts.

Because of this, in some functions it's not immediately clear if a variable is a storage variable or function argument.

Recommendation

To summarize, mark storage variables more clearly compared to function arguments. Some possible naming conventions include:

- Adding a _ prefix for all storage variables (e.g. `address public _p12` instead of `address public p12`) - OpenZeppelin follows this convention
- OR, Adding a _ prefix for all function arguments.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L116-L123

Status

Acknowledged, Wontfix

3.F Minor Consider emitting events for important operations

Description

Consider adding an event for important operations such as owner-only functions. This helps improve visibility and allows monitoring tools to subscribe to critical events.

Recommendation

Emit an event when `setInfo()` and other critical operations are called.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L98-L103

Status

Acknowledged, Fixed

3.G Minor Pragma version should be locked

Description

Most contracts uses a floating ^0.8.x compiler version, instead of a fixed version number.

Recommendation

Lock the pragma version to a single fixed 0.8.x version, preferably the latest.

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

This is especially important for upgradable contracts to ensure consistent bytecode generation.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

https://github.com/ethereum/solidity/releases

Code

All the contracts use floating pragmas.

Note: https://github.com/ProjectTwelve/p12-airdrop-contracts/blob/main/contracts/airdrop/P12AirdropSteamDeveloper.sol is still using floating pragmas

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/617cd691d9012572063c0f9446b783846e469875

3.H Minor Commented code should be removed

Description

There's a significant amount of commented code in https://github.com/ProjectTwelve/contracts/blob/main/contracts/auctionHouse/AuctionHouseUpgradable.sol Why is that?

Recommendation

It's recommended that any unused code is removed from all contracts to minimize confusion and eliminate the risk of accidentally introducing bad code.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/auctionHouse/AuctionHouseUpgradable.sol

https://github.com/ProjectTwelve/contracts/blob/main/contracts/auctionHouse/ERC1155Delegate.sol#L78-L138

https://github.com/ProjectTwelve/contracts/blob/main/contracts/auctionHouse/ERC1155Delegate.sol#L53-L56

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/60c509935c39ec103349d8d57e05f2d3753c4f44

3.I Minor Remove unused function inputs

Description

The `itemHash` function input in the `_takePayment(itemHash)` function is unused.

Recommendation

Remove any unused function inputs.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/auctionHouse/AuctionHouseUpgradable.sol#L683

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/60c509935c39ec103349d8d57e05f2d3753c4f44

3.J Minor Incomplete inline documentation

Description

A lot of contracts are missing inline documentation for each of its function params. None of the functions have **@param** annotations. Some functions has no inline documentation at all.

Recommendation

Add inline @param documentation that explains each function argument in more detail.

// @dev blah blah

// @param arg1 what it is

function test(arg1, ...)

It is recommended that Solidity contracts and functions are fully annotated using NatSpec for all public interfaces (everything in the ABI). Ensure that the code is well commented both with NatSpec and inline comments for better readability and maintainability.

https://docs.soliditylang.org/en/v0.8.13/natspec-format.html

Code

All the contracts are missing inline documentation.

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/e7db54cd215228b7579be554e7fd8c9b89a7db95

3.K Major Potential re-entrancy exploit in AuctionHouse._run for ERC777 tokens

Description

External calls in `_run`:

- _takePayment: ERC20.safetransferFrom
- detail.executionDelegate.executeSell(order.user,shared.user,data)
- _distributeFeeAndProfit: ERC20.safeTransfer

Question: What `currency` will the auction house support? There is risk with supporting arbitrary currencies because **ERC777 tokens** support token transfer callbacks that may lead to re-entrancy or other exploits in the auction house.

Recommendation

For now: Make sure to add a whilelist for supported currencies, because there are some tokens that can introduce a re-entrancy issue.

Mark the `run()` function as `nonReentrant`

Code

Function: https://github.com/ProjectTwelve/contracts/blob/main/contracts/auctionHouse/AuctionHouseUpgradable.sol#L335-L339

Specific lines: https://github.com/ProjectTwelve/contracts/blob/main/contracts/auctionHouse/AuctionHouseUpgradable.sol#L378-L381

Status

Acknowledged, Addressed by currencies whitelist.

