



AUDIT REPORT

June , 2025

For



Table of Content

Table of Content	02
Executive Summary	03
Number of Issues per Severity	05
Checked Vulnerabilities	06
Techniques & Methods	08
Types of Severity	10
Types of Issues	11
█ High Severity Issues	12
1. Contract upgradability is not optimal, upgrades could fail	12
█ Low Severity Issues	13
1. XFlyVoteModule contract has no voting mechanisms, only has staking and reward functionality	13
█ Informational Severity Issues	14
1. Unused public functions can be made external to save gas	14
2. Upgradeable contracts missing a gap variable increases storage collision risk in future upgrades/dependencies.	15
Functional Tests	16
Closing Summary & Disclaimer	17

Executive Summary

Project name	Magpie
Project URL	https://www.fly.trade
Overview	Magpie token contracts have a wrapping mechanism used to create rewards for stakers with a maximum vesting period of 90 days subject to penalties for early unstakers.
Audit Scope	https://github.com/magpieprotocol/magpie-token-contracts/tree/feat/add-token-contracts/contracts
Contracts in Scope	Fly.sol XFly.sol XFlyVoteModule.sol XFlyEmissionsDistributor.sol
Commit Hash	6742eeda8f032a3eb8d7e7843dea4eb9c2d4d992
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	12th May - 22th May, 2025
Updated Code Received	2nd June 2025
Review 2	2nd June 2025

Fixed In

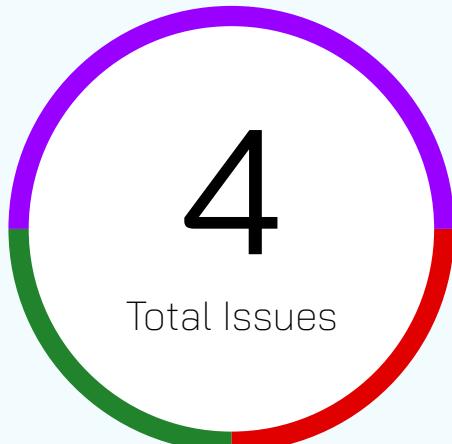
Fly.sol: f0b4507afe7cf74bca91f3971160dba42ca0571a

XFly.sol: 43beec3ec5e35569e8fd43833e8e7372e10caf6b

XFlyVoteModule.sol:cc22fd8121d261ac192bcfbff5b4336e6d712aaf

X Fly Emission Distributor:43beec3ec5e35569e8fd43833e8e7372e10caf6b

Number of Issues per Severity



High	1(25.00%)
Medium	0(0.00%)
Low	1(25.00%)
Informational	2(50.00%)

Issues	Severity			
	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	1	0	0	0
Acknowledged	0	0	1	2
Partially Resolved	0	0	0	0

Checked Vulnerabilities

<input checked="" type="checkbox"/> Access Management	<input checked="" type="checkbox"/> Compiler version not fixed
<input checked="" type="checkbox"/> Arbitrary write to storage	<input checked="" type="checkbox"/> Address hardcoded
<input checked="" type="checkbox"/> Centralization of control	<input checked="" type="checkbox"/> Divide before multiply
<input checked="" type="checkbox"/> Ether theft	<input checked="" type="checkbox"/> Integer overflow/underflow
<input checked="" type="checkbox"/> Improper or missing events	<input checked="" type="checkbox"/> ERC's conformance
<input checked="" type="checkbox"/> Logical issues and flaws	<input checked="" type="checkbox"/> Dangerous strict equalities
<input checked="" type="checkbox"/> Arithmetic Computations Correctness	<input checked="" type="checkbox"/> Tautology or contradiction
<input checked="" type="checkbox"/> Race conditions/front running	<input checked="" type="checkbox"/> Return values of low-level calls
<input checked="" type="checkbox"/> SWC Registry	<input checked="" type="checkbox"/> Missing Zero Address Validation
<input checked="" type="checkbox"/> Re-entrancy	<input checked="" type="checkbox"/> Private modifier
<input checked="" type="checkbox"/> Timestamp Dependence	<input checked="" type="checkbox"/> Revert/require functions
<input checked="" type="checkbox"/> Gas Limit and Loops	<input checked="" type="checkbox"/> Multiple Sends
<input checked="" type="checkbox"/> Exception Disorder	<input checked="" type="checkbox"/> Using suicide
<input checked="" type="checkbox"/> Gasless Send	<input checked="" type="checkbox"/> Using delegatecall
<input checked="" type="checkbox"/> Use of tx.origin	<input checked="" type="checkbox"/> Upgradeable safety
<input checked="" type="checkbox"/> Malicious libraries	<input checked="" type="checkbox"/> Using throw

Using inline assembly

Style guide violation

Unsafe type inference,Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

● High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

■ Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

● Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

■ Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open Security vulnerabilities identified that must be resolved and are currently unresolved.	Resolved Security vulnerabilities identified that must be resolved and are currently unresolved.
Acknowledged Vulnerabilities which have been acknowledged but are yet to be resolved.	Partially Resolved Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

High Severity Issues

Contract upgradability is not optimal, upgrades could fail

Resolved

Path

Fly.sol, XFly.sol, XFlyVoteModule.sol

Description

The contracts do not have a well-defined upgrade mechanism. The deploy scripts implement Open-Zeppelin's Foundry Upgrades library and directly calls `deployUUPSProxy` showing a preference for the UUPS proxy pattern. If the contracts are expected to comply with the UUPS proxy pattern, it would be beneficial to import the `UUPSUpgradeable` library directly in each contract and implement the critical calls necessary for a seamless upgrade.

```
1  contract DeployFly is Script {
2      function run() public {
3          address owner = makeAddr("owner");
4          address minter = makeAddr("alice");
5          vm.startBroadcast(owner);
6
7          address proxy = Upgrades.deployUUPSProxy(
8              "Fly.sol",
9              abi.encodeCall(Fly.initialize, (minter, owner))
10     );
}
```

Recommendation

- Properly inherit UUPS Upgradeable in the affected contracts.
- Call `__UUPSUpgradeable_init()`.
- Call `_authorizeUpgrade()` with an authorized account.

