Pop

DAO

Security Assessment

March 29th, 2021

Audited By:
Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org
Reviewed By:
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

CERTIK

Overview

Project Summary

Project Name	Pop - DAO
Description	Pop token, marketplace and liquidity pool smart contracts
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 5d3fdb27d19ae2f96a8d946665e423068b7343b1 2. a89a421783932bf14f7df5838164bc82079fa2d8

Audit Summary

Delivery Date	March 29th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 24th, 2021 - March 29th, 2021

Vulnerability Summary

Total Issues	55
● **Total Critical**	0
● **Total Major**	1
● **Total Medium**	0
● **Total Minor**	12
● **Total Informational**	42

Executive Summary

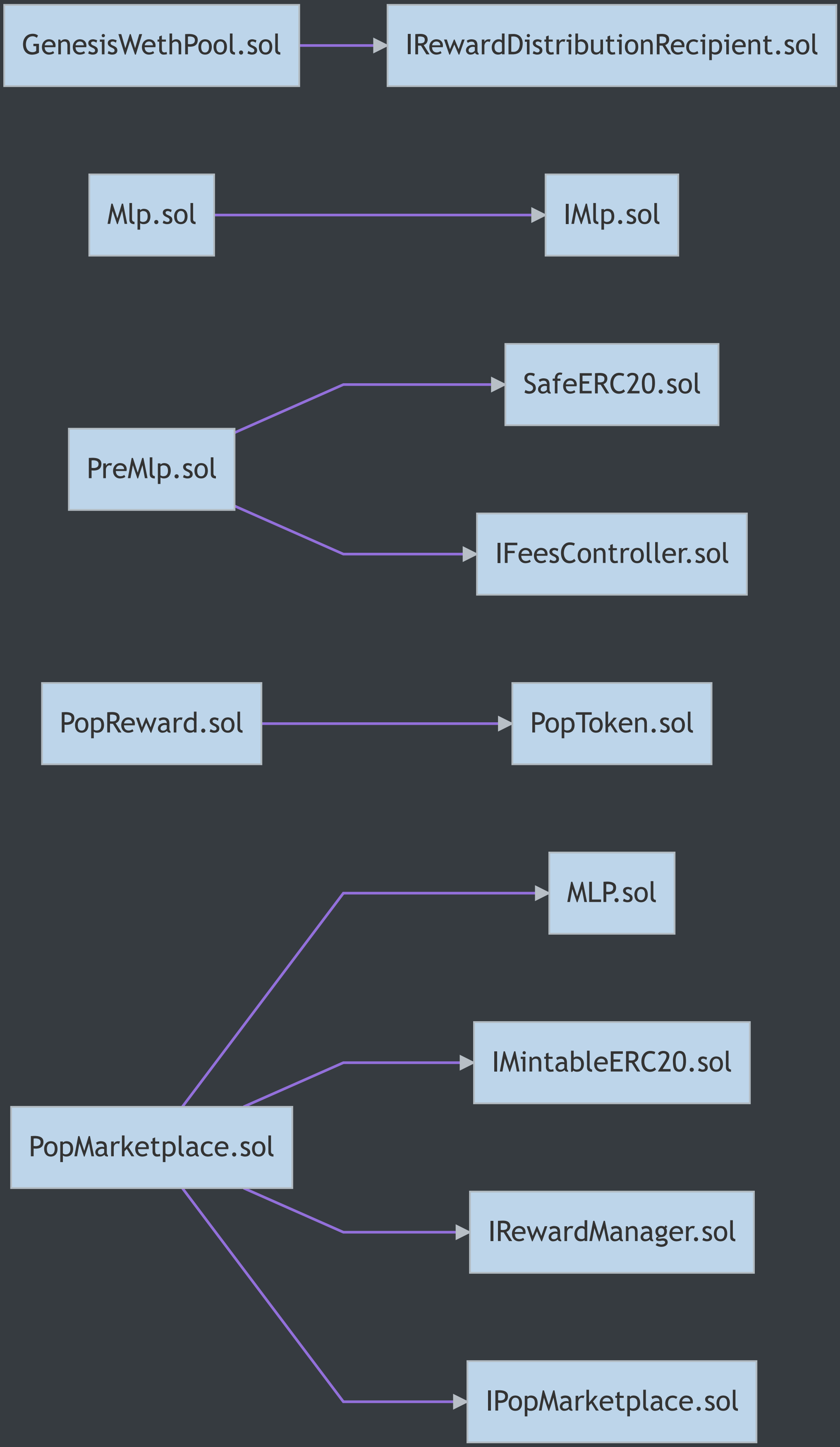
The report represents the results of our engagement with Pop on their implementation of Pop DAO related smart contracts.

Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract's safety.

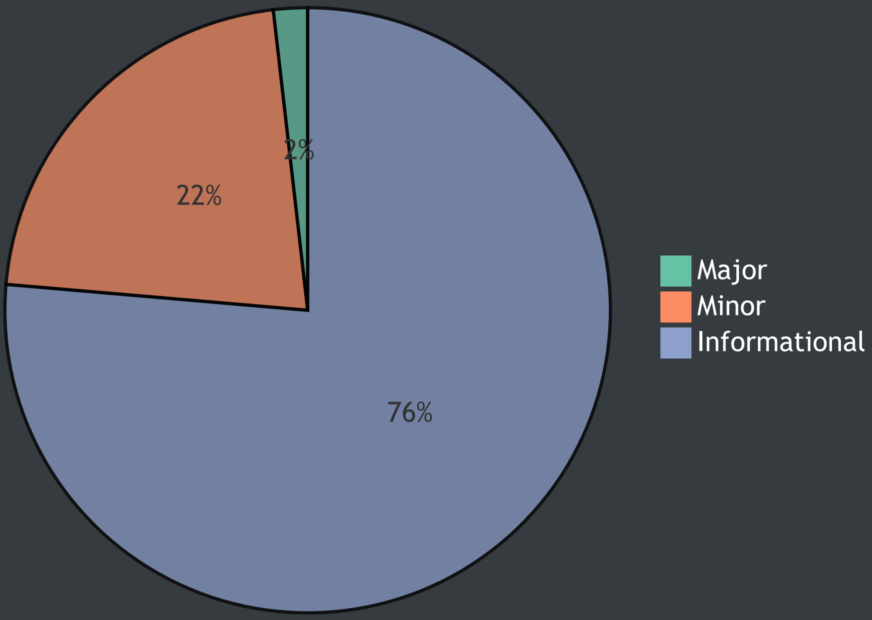
Files In Scope

ID	Contract	Location
DVG	DevVesting.sol	contracts/DevVesting.sol
GUP	GenesisUsdcPool.sol	contracts/GenesisUsdcPool.sol
GWP	GenesisWethPool.sol	contracts/GenesisWethPool.sol
MLP	Mlp.sol	contracts/Mlp.sol
PME	PopMarketplace.sol	contracts/PopMarketplace.sol
PRD	PopReward.sol	contracts/PopReward.sol
PTN	PopToken.sol	contracts/PopToken.sol
PMP	PreMlp.sol	contracts/PreMlp.sol

File Dependency Graph


GenesisWethPool.sol
IRewardDistributionRecipient.sol
Mlp.sol
IMlp.sol
SafeERC20.sol
PreMlp.sol
IFeesController.sol
PopReward.sol
PopToken.sol
MLP.sol
IMintableERC20.sol
PopMarketplace.sol
IRewardManager.sol
IPopMarketplace.sol

Finding Summary



Manual Review Findings

ID	Title	Type	Severity	Resolved
MLP-01M	Inexistent Input Sanitization	Volatile Code	● Minor	✓
MLP-02M	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	● Minor	✓
MLP-03M	`require` Over `assert` Statement	Volatile Code	● Minor	✓
MLP-04M	Visibility Specifiers Missing	Language Specific	● Informational	✓
MLP-05M	Redundant Statement	Gas Optimization	● Informational	✓
MLP-06M	`struct` Optimization	Gas Optimization	● Informational	✓
MLP-07M	Inconsistent Order of Layout	Inconsistency	● Informational	✓
MLP-08M	Lack of Error Message	Coding Style	● Informational	✓
MLP-09M	Statement Optimization	Gas Optimization	● Informational	✓
MLP-10M	Function Visibility Optimization	Gas Optimization	● Informational	✓
MLP-11M	Redundant `require` Statement	Gas Optimization	● Informational	✓
MLP-12M	Ambiguous `if` Block	Coding Style	● Informational	✓
MLP-	Code Optimization	Gas Optimization	● Informational	✓