3.L Major Clarify use of ERC1155Delegate contract in order flow

Description

Question: Can users pick which ERC1155 `delegate` address they call? Consider the possibility that a hacker deploys a malicious delegate contract.

Question: Can users specify an arbitrary `SettleDetail.executionDelegate` address? Pointing to unknown, untrusted contracts can be very dangerous.

This validation for the `delegateType()` is not sufficient to prevent hackers from writing malicious delegate contracts.

Recommendation

There is already a whitelist check for the delegates addresses in the _run function. This addresses the issue.

Code

Status

Closed

3.M Minor There are open TODOs in the contracts

Description

There are open TODOs (linked below) in the contracts.

Recommendation

Please review if they should be resolved before the launch, or if they are planned to be future improvements. Any open TODOs should be implemented before an audit.

Code

https://github.com/ProjectTwelve/contracts/blob/56bc81f9ae0c031e7f5b99fbfbbf15a238e449e8/contracts/auctionHouse/ERC1155Delegate.sol#L63-L65

Status

Acknowledged, Fixed

3.N Medium Consider adding circuit breakers and emergency levers in the contracts

Description

What happens when one of the contracts stops working or suffers an exploit? Can we limit the blast radius of the damage by pausing certain operations to allow for a safe upgrade?

Recommendation

Consider inheriting from a `Pausable` contract to allow some critical operations to be paused during emergency scenarios and / or critical upgrades.

Code

Applies to all contracts.

Status

Acknowledged, Fixed

3.O Minor Check return value of ERC20 transfers

Description

Several tokens do not revert in case of failure and return false. If one of these tokens is used in P12V0FactoryUpgradeable, create will not revert if the transfer fails.

Recommendation

Use `SafeERC20`, or ensure that the **transfer/transferFrom** return value is checked.

You should use `safeTransfer`, `safeTransferFrom` instead.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L129
https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L205
https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L278

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/4ca67ed58ec124044da4c3db376e1ceb8c2f3a20

3.Q Medium Avoid using unaudited math libraries

Description

Has the `FullMath` library used by the economy contracts been audited? Be careful when using random code on the internet.

Answer: Yes, it's used by Uniswap https://github.com/Uniswap/solidity-lib/blob/master/contracts/libraries/FullMath.sol

Recommendation

Use audited contracts where possible: https://ethereum.stackexchange.com/questions/83785/what-fixed-or-float-point-math-libraries-are-available-in-solidity

Code

https://github.com/ProjectTwelve/contracts/blob/c4ebcd46ac7599db092633c3d77c8aa7517cb6d9/contracts/factory/P12V0FactoryUpgradeable.sol#L8

Status

Acknowledged

3.R Medium Follow the Checks-Effects-Interactions pattern to prevent re-entrancy

Description
Be aware that not following the pattern can lead to re-entrancy exploits. · Local state changes + validations · Events · External contracts last https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html
Recommendation
Defer external contract calls to after all local state changes. Events should also be emitted before external contract calls.
Code
https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L204-L205 https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol#L375 https://github.com/ProjectTwelve/contracts/blob/main/contracts/sft-factory/P12Asset.sol#L64-L65 Many other instances
Status
Acknowledged

3.S Minor Boolean constants can be compared directly (CLOSED)

Description
Boolean constants can be used directly and do not need to be compare to `true` or `false`
Recommendation

Remove the equality to the boolean constant.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L204-L205

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol#L375

https://github.com/ProjectTwelve/contracts/blob/main/contracts/sft-factory/P12Asset.sol#L64-L65

Many other instances

Status

Fixed

3.T Medium External calls in a loop can lead to a DoS

Description

Calls inside a loop might lead to a denial-of-service attack.

Recommendation

```
for () {
 <unknown untrusted contracts>.distribute(...)
 <unknown untrusted contracts>.distribute(...)
 <unknown untrusted contracts>.distribute(...)
}

// executeSell()
 for (uint256 i = 0; i < pairs.length; i++) {
    Pair memory p = pairs[i];
    p.token.safeTransferFrom(seller, buyer, p.tokenId, p.amount, new bytes(0)); // external contract acll
 }
```

If p.token is an ERC223 token

- safeTransfer(receiver)

- receiver.onTokenReceived(...) { ... any code }

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/secretShop/ERC1155Delegate.sol#L91-L94

Status

Acknowledged, Wontfix

3.U Medium Missing interface inheritance

Description

Contracts should inherit from its corresponding interfaces.

