POLYTRADE

NFT Marketplace

SMART CONTRACT AUDIT REPORT

IMMUNEBYTES

29 JANUARY 2024

Introduction

About PolyTrade

Polytrade is building the first global marketplace focused exclusively on tokenized real-world assets ("RWA"). The RWA marketplace uses Polytrade's ERC-6960 to enhance discovery, consideration, investment, trade, fractionalization, and leverage for assets ranging from treasury bills, real estate, trade finance, and commodities, to physical items such as collectibles and luxury items. Users can use Polytrade as a single gateway to global tokenized opportunities.

Visit https://polytrade.finance/ to know more about it.

About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The team has provided the following doc for audit:

1. Internal Google Doc

Audit Goals

The audit focused on verifying that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include

    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Audit Process & Methodology

ImmuneBytes team have performed thorough testing of the project, starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safely used by third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

A team of independent auditors audited the code, including -

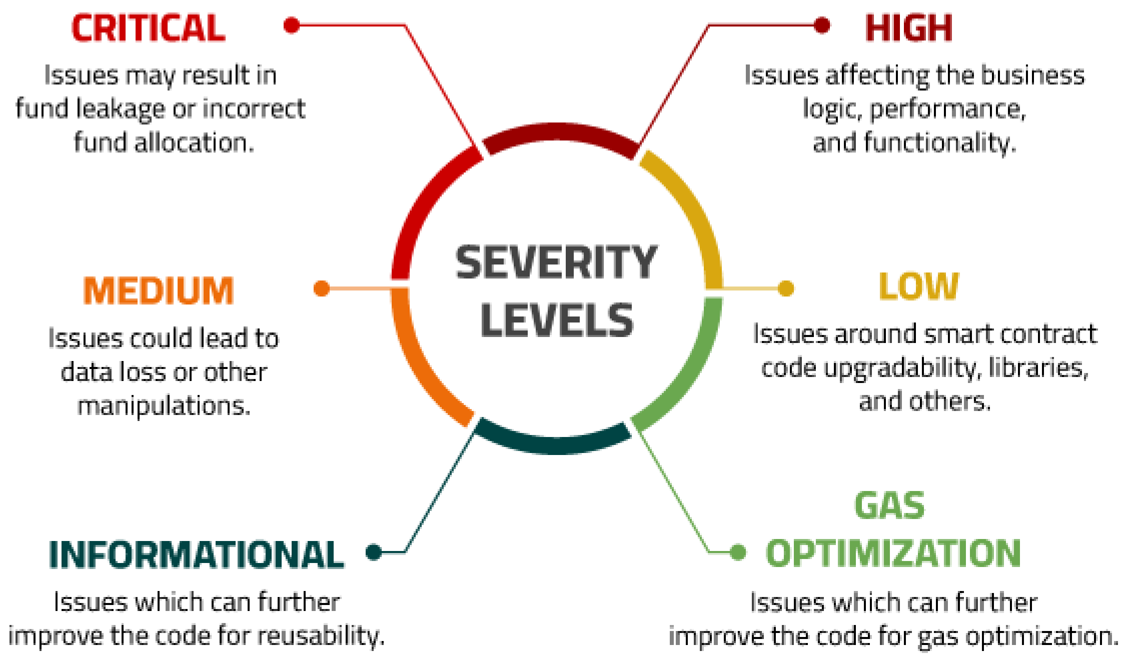
1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

Project Name	PolyTrade
Platform	Polygon
Languages	Solidity
GitHub Link	https://github.com/polytrade-finance/ntf-marketplace-smart-contract/tree/dev
Commit - Final Audit	68e0fc4d60ff65ef04eee7b50733c6b5c429c7fd
Platforms & Tools	Remix IDE, Truffle, VScode, Contract Library, Slither, SmartCheck, Fuzz

Security Level References

Every issue in this report was assigned a severity level from the following:


CRITICAL
Issues may result in fund leakage or incorrect fund allocation.
HIGH
Issues affecting the business logic, performance, and functionality.
SEVERITY LEVELS
MEDIUM
Issues could lead to data loss or other manipulations.
LOW
Issues around smart contract code upgradability, libraries, and others.
INFORMATIONAL
Issues which can further improve the code for reusability.
GAS OPTIMIZATION
Issues which can further improve the code for gas optimization.

Security Status References

OPEN

The issue hasn't been fixed as per our recommendation(s) and still stays pertinent.

FIXED

The issue has been agreed upon to be true and fixed as per our recommendation(s).

ACKNOWLEDGED

The issue is an intended behaviour and doesn't require a change as per our recommendation(s).

REDACTED

The reported issue is invalid, confirmed to be a false positive after communication with dev team.