Low Severity Issues

XFlyVoteModule contract has no voting mechanisms, only has staking and reward functionality

Acknowledged

Path

XFlyVoteModule.sol

Description

Users will expect the XFlyVoteModule to contain some voting mechanism that leverages on their staked tokens. The delegate function puts another user on an allowlist that is currently unused. As of now, the vote module contract does not provide users with voting abilities leaving them with no voting power and as such can be renamed.

Recommendation

- Implement voting mechanism in current or future upgrades.
- Ensure this contract is forward-compatible if upgrades will be built atop this one, there should be storage slots properly defined, and needed functions implemented for use.
- Consider inheriting ERC20VotesUpgradeable for a similar voting system structure.

Informational Severity Issues

Unused public functions can be made external to save gas

Acknowledged

Path

XFly.sol, XFlyVoteModule.sol

Function

getBalanceResiding(), usersTotalVests(), getVestInfo(), isWhitelistedFrom(), isWhitelistedTo(), left()

Description

There are several public functions in the codebase only called externally that can be optimized by changing their visibility to external. This change would reduce gas costs and better reflect the intended usage of these functions.

Recommendation

Change the visibility modifier from public to external for all affected functions

Upgradeable contracts missing a gap variable increases storage collision risk in future upgrades/dependencies.

Acknowledged

Path

Fly.sol, XFly.sol, XFlyVoteModule.sol

Description

Storage gaps allow for adding new storage variables in future upgrades, to prevent storage collisions between parent and child contracts. The contracts listed above inherit from multiple upgradeable contracts but do not implement a `__gap` which means future upgrades could cause storage collisions.

Recommendation

Consider including a storage gap `x` that sums up the total number of storage slots used in the contract to 50 slots as is customary.

```
uint256[x] private __gap;
```

Magpie's Team comment :

We are using Open Zeppelin 5, which doesn't use gaps anymore. it uses ERC-7201 instead. The main contract shouldn't have collision until we keep the original variables to preserve the right slots. If there is still a concern that there could be an issue, we can use ERC-7201 also in the main contracts (Fly, XFly, etc...) if necessary. If we use ERC-7201 in the upgrades only, it should still avoid collision.

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should mint Fly only to the minter address
- ✓ Should convert Fly to XFly
- ✓ Should compulsorily pay penalties upon exit before MAX_VESTING time is reached
- ✓ Should refund XFly and Fly tokens depending on position in the vesting schedule
- ✓ Should claim rewards within 2-week window
- ✓ Should allow owner withdraw only after 2-week window passes
- ✓ Should distribute rewards
- ✓ Should emit events for state-changing functions
- ✓ Should be resistant to block-stuffing attacks

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Magpie. We performed our audit according to the procedure described above.

1 High, 1 Low and 2 Informational issues were found. One of the issue was resolved and others were acknowledged

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

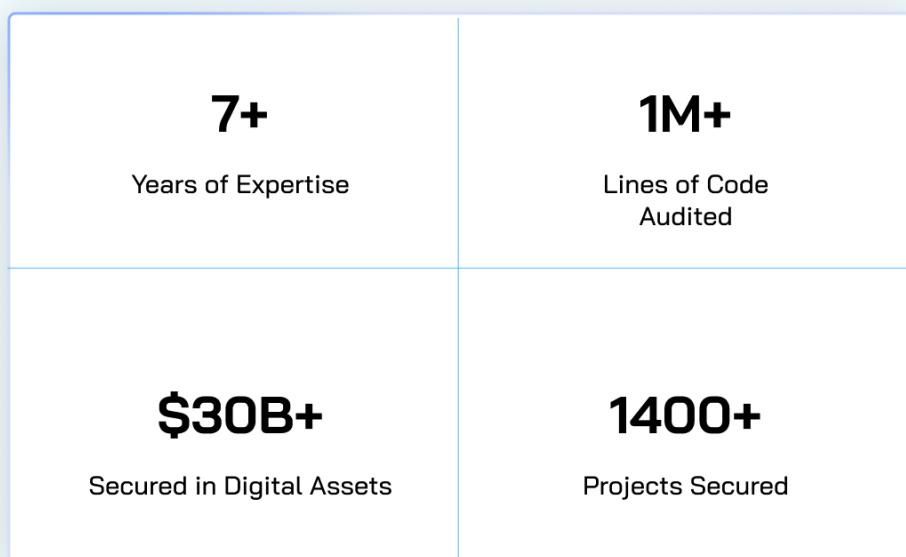
While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.



About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem



Follow Our Journey



AUDIT REPORT

June , 2025

For



Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com