13M				
PME-01M	`public` Setter Function	Volatile Code	🟠 Major	✓
PME-02M	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
PME-03M	`pair` Verification	Volatile Code	🔵 Minor	✓
PME-04M	Inconsistent Order of Layout	Inconsistency	🟢 Informational	✓
PME-05M	Visibility Specifiers Missing	Language Specific	🟢 Informational	✓
PME-06M	Redundant Variable Initialization	Coding Style	🟢 Informational	✓
PME-07M	`event` Optimization	Language Specific	🟢 Informational	✓
PME-08M	Lack of Error Message	Coding Style	🟢 Informational	✓
PME-09M	User-Defined Getters	Gas Optimization	🟢 Informational	✓
PME-10M	Inexistent Input Sanitization	Volatile Code	🟢 Informational	✓
PRD-01M	Empty Pop Marketplace	Volatile Code	🟢 Informational	✓
PTN-01M	`event` Optimization	Language Specific	🟢 Informational	✓
PTN-02M	Lack of Error Message	Coding Style	🟢 Informational	✓
PTN-	Inexistent Input	Volatile Code	🟢 Informational	✓

03M	Sanitization			
PMP-01M	Dust Tokens	Logical Issue	● Minor	✓
PMP-02M	`pair` Verification	Volatile Code	● Minor	✓
PMP-03M	Inconsistent Order of Layout	Inconsistency	● Informational	✓
PMP-04M	Redundant Statement	Gas Optimization	● Informational	✓
PMP-05M	Redundant Variable Initialization	Coding Style	● Informational	✓
PMP-06M	Visibility Specifiers Missing	Language Specific	● Informational	✓
PMP-07M	Lack of Error Message	Coding Style	● Informational	✓
PMP-08M	`event` Optimization	Language Specific	● Informational	✓
PMP-09M	Function Visibility Optimization	Gas Optimization	● Informational	✓

Static Analysis Findings

ID	Title	Type	Severity	Resolved
DVG-01S	Unlocked Compiler Version	Language Specific	● Informational	✓
GUP-01S	Unlocked Compiler Version	Language Specific	● Informational	✓
GWP-01S	Unlocked Compiler Version	Language Specific	● Informational	✓
MLP-01S	Potential Re-Entrancy	Volatile Code	● Minor	✓
MLP-02S	Unlocked Compiler Version	Language Specific	● Informational	✓
MLP-03S	Boolean Comparison	Gas Optimization	● Informational	✓
PME-01S	Potential Re-Entrancy	Volatile Code	● Minor	✓
PME-02S	Omitted Returned Value	Logical Issue	● Minor	✓
PME-03S	Unlocked Compiler Version	Language Specific	● Informational	✓
PME-04S	Incorrect `import` Statement	Compiler Error	● Informational	✓
PME-05S	Contract Size	Language Specific	● Informational	✓
PRD-01S	Potential Re-Entrancy	Volatile Code	● Minor	✓
PRD-02S	Unlocked Compiler Version	Language Specific	● Informational	✓
PRD-03S	Omitted Returned	Volatile Code	● Informational	✓

	Value			
PTN-01S	Unlocked Compiler Version	Language Specific	🟢 Informational	✓
PMP-01S	Potential Re-Entrancy	Volatile Code	🔵 Minor	✓
PMP-02S	Unlocked Compiler Version	Language Specific	🟢 Informational	✓
PMP-03S	Unused State Variable	Gas Optimization	🟢 Informational	✓
PMP-04S	Omitted Returned Value	Volatile Code	🟢 Informational	✓

MLP-01M: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	● Minor	Mlp.sol L79-L102

Description:

The constructor fails to check against non-zero values. This can lead to unexpected functionality, as the `PopMarketplace` contract creates `Mlp` instances without sanitization.