Severity Status

Issues	Critical	High	Medium	Low	Informational
Open	-	-	-	-	-
Fixed	-	-	3	2	9
Redacted	-	-	-	-	-
Acknowledged	-	-	-	1	1

Finding

#	Findings	Risk	Status
01	InvoiceAsset.sol, PropertyAsset.sol: Missing input validations for imperative invoiceInfo/propertyInfo parameters during asset creation	**Medium**	**Fixed**
02	Risk of Gas Limit Exceedance in Public Batch Functions	**Medium**	**Fixed**
03	Marketplace.sol: Inadequate Validation in unlist Function and Potential Offer-Listing Mismatch	**Medium**	**Fixed**
04	InvoiceAsset.sol: _getAvailableRewards() includes redundant logic for tenure calculation of a given invoice	**Low**	**Fixed**
05	FeeManager.sol: Lack of Input Validations for Fee Settings in setDefaultFees	**Low**	**Fixed**
06	DLT.sol: Redundant require statements in functions using _checkOnDLTReceived	**Low**	**Acknowledged**
07	PropertyAsset.sol: Unused struct member PropertyInfo.price	**Informational**	**Intentional**
08	WrappedAsset.sol: Optimization of Storage Read in unWrap Functions _unwrapERC20	**Informational**	**Fixed**
09	WrappedAsset.sol: Unused Initializable Import in Immutable Contract	**Informational**	**Fixed**
10	Marketplace.sol: Enhancement of require Statement Conditions for Clarity and Correctness	**Informational**	**Fixed**
11	Marketplace.sol: Unclear Error Message in _list() function	**Informational**	**Fixed**
12	WrappedAsset.sol, PropertyAsset.sol: CHAIN_ID can be marked as immutable	**Informational**	**Fixed**
13	InvoiceAsset.sol: Non-Standard Definition of _YEAR in InvoiceAsset contract	**Informational**	**Fixed**
14	Consider shorter error messages in require statements	**Informational**	**Fixed**
15	PropertyAsset.sol: Incorrect local variable naming	**Informational**	**Fixed**
16	Natspec Annotations issues in contract	**Informational**	**Fixed**

Critical Severity Issues

No issues were found.

High Severity Issues

No issues were found.

Medium severity issues

1. **InvoiceAsset.sol, PropertyAsset.sol: Missing input validations for imperative invoiceInfo/propertyInfo parameters during asset creation**

   Note: The issue is relevant for both Invoice and Property asset functions (*mainly createInvoice & createProperty functions*). The example used below, however, is for the *_createInvoice* function from InvoiceAsset contract.

   The createInvoice function and its private counterpart _createInvoice in the smart contract are responsible for creating new invoices. However, the functions lack comprehensive input validation for the InvoiceInfo structure, potentially leading to unintended behaviors and vulnerabilities in the contract.

   During the review, it was found that only the settlementToken address within InvoiceInfo is validated for being a non-zero address. Other fields in InvoiceInfo such as price, dueDate, rewardApr, and fractions are not validated, allowing for any values, including zero, to be passed.

   Not including adequate validations during invoice creation itself, could lead to unwanted behavior in the contract.

   - For instance, If dueDate is set to an invalid value or a value less than the current block.timestamp, the invoice would be immediately eligible for settlement upon creation, which is likely not the desired behavior and could disrupt the invoice lifecycle and settlement processes.
   - Similarly, price and rewardApr are crucial for settlement and reward calculation. Inappropriate values for these fields could lead to incorrect financial calculations or zero rewards.

   Recommendation
   Include adequate input validation for all crucial parameters being passed during invoice creation in the contract.

   Status: Fixed

2. Risk of Gas Limit Exceedance in Public Batch Functions

Description

This issue is related to all contract that have batch functions that are publicly accessible.
For instance, we will consider the batchList() function of the Marketplace contract as an example for this issue.

The batchList function in the Marketplace contract allows for batch listing of assets, iterating over arrays of mainIds, subIds, and listedInfos. This function is marked as public, meaning it can be called by any external entity. The potential issue arises from the function's design, which allows for processing a potentially large number of listings in a single transaction.

As the number of listings increases, so does the gas required for each transaction. If the gas required for a transaction exceeds the block gas limit set by the Ethereum network, the transaction will fail, causing it to be reverted and not included in the blockchain.

Recommendations

The following patterns could be used to avoid any such instances of hitting the block gas limit:

1. Implement Batch Size Limitations:
   - Enforce a maximum batch size limit within the function to prevent transactions from exceeding the block gas limit. This limit should be determined based on typical gas usage patterns for listing operations.
   - Include a require statement that checks the length of the input arrays against this maximum batch size.

2. Gas Estimation and User Guidance:
   - Provide a mechanism (either on-chain or via off-chain tools) for users to estimate the gas cost of their batch listing operations before executing them.
   - Offer clear documentation or user interfaces that guide users on the optimal batch sizes to prevent failed transactions.

3. Restrict Access if possible:
   - For such functions, we can also consider making the function accessible only to specific roles or addresses if public access leads to frequent issues.

Status: Fixed

3. Marketplace.sol: Inadequate Validation in unlist Function and Potential Offer-Listing Mismatch

Line no: 388-342

Description:
The unlist function in the contract is designed to remove an asset's listing information.

However, it lacks any validation to check whether there is an active listing for the provided mainId and subId. Additionally, the current design of the contract supports the behavior where an asset owner can sign an offer for a listed asset but then unlist the asset, potentially causing the offer to fail.

Therefore, the current unlist() functions leads to 2 main issues:

Potential Offer-Listing Mismatch:

- The contract's offer function includes a mechanism for the user to verify asset owner's sign off on an offer, suggesting a commitment to the transaction and buy the asset.
- However, the owner can call unlist to remove the listing even after signing an offer, potentially invalidating the offer. This discrepancy could lead to failed transactions and a trust issue in the marketplace.
- Moreover, the offer function doesn't handle any such scenarios adequately and there isn't any error message to indicate such event to the user.

Missing Validation in unlist Function:

- The _unlist function, called by unlist, directly deletes the listing information without verifying if an active listing exists.
- This could result in redundant transactions and wasted gas if the function is called without an existing listing.

Recommendations

Unless the current design is intended, the _unlist and _offer functions should be redesigned to handle the above-mentioned scenario adequately.