Recommendation

P12V0ERC20 should inherit from IP12Token

P12MineUpgradeable should inherit from IP12Mine

P12RewardVault should inherit from IP12RewardVault

Code

See above

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/d9a2b1dff738fccba7e4dd409fed864e2ff81879

3.V Medium OpenZeppelin's Reentrancy guard is more gas efficient than the custom lock implementation

Description

The contracts use a custom `lock()` modifier. However, in the implementation, setting a non-zero value and zero value repeatedly is expensive gas-wise. OZ's Reentrancy Guard uses strictly non-zero values so it's more gas efficient.

Recommendation

Use OpenZeppelin's ReentrancyGuard contract instead https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.6.0/contracts/security/ReentrancyGuard.sol

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L73-L78

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/ba33ddfa5ccf1b19ad14939b8c7ab68dfba64a24

3.W Minor Use helper functions to reduce code duplication

Description

Reduce code duplication by re-using internal and helper functions.

Recommendation

Call `getMintDelay()` in `_create`

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L206

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/7fa445dd2e8fc1b5ebbfae3dfb2a4e9a9f84a218

3.X Major Potential flashloan exploit (CLOSED)

Description

One of the contracts reads balances on an AMM pool.

This could be vulnerable to flashloan exploits.

This is called by an onlyP12Factory function, which is called by the `factory.create()` public function https://github.com/ProjectTwelve/contracts/blob/c4ebcd46ac7599db092633c3d77c8aa7517cb6d9/contracts/factory/P12V0FactoryUpgradeable.sol#L158

Could a developer exploit the pool balances with flashloans to get a favourable rewardDebt balance in the staking contract?

Recommendation

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol#L204-L211

Status

Acknowledged and Closed, the team has decided to implement a different Staking mining model altogether. The new staking model is based on Curve's Liquidity Mining model.

3.Z Major Blacklist deflationary and fee-on-transfer tokens as auction house currencies

Description

There is a group of non-compliant ERC20 tokens that will break some of the marketplace functionality.

These tokens behave in a way where if you send 100 tokens to A, A may receive < 100. This break the internal accounting of some contracts.

Here's an example exploit with fee-on-transfer / deflationary tokens https://defirate.com/balancer-sta-hack/ (on Balancer.) Their resolution to this was:

> To ensure this doesn’ t happen again, Balancer will be adding transfer fee tokens (like STA and STONK) to the UI blacklist similarly to what they have done for no bool transfer tokens.

Recommendation

Restrict which tokens can be used in the auction house with a whitelist.

It's also possible to update the contracts to handle these tokens, but it introduces a high gas overhead (you need to check token balances before and after the transfer each time) and there are only a few of these small tokens in existence.

Code

All AuctionHouse contracts that can support arbitrary tokens should use this whitelist so we don't mistakenly used a dangerous token contract.

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/85d207d2414034f33b335433536c2b169f1eea41

3.AA Medium Rename declareMintCoin to queueMint

Description

'Queue' is a clearer term to use for this Factory function because this term is widely used for governance contracts to queue proposals. (https://compound.finance/docs/governance#introduction)

Recommendation

Rename `declareMintCoin()` to `queueMint()`. Also rename the event.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L182

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L212

Status

Fixed in https://github.com/ProjectTwelve/contracts/pull/90/commits/08255a0218ef050b86f825188cb067a656b18e1d

3.AB Medium Document how different values of delayK and delayB affect mint delay time for users

Description

The mint delay is a value calculated from the following variables:

- A delayK config value
- A delayB config value
- The amount of game coin a developer wants to mint
- The total supply of the game coin

C++

```
/**
 * @dev linear function to calculate the delay time
 */
function getMintDelay(address gameCoinAddress, uint256 amountGameCoin) public view virtual override returns (uint256 time) {
    time = amountGameCoin.mul(delayK).div(P12V0ERC20(gameCoinAddress).totalSupply()) + 4 * delayB;
    return time;
}
```

The formula is simple, but we have to careful in choosing the config values to ensure the time is not too short and not too long.