Recommendation:

We advise to add proper `require` statements, ensuring that any instance of the `Mlp` contract will not break the flow of the system.

Alleviation:

The development team acknowledged this exhibit but opted to keep the constructor function in its current version.

MLP-02M: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	Mlp.sol L236, L254, L433, L438

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

The development team opted to consider our references and utilized the `safeTransfer()` from the `SafeERC20.sol` library for the linked statements.

MLP-03M: `require` Over `assert` Statement

Type	Severity	Location
Volatile Code	● Minor	Mlp.sol L238, L256

Description:

In general, using `assert` is not the optimal, as a failed statement will consume the remaining gas.

Recommendation:

We advise to change the linked `assert` statement to `require` ones.

Alleviation:

The development team acknowledged this exhibit but opted to completely remove the linked `assert` statements.

MLP-04M: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	● Informational	Mlp.sol L20, L22, L62, L63, L76, L77

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references and added explicit visibility specifier to the linked state variables.

MLP-05M: Redundant Statement

Type	Severity	Location
Gas Optimization	● Informational	Mlp.sol L17

Description:

The linked statement is redundant, as the one in L16 ensures that the `SafeMath` library will be used for the `uint256` data type.

Recommendation:

We advise to remove redundant code.

Alleviation:

The development team opted to consider our references and removed the redundant code.

MLP-06M: `struct` Optimization

Type	Severity	Location
Gas Optimization	● Informational	Mlp.sol L49, L65

Description:

The `PendingOffer` and `ActiveOffer` structs can be further optimized, by striving for a 256-bit packing.

Recommendation:

We advise to change the linked `struct`s by grouping the boolean `struct` members along with the address ones, hence striving for a tight packing.

Alleviation:

The development team acknowledged this exhibit but opted to keep the linked `struct`s in their current version.

MLP-07M: Inconsistent Order of Layout

Type	Severity	Location
Inconsistency	● Informational	Mlp.sol General

Description:

The contract does not follow the Solidity conventions in regards to its structure.

Recommendation:

We advise to closely follow the Solidity style guide.

Alleviation:

The development team opted to consider our references and fixed the layout of the contract, closely following the Solidity conventions.

MLP-08M: Lack of Error Message

Type	Severity	Location
Coding Style	● Informational	Mlp.sol L181, L182, L183, L184, L219, L220, L336, L437

Description:

The linked `require` statements omit the error message string.

Recommendation:

We advise to add an error message to the linked statements.

Alleviation:

The development team acknowledged this exhibit but opted to keep the linked `require` statements in their current version.

MLP-09M: Statement Optimization

Type	Severity	Location
Gas Optimization	● Informational	Mlp.sol L96, L98, L100

Description:

The linked statements use state variables instead of the function parameters, hence increasing the gas consumption.

Recommendation:

We advise to use the local variables instead.

Alleviation:

The development team opted to consider our references and used the function parameters in the linked statements.

MLP-10M: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	● Informational	Mlp.sol L144, L170, L175, L304, L410

Description:

The linked functions are used for internal operations.

Recommendation:

We advise to change the visibility of the linked functions to `internal`.

Alleviation:

The development team opted to consider our references and changed the visibility of the `updateRewards()`, `_notifyDeposit()`, `_notifyWithdraw()`, `_provideLiquidity()` and `_getPriceVariation()` functions to `internal`.

MLP-11M: Redundant `require` Statement

Type	Severity	Location
Gas Optimization	● Informational	Mlp.sol L182, L430

Description:

The linked `require` statements are redundant, as the conditionals checked are being covered by either the subsequent `require` statements or by the subsequent `SafeMath.sub()` invocation.

Recommendation:

We advise to remove the redundant code.

Alleviation:

The development team opted to consider our references and removed the redundant code.

MLP-12M: Ambiguous `if` Block

Type	Severity	Location
Coding Style	● Informational	Mlp.sol L224

Description:

The linked `if` block is redundant, as it covers the opposite case of the previous `if` in L221

Recommendation:

We advise to change to an `else` block instead.

Alleviation:

The development team acknowledged this exhibit but opted to completely remove the two linked `if` blocks.