Status: Fixed

Low severity issues

1. **InvoiceAsset.sol: _getAvailableRewards() includes redundant logic for tenure calculation of a given invoice**

   Line no: 441-445

   Description:
   As per the current design of the contract, the _claimRewards() function is only triggered before an invoice is settled, i.e., by the settleInvoice() function itself.

   The settleInvoice() function, however, has a strict condition that invoice can only be settled after the *dueDate* has passed. This indicates that rewards are also only claimable after the *dueDate* has passed.

   However, during the manual review, it was found that the _getAvailableRewards function performs a tenure calculation (*Line 441 to 445*) to check if the tenure is the dueDate or the current time. This calculation is redundant in the current context of the contract as tenure will always be the *dueDate* itself as it is enforced by the settleInvoice function.

   This redundant logic not only affects the gas consumption but also leads to a misleading logic and affects the readability of the code as it indicates that the rewards can be claimed even if due date is not passed while in reality that's not the case.

   Recommendation

   - If the intended behavior is to allow users to claim reward without relying strictly on due date, then the reward claiming function can be a stand-alone function which doesn't rely on invoice settlement rules.
   - Otherwise, the redundant logic can be removed to make the reward claiming functions simpler, gas efficient and readable.

   Status: Fixed

2. **FeeManager.sol: Lack of Input Validations for Fee Settings in setDefaultFees**
   Line no: 36 to 49

   Description
   The setDefaultFees function in the contract is responsible for setting default fees for initial and buying fees. However, the function lacks necessary input validations to ensure that the default fee values are appropriate, particularly considering that zero values could lead to unintended consequences in associated fee calculation functions and an overall risk in the fee management logic of the contract.

   Given that getInitialFee and getBuyingFee functions rely on these defaults when specific fee values are zero, setting a default fee to zero could effectively make all transactions free of charge, which may not be the intended behavior.

   Recommendations

   Include adequate checks to validate the arguments being passed as default fees.

   Status: Fixed

3. **DLT.sol: Redundant require statements in functions using _checkOnDLTReceived**
   Line no: 202, 243

   Description
   The DLT.sol contract includes a few functions to ensure that the recipient of tokens implements the DLTReceiver interface. These functions (e.g., *safeMint, safeTransfer*) uses the _checkOnDLTReceived function to perform this check. However, a redundancy exists in the way _checkOnDLTReceived's return value is used.

   Redundant Require Statement:

   - The _checkOnDLTReceived function already includes logic to revert the transaction if the recipient does not implement the expected interface.
   - The require statement in _safeMint, which checks the return value of _checkOnDLTReceived, is redundant because _checkOnDLTReceived will never return a false value – it either returns true or reverts.
   - Additionally, the redundancy leads to unnecessary gas consumption, as the EVM performs a check that is essentially guaranteed to pass.

   Recommendations
   Since _checkOnDLTReceived already handles the necessary reversion logic, the additional require statement in functions like _safeMint *or* _safeTransfer can be safely removed.

   Status: Acknowledged

Informational

1. **PropertyAsset.sol: Unused struct member PropertyInfo.price**

Description
The PropertyAsset.sol contract defines a PropertyInfo struct containing several members, including price. Upon review, it is observed that the price member is not utilized in any part of the PropertyAsset contract, leading to potential redundancy.

The _settleProperty function in the contracts allows passing an additional settlePrice argument instead of using the already existing price member in the PropertyInfo struct. This leads to redundant inclusion of two different members representing the same state in the contract which might lead to ambiguity.

Recommendation
In order to maintain the consistency of the contract design, it is recommended to use the price member directly from the PropertyInfo struct itself instead of introducing a new argument in the settlement function.

Alternatively, the unused struct member can be removed.

Status: Intentional

2. **WrappedAsset.sol: Optimization of Storage Read in unWrap Functions _unwrapERC20**
Line no: 463, 487

Description
The _unwrapERC20 function in the contract is designed for unwrapping ERC20 tokens. However, a review of the function reveals a redundancy in how it accesses storage, specifically within a require statement. This redundancy leads to unnecessary gas consumption.

The function reads from _wrappedInfo[mainId] to populate the info local variable and then again directly accesses _wrappedInfo[mainId] in the subsequent require statement.

Accessing storage variables multiple times increases gas costs, as storage reads are more expensive than memory reads in the Ethereum Virtual Machine (EVM).

Recommendation
Update the require statement to use the info variable itself.

Status: Fixed

3. **WrappedAsset.sol: Unused Initializable Import in Immutable Contract**

   Description
   The WrappedAsset contract imports the Initializable contract from OpenZeppelin's upgradeable contracts suite but does not utilize it.

   WrappedAsset is designed as an immutable contract, evidenced by the presence of a constructor which is executed once upon contract deployment. The contract is designed as an immutable contract with a constructor and therefore doesn't need the Initializable import.

   Recommendations
   Since the Initializable contract is not used, it should be removed from the import statements. This will clean up the code and clarify the contract's design as non-upgradeable.

   Status: Fixed

4. **Marketplace.sol: Enhancement of require Statement Conditions for Clarity and Correctness**
   Line no: 311, 316

   Description
   The _list function in the contract is crucial for listing assets. It includes several require statements for input validation.

   The _list function in the contract is crucial for listing assets. It includes several require statements for input validation. However, certain statements use != (not equal) to check against zero, where > (greater than) would be more appropriate and semantically clear.