Recommendation

Write documentation that explains the following:

- Graphs showing how different values of delayK and delayB affect the mint delay
- Example numbers showing how different values result in different mint delays
- What the initial values of delayK and delayB will be at launch.

Both users and auditors will need to know in more detail how these values are used in practice.

Open Question: Is there a minimum and maximum mintDelay value?

Open Question: Can a developer mistakenly queue an amount that can never be minted because the delay is so large?

Open Question: What is the '4' in the formula and can it not be part of delayB?

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L103-L109

Status

Fixed in https://github.com/ProjectTwelve/contracts/pull/105/commits/68b8bb525d323c028c3cc7d06d8abc698f83f520

3.AC Medium Rename withdrawDelay() to queueWithdraw()

Description

'Queue' is a clearer term to use for this P12Mine function because this term is widely used for governance contracts to queue proposals.

Recommendation

Rename `withdrawDelay()` to `queueWithdraw()`. Also rename the event.

To be consistent with the Factory contract, also rename `withdraw()` to `executeWithdraw()`.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol#L248

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol#L270

Status

Fixed in https://github.com/ProjectTwelve/contracts/pull/90/commits/08255a0218ef050b86f825188cb067a656b18e1d

3.AD Medium Add a getWithdrawDelay() view function to improve staking usability

Description

Before a user executes the `withdrawDelay()` function, they may want to know what the unlock timestamp will be for an amount of tokens.

Recommendation

Add a view function to show the unlock timestamp for a given withdrawal, something like:

```
Apache
1  getWithdrawUnlockTimestamp(address lpToken, uint256 amount)
```

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol#L264-L265

Status

Fixed in https://github.com/ProjectTwelve/contracts/pull/90/commits/5b1b68bcb4036676271f6288b21dc5c5e9dd4351

3.AE Medium Staking withdrawals requiring 2 transactions instead of 1 leads to bad usability for stakers

Description

`withdrawDelay()` -> wait -> `withdraw()`

The design of the P12Mint staking contract is not gas efficient because users have to make 2 transactions: one to declare their withdrawal and another to actually withdraw. This staking contract uses a similar model as the factory contract, but it doesn't make sense here. Most staking contracts only require 1 transaction for users to withdraw.

Recommendation

Consider having only a single `withdraw()` function (remove `withdrawDelay()`). If not enough time has passed since the user stakes the LP tokens, user withdrawals are hit by a 50% penalty. The penalty remains in the staking contract for the remaining stakers. This approach use the withdraw delay as a soft restriction instead of a hard restriction.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol

Status

Acknowledged, wontfix.

This is indeed the case, but this design is to protect the players and maintain the status quo for the time being

3.AF Medium Do you need a cancelMint() function in the Factory?

Description

Developers have to queue a mint before they can execute it. What happens if they queued the wrong amount? Is there a way to cancel it or will a developer have no choice but to execute that incorrect mint?

Recommendation

Consider adding a `cancelMint()` function that can be executed for queued, non-executed mints.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/sft-factory/P12AssetFactoryUpgradable.sol

Reply

We consider cancel function before, but don't add it finally. If developer can cancel mint freely, he can just declare every day and cancel it one minite before time up. When he really want to mint, nobody can expect it. We can compare entering wrong amount with the error of entering the wrong transfer destination address. On blockchain, everything should be dealed carefully.

Status

Acknowledged, Wontfix

3.AG Medium Order of functions in code should be: external, public, internal, then modifiers

Description

Functions should be sorted by visibility for better readability.