MLP-13M: Code Optimization

Type	Severity	Location
Gas Optimization	● Informational	Mlp.sol L240-L248, L258-L265

Description:

The linked code block can be moved outside of the nested `if-else` block, as in both cases, this code segment is executed.

Recommendation:

We advise to optimize the linked code segment as described.

Alleviation:

The development team acknowledged this exhibit but opted to completely remove the nested `if-else` blocks.

PME-01M: `public` Setter Function

Type	Severity	Location
Volatile Code	● Major	PopMarketplace.sol L130-L132, L138-L140

Description:

The linked `public` functions set the fees and the fee collector, hence can be manipulated by any user.

Recommendation:

We advise to add a control group that can invoke the linked functions.

Alleviation:

The development team opted to consider our references and added the `onlyOwner` modifier to the linked functions, ensuring that only the contract owner can invoke them.

PME-02M: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	PopMarketplace.sol L60, L63, L102, L103

Description:

The linked statements omit the returned value of the `transferFrom()` / `transfer()` calls. While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

The development team opted to consider our references and utilized the `safeTransfer()` and `safeTransferFrom()` from the `SafeERC20.sol` library for the linked statements.

PME-03M: `pair` Verification

Type	Severity	Location
Volatile Code	● Minor	PopMarketplace.sol L118, L151

Description:

The linked statements fail to directly check the existence of a token pair.

Recommendation:

We advise to add a `require` statement checking the address of the `pair` against the zero address.

Alleviation:

The development team acknowledged this exhibit but opted to keep the linked functions in their current version.

PME-04M: Inconsistent Order of Layout

Type	Severity	Location
Inconsistency	● Informational	PopMarketplace.sol General

Description:

The contract does not follow the Solidity conventions in regards to its structure.

Recommendation:

We advise to follow the Solidity style guide.

Alleviation:

The development team opted to consider our references and fixed the layout of the contract, closely following the Solidity conventions.

PME-05M: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	● Informational	PopMarketplace.sol L17, L21

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references and added explicit visibility specifier to the linked state variables.

PME-06M: Redundant Variable Initialization

Type	Severity	Location
Coding Style	● Informational	PopMarketplace.sol L20

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The development team opted to consider our references and removed the redundant code.

PME-07M: `event` Optimization

Type	Severity	Location
Language Specific	● Informational	PopMarketplace.sol L31

Description:

The `MlpCreated` event does not mark its `address` parameter with the `indexed` attribute.

Recommendation:

We advise to add the `indexed` attribute to the `address` parameter of the linked event.

Alleviation:

The development team opted to consider our references and added the `indexed` attribute to the `MlpCreated` event declaration.

PME-08M: Lack of Error Message

Type	Severity	Location
Coding Style	● Informational	PopMarketplace.sol L51, L54, L86, L114, L115, L145, L146

Description:

The linked `require` statements omit the error message string.

Recommendation:

We advise to add an error message to the linked statements.

Alleviation:

The development team opted to consider our references and added error messages to the linked `require` statements.

PME-09M: User-Defined Getters

Type	Severity	Location
Gas Optimization	● Informational	PopMarketplace.sol L126, L134

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The development team acknowledged this exhibit but opted to keep the user-defined getter functions, while also keeping the `private` visibility pecifiers for the respective state variables.

PME-10M: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	● Informational	PopMarketplace.sol L130-L132

Description:

The `setFeesTo()` function fails to check the value of the `_newFeesTo` parameter.

Recommendation:

We advise to add a `require` statement, checking the `_newFeesTo` parameter against the zero address.

Alleviation:

The development team opted to consider our references and added a `require` statement, checking the `_newFeesTo` parameter against the zero address.

PRD-01M: Empty Pop Marketplace

Type	Severity	Location
Volatile Code	● Informational	PopReward.sol L94-L96

Description:

The `setPopMarketplace()` function allows for an empty `popMarketplace`.

Recommendation:

We advise to add a `require` statement checking the input address against the zero address, if this is not an intended functionality.

Alleviation:

The development team opted to consider our references and added a `require` statement, checking the `_newMarketplace` parameter against the zero address.

PTN-01M: `event` Optimization

Type	Severity	Location
Language Specific	● Informational	PopToken.sol L13

Description:

The `MinterUpdate` event does not mark its `address` parameter with the `indexed` attribute.