   The use of != 0 in require statements, although functionally correct, may not adequately convey the intended condition, especially for quantities like listedInfo.minFraction and listedInfo.salePrice.

   Status: Fixed

5. **Marketplace.sol: Unclear Error Message in _list() function**
   Line no: 323

   Description
   The error message *"Fraction to list > Balance"* in one of the require statements of *_list()* function, may not be clear in scenarios where the asset is being listed without ownership. This could lead to confusion when diagnosing transaction failures.

   Status: Fixed

6. **WrappedAsset.sol, PropertyAsset.sol: CHAIN_ID can be marked as immutable**

   Description
   The state variable in WrappedAsset contract is only updated once in the constructor and there can be marked as immutable as per solidity best practices.

   Status: Fixed

7. **InvoiceAsset.sol: Non-Standard Definition of _YEAR in InvoiceAsset contract**
   Line no: 33

   Description
   The InvoiceAsset contract utilizes a custom definition of a year, set as 360 days instead of the standard 365 days. While this is intentional as per the client's specification, it deviates from the common understanding and could lead to confusion or miscalculations if not well-documented.

   Recommendation
   Clearly document the reasoning and implications of using a 360-day year in both internal code comments and external documentation accessible to users and developers.

   Status: Fixed

8. **Consider shorter error messages in require statements**

   Description
   Longer error messages in require statements contribute to increased bytecode size of the contract. In Ethereum, contracts with larger bytecode sizes consume more gas when deployed. This increases the deployment costs as well.

   Recommendations
   Error messages should be kept short and concise. Alternatively, adopting a standardized approach for error messages could also be helpful. For instance, using short codes or abbreviations that are documented and can be easily referenced.

   Status: Fixed

9. **PropertyAsset.sol: Incorrect local variable naming**
   Line: 148

   Description
   The burnProperty() function includes an incorrect local variable name at Line 148. It uses totalSubSupply as the variable while it actually fetches the data from _assetCollection.totalMainSupply().

   While it doesn't affect the intended behavior of the function, it affects its readability.

   Recommendation
   Use of clear and adequate local variable should be used in functions.

   Status: Fixed

10. **Natspec Annotations issues in contract**

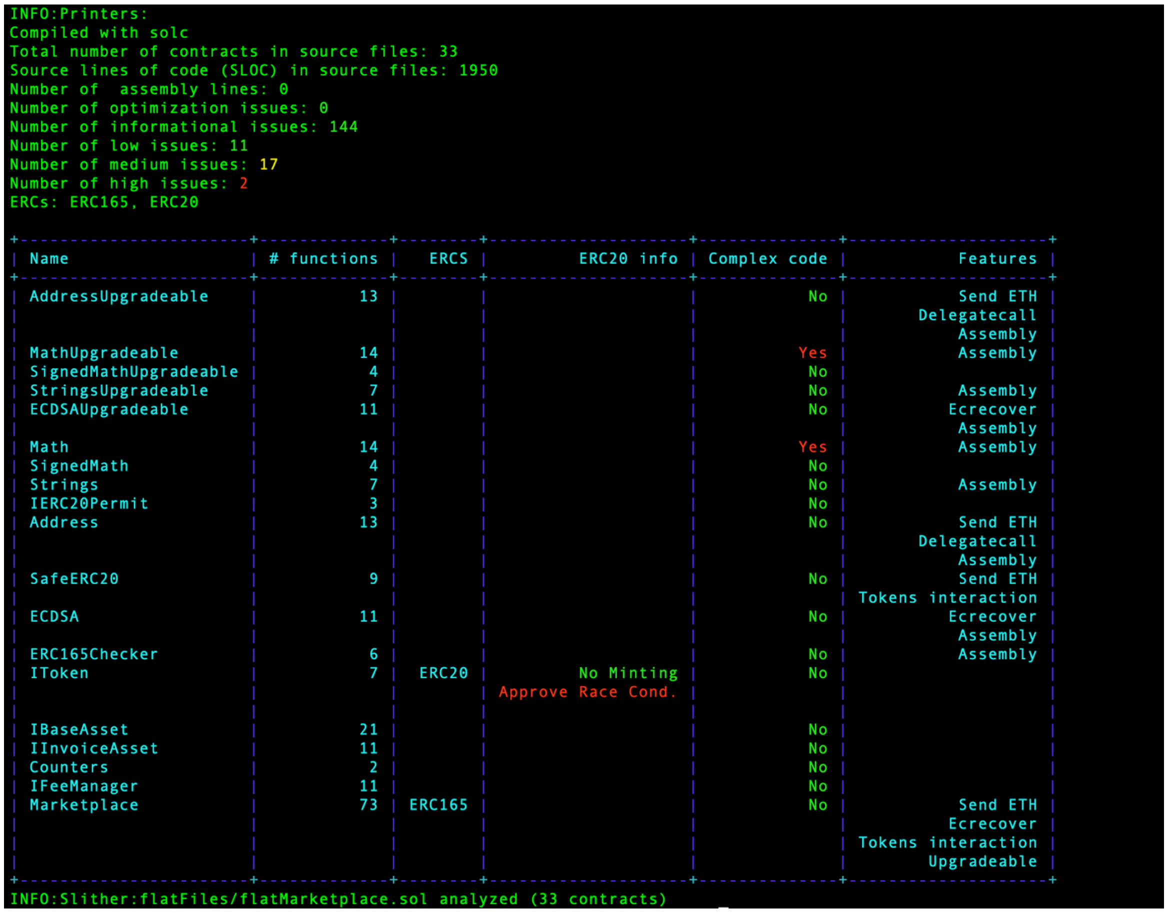
    Contracts: InvoiceAsset, WrappedAsset, Structs, IWrappedAsset

    Description
    Mentioned below are some of the issues related to Natspec Annotations of the provided contracts. These indicates inadequate or missing documentation of specific functions of the contracts and leads code readability issues as well.