Recommendation

The standard convention is to order your functions and modifiers in this order:

- External
- Public
- Internal
- Modifiers

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/sft-factory/P12AssetFactoryUpgradable.sol

Other contracts

Status
Acknowledged, Fixed

3.AH Minor Unused function inputs

Description
The `netPrice` input in the `_distributeFeeAndProfit` function is redundant.
Recommendation
Remove redundant code unless it's required.
Code
https://github.com/ProjectTwelve/contracts/blob/main/contracts/secretShop/SecretShopUpgradable.sol#L369-L372 https://github.com/ProjectTwelve/contracts/blob/main/contracts/secretShop/SecretShopUpgradable.sol#L308
Status
Fixed

3.AI Medium RewardVault has no fund recovery method

Description
The RewardVault contract needs to be funded for the staking contract to work. What happens if we need to recover funds in it? (due to a bug or upgrades.) **Question: How is this contract funded? Be aware that the staking contract could revert if the reward vault has insufficient funds.** **Answer：After the staking contract is deployed, we will transfer a certain amount of P12token to the RewardVault contract** Monitor with **tenderly.co**
Recommendation
Add a way for the team to recover funds from the RewardVault during emergency scenarios.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12MineUpgradeable.sol#L417

https://github.com/ProjectTwelve/contracts/blob/main/contracts/staking/P12RewardVault.sol

Status

Fixed in https://github.com/ProjectTwelve/contracts/pull/90/commits/91aae450e7460755b4a63cbc9a55fcae6e748174

3.AJ Medium SafeMath is not needed in Solidity >0.8

Description

The SafeMath library validates if an arithmetic operation would result in an integer overflow/underflow. If it would, the library throws an exception, effectively reverting the transaction.

Since Solidity 0.8, the overflow/underflow check is implemented on the language level - it adds the validation to the bytecode during compilation.

Recommendation

Remove all unnecessary SafeMath imports.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/factory/P12V0FactoryUpgradeable.sol#L32

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/ac1d71c3fc74c4093ac14adb6c0e8f56f0b20718

3.AK Medium UUPS Initializer function can be frontrun

Description

Most contracts use an init pattern (instead of a constructor) to initialize contract parameters. Unless these are enforced to be atomic with contact deployment via deployment script or factory contracts, they are susceptible to front-running race conditions where an attacker/griefer can front-run (cannot access control because admin roles are not initialized) to initially with their own (malicious) parameters upon detecting (if

an event is emitted) which the contract deployer has to redeploy wasting gas and risking other transactions from interacting with the attacker-initialized contract.

Recommendation

Ensure atomic calls to init functions along with deployment via robust deployment scripts or factory contracts. Emit explicit events for initializations of contracts.

OR,

As per OpenZeppelin's (OZ) recommendation, "The guidelines are now to make it impossible for *anyone* to run `initialize` on an implementation contract, by adding an empty constructor with the `initializer` modifier. So the implementation contract gets initialized automatically upon deployment."

Note that this behaviour is also incorporated the OZ Wizard since the UUPS vulnerability discovery: "Additionally, we modified the code generated by the Wizard 19 to include a constructor that automatically initializes the implementation when deployed."

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/secretShop/SecretShopUpgradable.sol#L54

Status

Acknowledged

3.AL Medium Use the 2-step ownership transfer pattern

Description

Many contracts use a 1-step `Ownable` contract. The contract has a function that allows an admin to change it to a different address. If the admin accidentally uses an invalid address for which they do not have the private key, then the system gets locked.

Contract Ownable {

function transferOwnership(address newOwner) => owner = newOwner

}

Contract BoringOwnable {

```
    function transferOwnership(address newOwner) => pendingOwner = newOwner
    function acceptOwnership() onlyPendingOwner => owner = pendingOwner;
}
```

It is important to have two steps admin change where the first is announcing a pending new admin and the new address should then claim its ownership.

Recommendation

Use a 2-step Ownable contract instead, such as https://github.com/boringcrypto/BoringSolidity/blob/master/contracts/BoringOwnable.sol

Note that due to the use of UUPS, this change will require some new base contracts.

Code

https://github.com/ProjectTwelve/contracts/blob/main/contracts/secretShop/SecretShopUpgradable.sol#L9

Status

Fixed in https://github.com/ProjectTwelve/contracts/pull/86

3.BA Medium No way for contracts to upgrade to new VotingEscrow or GaugeController

Description

Consider the scenario when we need to upgrade `P12Mine` to point to a new voting escrow / gauge controller. Currently, we can't do that because there are no setters for these variables. Can we handle this scenario better?