Recommendation:

We advise to add the `indexed` attribute to the `address` parameter of the linked event.

Alleviation:

The development team opted to consider our references and added the `indexed` attribute to the `MinterUpdate` event declaration.

PTN-02M: Lack of Error Message

Type	Severity	Location
Coding Style	● Informational	PopToken.sol L20

Description:

The linked `require` statement omits the error message string.

Recommendation:

We advise to add an error message to the linked statement.

Alleviation:

The development team opted to consider our references and added an error message to the linked `require` statement.

PTN-03M: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	● Informational	PopToken.sol L34-L37

Description:

The `setMinter()` function fails to check the value of the `_account` parameter.

Recommendation:

We advise to add a `require` statement, checking the `_account` parameter against the zero address.

Alleviation:

The development team opted to consider our references and added a `require` statement, checking the `_account` parameter against the zero address.

PMP-01M: Dust Tokens

Type	Severity	Location
Logical Issue	● Minor	PreMlp.sol L185-L186

Description:

The linked token calculations will not transfer the remained of the tokens after the integer division.

Recommendation:

We advise to implement a function to collect the dust tokens.

Alleviation:

The development team opted to consider our references and added an additional transfer invocation of the remaining tokens to the contract owner.

PMP-02M: `pair` Verification

Type	Severity	Location
Volatile Code	● Minor	PreMlp.sol L177

Description:

The linked statement fails to check the existence of a token pair.

Recommendation:

We advise to add a `require` statement checking the address of the `pair` against the zero address.

Alleviation:

The development team acknowledged this exhibit but opted to keep the linked functions in their current version.

PMP-03M: Inconsistent Order of Layout

Type	Severity	Location
Inconsistency	● Informational	PreMlp.sol General

Description:

The contract does not follow the Solidity conventions in regards to its structure.

Recommendation:

e advise to closely follow the Solidity style guide.

Alleviation:

The development team opted to consider our references and fixed the layout of the contract, closely following the Solidity conventions.

PMP-04M: Redundant Statement

Type	Severity	Location
Gas Optimization	● Informational	PreMlp.sol L17

Description:

The linked statement is redundant, as the one in L16 ensures that the `SafeMath` library will be used for the `uint256` data type.

Recommendation:

We advise to remove redundant code.

Alleviation:

The development team opted to consider our references and removed the redundant code.

PMP-05M: Redundant Variable Initialization

Type	Severity	Location
Coding Style	● Informational	PreMlp.sol L21

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The development team acknowledged this exhibit but opted to remove the linked state variable from the contract.

PMP-06M: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	● Informational	PreMlp.sol L22, L23, L24

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references and added explicit visibility specifier to the linked state variables.

PMP-07M: Lack of Error Message

Type	Severity	Location
Coding Style	● Informational	PreMlp.sol L73, L100, L155, L156, L170-L173, L174

Description:

The linked `require` statements omit the error message string.

Recommendation:

We advise to add an error message to the linked statements.

Alleviation:

The development team opted to consider our references and added error messages to the linked `require` statements.

PMP-08M: `event` Optimization

Type	Severity	Location
Language Specific	● Informational	PreMlp.sol L38, L41

Description:

The linked events do not mark their `address` parameters with the `indexed` attribute.

Recommendation:

We advise to add the `indexed` attribute to the `address` parameters of the linked events.

Alleviation:

The development team opted to consider our references and added the `indexed` attribute to the `PreMlpCreated` and `PreMlpLiquidityReleased` event declarations.

PMP-09M: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	● Informational	PreMlp.sol L197

Description:

The `_provideLiquidity` function is used for internal operations.

Recommendation:

We advise to change the visibility of the linked function to `internal`.

Alleviation:

The development team opted to consider our references and changed the visibility of the `_provideLiquidity()` function to `internal`.

DVG-01S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	DevVesting.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

GUP-01S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	GenesisUsdcPool.sol L2

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

GWP-01S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	GenesisWethPool.sol L2

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

MLP-01S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	● Minor	Mlp.sol L180, L334, L344, L436

Description:

The linked functions update the state of the contract after external calls.