    1. Invoice.sol: No natspect annotations found for function onSubIdCreation()
    2. Structs.sol-Line 22: Invalid comment
    3. WrappedAsset.sol-Line 79: Invalid tag for wrapERC20 function
    4. IWrappedAsset.sol: Missing natspec annotations for *wrapERC20, batchWrapERC20, emergencyUnwrapERC20*

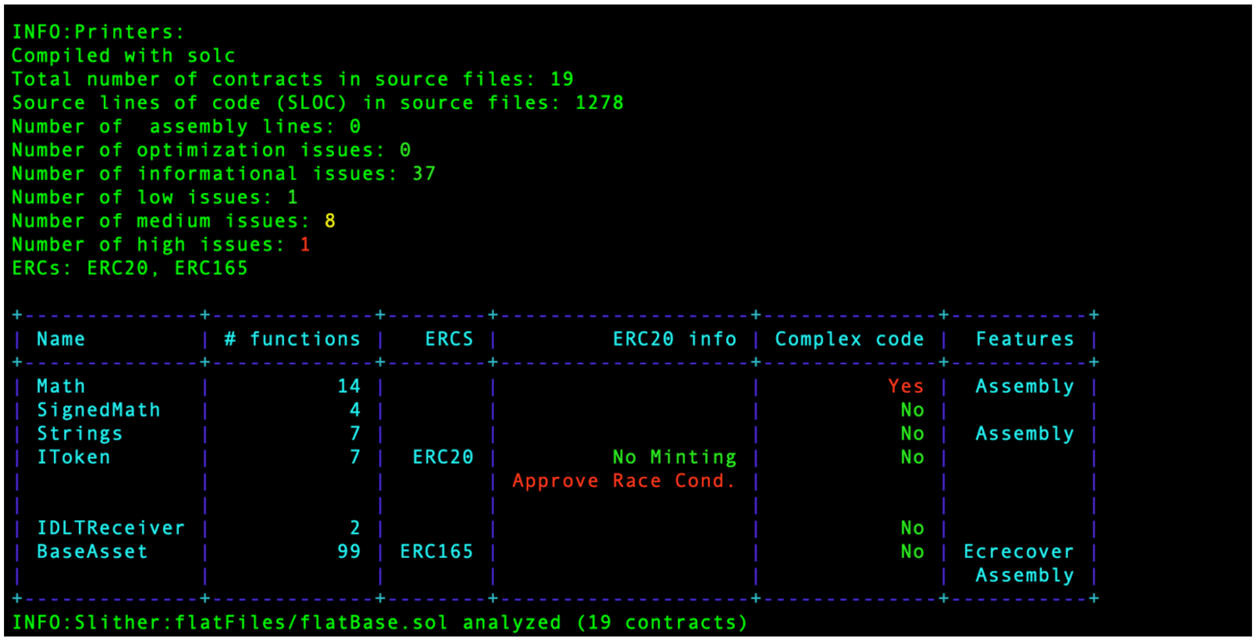
    Status: Fixed

Automated Test Results



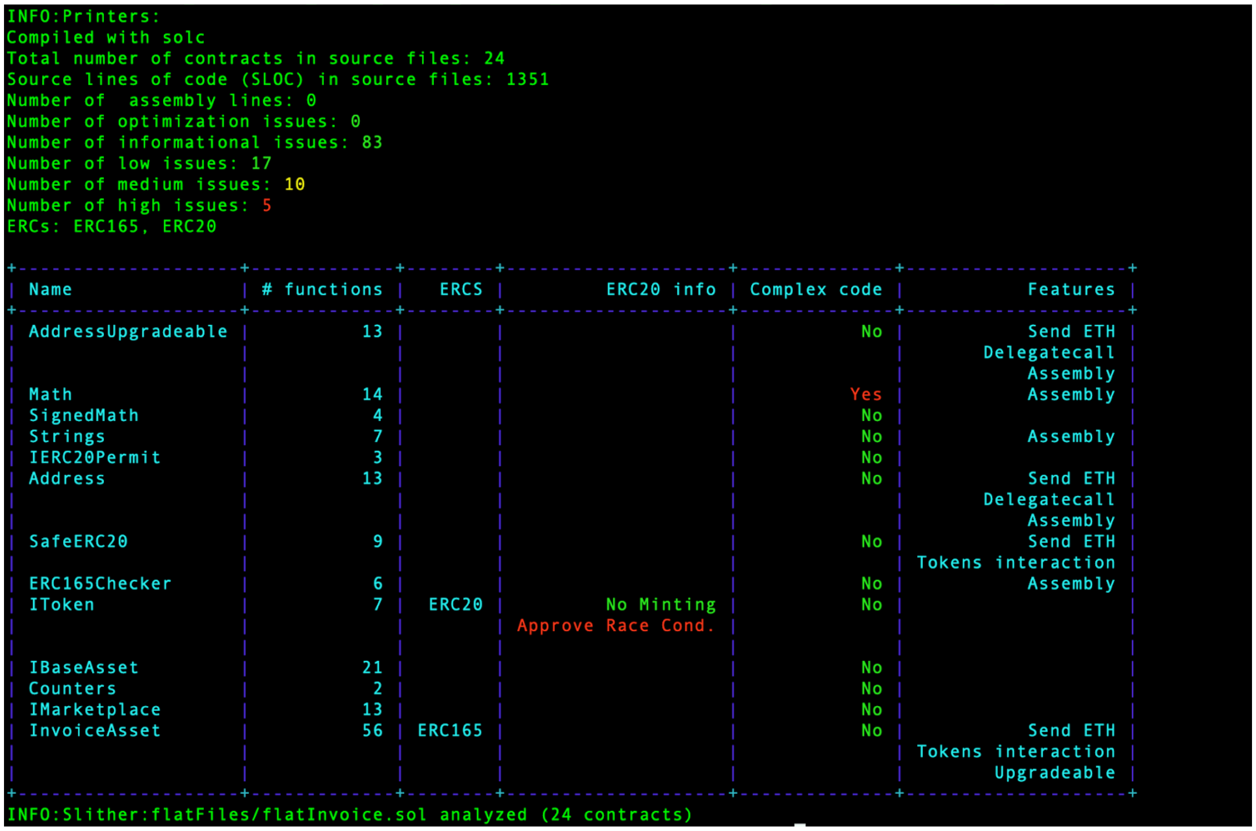
```
INFO:Printers:
Compiled with solc
Total number of contracts in source files: 33
Source lines of code (SLOC) in source files: 1950
Number of  assembly lines: 0
Number of optimization issues: 0
Number of informational issues: 144
Number of low issues: 11
Number of medium issues: 17
Number of high issues: 2
ERCs: ERC165, ERC20

+-----------------------+-------------+--------+--------------------+--------------+--------------------+
| Name                  | # functions |   ERCS |         ERC20 info | Complex code |           Features |
+-----------------------+-------------+--------+--------------------+--------------+--------------------+
| AddressUpgradeable    |          13 |        |                    |           No |           Send ETH |
|                       |             |        |                    |              |       Delegatecall |
|                       |             |        |                    |              |           Assembly |
| MathUpgradeable       |          14 |        |                    |          Yes |           Assembly |
| SignedMathUpgradeable |           4 |        |                    |           No |                    |
| StringsUpgradeable    |           7 |        |                    |           No |           Assembly |
| ECDSAUpgradeable      |          11 |        |                    |           No |          Ecrecover |
|                       |             |        |                    |              |           Assembly |
| Math                  |          14 |        |                    |          Yes |           Assembly |
| SignedMath            |           4 |        |                    |           No |                    |
| Strings               |           7 |        |                    |           No |           Assembly |