Recommendation

Add a `setVotingEscrow` and `setGaugeController` functions.

Code

https://github.com/ProjectTwelve/contracts/pull/90

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/e5a2c01681e61c31f8a5efc5bf5f5326063e7676

3.BB Minor Consider using interface types instead of address

Description
Consider labelling `p12Mine` and `gaugeController` variables with their interface types `IP12Mine` and `IGaugeController` instead of a raw `address` type. It can functionally work the same, but referencing the interfaces makes the relationship between contracts more explicit. We are not expecting any contract address, but only contracts that implement a specific set of functions.
Recommendation
Use interface types.
Code
https://github.com/ProjectTwelve/contracts/pull/90
Status
Fixed in https://github.com/ProjectTwelve/contracts/commit/e5a2c01681e61c31f8a5efc5bf5f5326063e7676

3.BC Medium Consider adding an exit function for VotingEscrow

Description
For the VotingEscrow, what if we need to upgrade it or switch to a different model in the future? Currently, users may not be able to withdraw their locked tokens for N years (up to 4 years MAXTIME) to switch if we do. We may want to handle this better.
Recommendation
I recommend adding an 'exit' function that would unlock all locked tokens for emergency or upgrade purposes. Something like: ``` /** Ends the contract, unlocking all stakes No more locking can happen. Only withdraw */

```
 function expire() external override onlyOwner contractNotExpired {
  _expired = true;
  emit Expired();
 }
```

Code

https://github.com/ProjectTwelve/contracts/pull/90

Status

Fixed in https://github.com/ProjectTwelve/contracts/pull/90/commits/426ef8c4aef33c2f354458af849d0f0bc66278b5

3.BD Medium Use the 2-step transfer pattern for granting roles

Description

Currently the `grant*Role` functions (grantSuperAdminRoleToMultiSignWallet, grantDevRole) are very risky operations. Especially grantSuperAdminRoleToMultiSignWallet, because it renounces the role for the previous owner. If the wrong address was set, we could lose the super admin role altogether.

Recommendation

Use the 2-step transfer pattern similar to SafeOwnable.

Code

https://github.com/ProjectTwelve/contracts/pull/90

Status

Fixed in https://github.com/ProjectTwelve/contracts/commit/e5a2c01681e61c31f8a5efc5bf5f5326063e7676

Badge Merge contract https://github.com/ProjectTwelve/Badge-Merge-Contract

3.BE Medium Implement a view state function.

Description

For this code snippet https://github.com/ProjectTwelve/Badge-Merge-Contract/blob/main/contracts/BadgeMergeUpgradable.sol#L124-L131, consider defining a helper view function `canMerge()` that returns a boolean. The UI will probably need something like this.

Recommendation

Define a helper view function `canMerge()` that returns a boolean.

Code

https://github.com/ProjectTwelve/Badge-Merge-Contract/blob/main/contracts/BadgeMergeUpgradable.sol#L124-L131

Status

Added in https://github.com/ProjectTwelve/Badge-Merge-Contract/commit/de83396bf3b40ad357dd263dd0e4fddf3d19d831

3.BA Medium Use the burnBatch function to improve gas efficiency

Description

Token and Essence ERC721s implement a `burn()` and a `burnBatch` function. Can we use `burnBatch` instead?

Recommendation

Use the `burnBatch` function instead of looping and calling `burn`

Code

https://github.com/ProjectTwelve/Badge-Merge-Contract/blob/main/contracts/BadgeMergeUpgradable.sol#L128

Status

Fixed in https://github.com/ProjectTwelve/Badge-Merge-Contract/commit/de83396bf3b40ad357dd263dd0e4fddf3d19d831

3.BF Medium Restrict the depositBadge() function to only the owner

Description

For the depositBadge function, a user may accidentally call it without knowing what it does and lose their badge forever.

Recommendation

Consider making this function only callable by the contract owner.

Code

https://github.com/ProjectTwelve/Badge-Merge-Contract/blob/main/contracts/BadgeMergeUpgradable.sol#L133

Status

Fixed in https://github.com/ProjectTwelve/Badge-Merge-Contract/commit/de83396bf3b40ad357dd263dd0e4fddf3d19d831