Recommendation:

We advise to apply the Checks-Effects-Interactions pattern.

Alleviation:

The development team opted to consider our references and applied the Checks-Effects-Interactions pattern to all but one exhibits.

MLP-02S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	Mlp.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

MLP-03S: Boolean Comparison

Type	Severity	Location
Gas Optimization	● Informational	Mlp.sol L349

Description:

The linked conditional redundantly compares two boolean values.

Recommendation:

We advise to directly use the `released` member of the `ActiveOffer` instance instead.

Alleviation:

The development team opted to consider our references and directly used the value of the `released` `struct` member.

PME-01S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	● Minor	PopMarketplace.sol L56-L64, L100-L103, L119, L152

Description:

The linked functions update the state of the contract after external calls.

Recommendation:

We advise to apply the Checks-Effects-Interactions pattern.

Alleviation:

The development team acknowledged this exhibit but opted to keep the linked functions in their current version.

PME-02S: Omitted Returned Value

Type	Severity	Location
Logical Issue	● Minor	PopMarketplace.sol L57, L119, L152

Description:

The linked statements omit the returned value of the `transferFrom()` / `transfer()` calls

Recommendation:

We advise that a `require` statement is added, ensuring the correct execution of the linked code.

Alleviation:

The development team opted to consider our references but utilized the `safeTransfer()` and `safeTransferFrom()` from the `SafeERC20.sol` library for the linked statements, after casting the tokens to the `IERC20` type.

PME-03S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	PopMarketplace.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

PME-04S: Incorrect `import` Statement

Type	Severity	Location
Compiler Error	● Informational	PopMarketplace.sol L8

Description:

The case conventions across platforms may not align, hence the linked `import` statement is generating a compilation error.

Recommendation:

We advise to change the name of the imported contract to the correct one.

Alleviation:

The development team opted to consider our references and fixed the linked `import` statement.

PME-05S: Contract Size

Type	Severity	Location
Language Specific	● Informational	PopMarketplace.sol General

Description:

Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet.

Recommendation:

We advise to remove redundant code.

Alleviation:

The development team acknowledged this exhibit.

PRD-01S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	● Minor	PopReward.sol L285

Description:

The linked functions update the state of the contract after external calls.

Recommendation:

We advise to apply the Checks-Effects-Interactions pattern.

Alleviation:

The development team opted to consider our references and moved the external call after the contract's state update.

PRD-02S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	PopReward.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

PRD-03S: Omitted Returned Value

Type	Severity	Location
Volatile Code	● Informational	PopReward.sol L285, L294

Description:

The linked statements omit the returned value of the `transferFrom()` / `transfer()` calls.

Recommendation:

We advise to add a `require` statement, ensuring the correct execution of the linked code.

Alleviation:

The development team opted to consider our references but utilized the `safeTransfer()` and `safeTransferFrom()` from the `SafeERC20.sol` library for the linked statements, after casting the tokens to the `IERC20` type.

PTN-01S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	PopToken.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

PMP-01S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	● Minor	PreMlp.sol L67, L93, L152, L167

Description:

The linked functions update the state of the contract after external calls.

Recommendation:

We advise to apply the Checks-Effects-Interactions pattern.

Alleviation:

The development team acknowledged this exhibit but opted to keep the linked functions in their current version.

PMP-02S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	PreMlp.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.6.6`.

PMP-03S: Unused State Variable

Type	Severity	Location
Gas Optimization	● Informational	PreMlp.sol L19, L20, L21

Description:

The linked state variables remain unused throughtout the codebase.

Recommendation:

We advise to remove redundant code.

Alleviation:

The development team opted to consider our references and removed the linked state variables from the contract.

PMP-04S: Omitted Returned Value

Type	Severity	Location
Volatile Code	● Informational	PreMlp.sol L133, L185, L186

Description:

The linked statements omit the returned value of the `approve()` / `transfer()` calls.

Recommendation:

We advise to add a `require` statement, ensuring the correct execution of the linked code.

Alleviation:

The development team opted to consider our references but utilized the `safeTransfer()` and `safeApprove()` from the `SafeERC20.sol` library for the linked statements, after casting the tokens to the `IERC20` type.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.