| IERC20Permit          |           3 |        |                    |           No |                    |
| Address               |          13 |        |                    |           No |           Send ETH |
|                       |             |        |                    |              |       Delegatecall |
|                       |             |        |                    |              |           Assembly |
| SafeERC20             |           9 |        |                    |           No |           Send ETH |
|                       |             |        |                    |              | Tokens interaction |
| ECDSA                 |          11 |        |                    |           No |          Ecrecover |
|                       |             |        |                    |              |           Assembly |
| ERC165Checker         |           6 |        |                    |           No |           Assembly |
| IToken                |           7 |  ERC20 |         No Minting |           No |                    |
|                       |             |        | Approve Race Cond. |              |                    |
|                       |             |        |                    |              |                    |
| IBaseAsset            |          21 |        |                    |           No |                    |
| IInvoiceAsset         |          11 |        |                    |           No |                    |
| Counters              |           2 |        |                    |           No |                    |
| IFeeManager           |          11 |        |                    |           No |                    |
| Marketplace           |          73 | ERC165 |                    |           No |           Send ETH |
|                       |             |        |                    |              |          Ecrecover |
|                       |             |        |                    |              | Tokens interaction |
|                       |             |        |                    |              |        Upgradeable |
+-----------------------+-------------+--------+--------------------+--------------+--------------------+
INFO:Slither:flatFiles/flatMarketplace.sol analyzed (33 contracts)
```



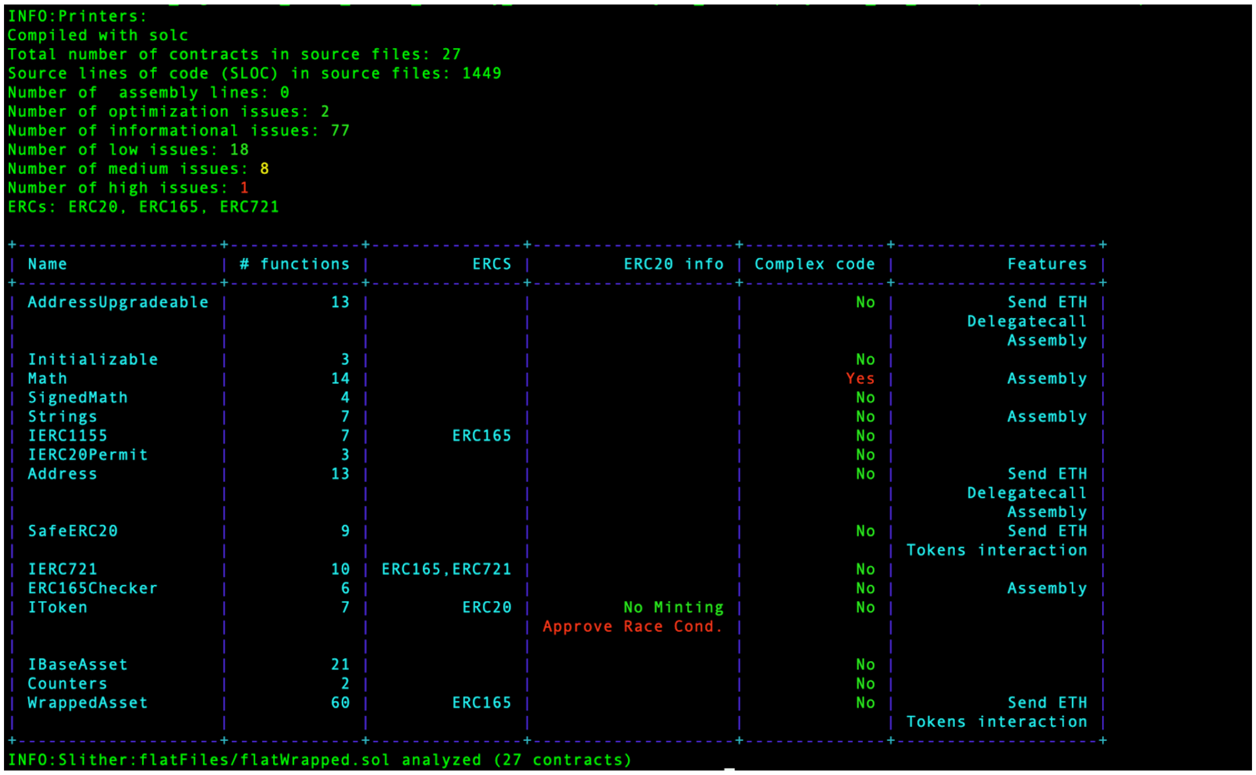
```
INFO:Printers:
Compiled with solc
Total number of contracts in source files: 19
Source lines of code (SLOC) in source files: 1278
Number of  assembly lines: 0
Number of optimization issues: 0
Number of informational issues: 37
Number of low issues: 1
Number of medium issues: 8
Number of high issues: 1
ERCs: ERC20, ERC165

+--------------+-------------+--------+----------------------+--------------+-----------+
| Name         | # functions |  ERCS  |           ERC20 info | Complex code |  Features |
+--------------+-------------+--------+----------------------+--------------+-----------+
| Math         |          14 |        |                      |          Yes |  Assembly |
| SignedMath   |           4 |        |                      |           No |           |
| Strings      |           7 |        |                      |           No |  Assembly |
| IToken       |           7 |  ERC20 |           No Minting |           No |           |
|              |             |        | Approve Race Cond.   |              |           |
|              |             |        |                      |              |           |
| IDLTReceiver |           2 |        |                      |           No |           |
| BaseAsset    |          99 | ERC165 |                      |           No | Ecrecover |
|              |             |        |                      |              |  Assembly |
+--------------+-------------+--------+----------------------+--------------+-----------+
INFO:Slither:flatFiles/flatBase.sol analyzed (19 contracts)
```



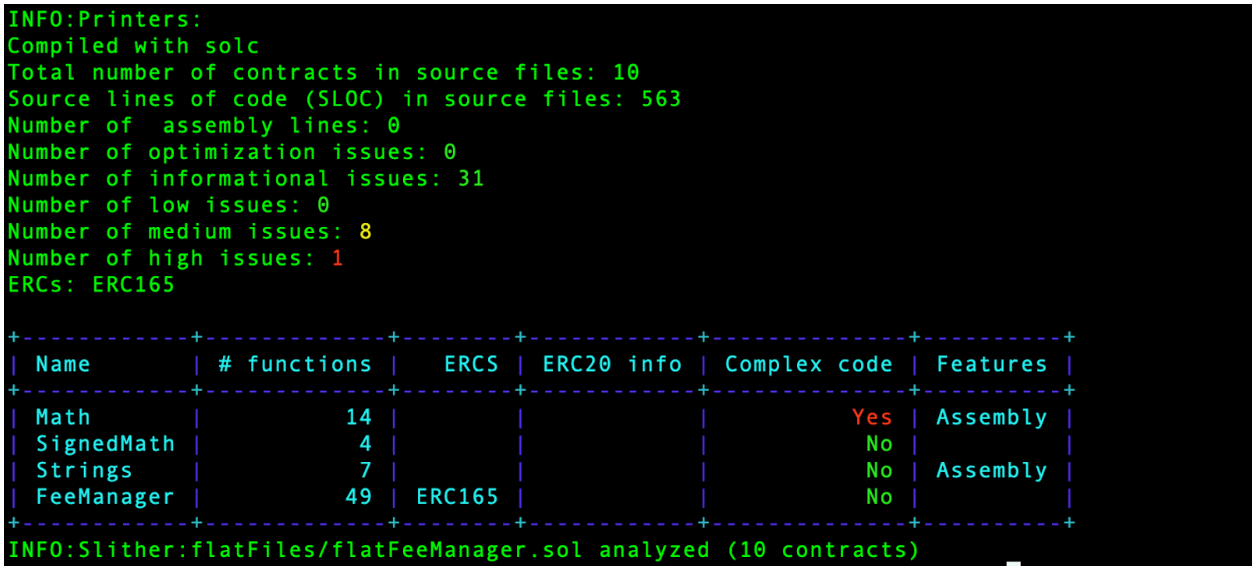
```
INFO:Printers:
Compiled with solc
Total number of contracts in source files: 24
Source lines of code (SLOC) in source files: 1351
Number of  assembly lines: 0
Number of optimization issues: 0
Number of informational issues: 83
Number of low issues: 17
Number of medium issues: 10
Number of high issues: 5
ERCs: ERC165, ERC20

+--------------------+-------------+--------+--------------------+--------------+--------------------+
| Name               | # functions |  ERCS  |         ERC20 info | Complex code |           Features |
+--------------------+-------------+--------+--------------------+--------------+--------------------+
| AddressUpgradeable |          13 |        |                    |           No |           Send ETH |
|                    |             |        |                    |              |       Delegatecall |
|                    |             |        |                    |              |           Assembly |
| Math               |          14 |        |                    |          Yes |           Assembly |
| SignedMath         |           4 |        |                    |           No |                    |
| Strings            |           7 |        |                    |           No |           Assembly |
| IERC20Permit       |           3 |        |                    |           No |                    |
| Address            |          13 |        |                    |           No |           Send ETH |
|                    |             |        |                    |              |       Delegatecall |
|                    |             |        |                    |              |           Assembly |
| SafeERC20          |           9 |        |                    |           No |           Send ETH |
|                    |             |        |                    |              | Tokens interaction |
| ERC165Checker      |           6 |        |                    |           No |           Assembly |
| IToken             |           7 |  ERC20 |         No Minting |           No |                    |
|                    |             |        | Approve Race Cond. |              |                    |
|                    |             |        |                    |              |                    |
| IBaseAsset         |          21 |        |                    |           No |                    |
| Counters           |           2 |        |                    |           No |                    |
| IMarketplace       |          13 |        |                    |           No |                    |
| InvoiceAsset       |          56 | ERC165 |                    |           No |           Send ETH |
|                    |             |        |                    |              | Tokens interaction |
|                    |             |        |                    |              |        Upgradeable |
+--------------------+-------------+--------+--------------------+--------------+--------------------+
INFO:Slither:flatFiles/flatInvoice.sol analyzed (24 contracts)
```



```
INFO:Printers:
Compiled with solc
Total number of contracts in source files: 27
Source lines of code (SLOC) in source files: 1449
Number of  assembly lines: 0
Number of optimization issues: 2
Number of informational issues: 77
Number of low issues: 18
Number of medium issues: 8
Number of high issues: 1
ERCs: ERC20, ERC165, ERC721

+--------------------+-------------+---------------+--------------------+--------------+--------------------+
| Name               | # functions |          ERCS |         ERC20 info | Complex code |           Features |
+--------------------+-------------+---------------+--------------------+--------------+--------------------+
| AddressUpgradeable |          13 |               |                    |           No |           Send ETH |
|                    |             |               |                    |              |       Delegatecall |
|                    |             |               |                    |              |           Assembly |
| Initializable      |           3 |               |                    |           No |                    |
| Math               |          14 |               |                    |          Yes |           Assembly |
| SignedMath         |           4 |               |                    |           No |                    |
| Strings            |           7 |               |                    |           No |           Assembly |
| IERC1155           |           7 |        ERC165 |                    |           No |                    |
| IERC20Permit       |           3 |               |                    |           No |                    |
| Address            |          13 |               |                    |           No |           Send ETH |
|                    |             |               |                    |              |       Delegatecall |
|                    |             |               |                    |              |           Assembly |
| SafeERC20          |           9 |               |                    |           No |           Send ETH |
|                    |             |               |                    |              | Tokens interaction |
| IERC721            |          10 | ERC165,ERC721 |                    |           No |                    |
| ERC165Checker      |           6 |               |                    |           No |           Assembly |
| IToken             |           7 |         ERC20 |         No Minting |           No |                    |
|                    |             |               | Approve Race Cond. |              |                    |
|                    |             |               |                    |              |                    |
| IBaseAsset         |          21 |               |                    |           No |                    |
| Counters           |           2 |               |                    |           No |                    |
| WrappedAsset       |          60 |        ERC165 |                    |           No |           Send ETH |
|                    |             |               |                    |              | Tokens interaction |
+--------------------+-------------+---------------+--------------------+--------------+--------------------+
INFO:Slither:flatFiles/flatWrapped.sol analyzed (27 contracts)
```



```
INFO:Printers:
Compiled with solc
Total number of contracts in source files: 10
Source lines of code (SLOC) in source files: 563
Number of  assembly lines: 0
Number of optimization issues: 0
Number of informational issues: 31
Number of low issues: 0
Number of medium issues: 8
Number of high issues: 1
ERCs: ERC165

+------------+-------------+--------+------------+--------------+----------+
| Name       | # functions |   ERCS | ERC20 info | Complex code | Features |
+------------+-------------+--------+------------+--------------+----------+
| Math       |          14 |        |            |          Yes | Assembly |
| SignedMath |           4 |        |            |           No |          |
| Strings    |           7 |        |            |           No | Assembly |
| FeeManager |          49 | ERC165 |            |           No |          |
+------------+-------------+--------+------------+--------------+----------+
INFO:Slither:flatFiles/flatFeeManager.sol analyzed (10 contracts)
```

Concluding Remarks

While conducting the audits of PolyTrade Finance, it was observed that the contracts contain Medium, and Low severity issues, along with a few recommendations.

Our auditors suggest that developers resolve Medium and Low-severity issues. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes' audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program complementing this audit is strongly recommended.

Our team does not endorse the PolyTrade Finance platform or its product, nor is this audit investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

IMMUNEBYTES

AUDITS