



Audit Report November, 2021

For



Amplify

<https://ampt.finance>

Contents

Scope of Audit	01
Check Vulnerabilities	01
Techniques and Methods	02
Issue Categories	03
Number of security issues per severity.	03
Introduction	04
A. Contract - Amplify Token	05
High Severity Issues	05
Medium Severity Issues	05
Low Severity Issues	05
Informational Issues	05
Functional Tests	06
Automated Tests	07
Closing Summary	09

Scope of the Audit

The scope of this audit was to analyze and document the Amplify Token Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token 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 analysed 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

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 analysed, 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 behaviour 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, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

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.

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

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	0
Closed	0	0	0	0

Introduction

During the period of **October 18, 2021 to October 26, 2021** - QuillAudits Team performed a security audit for **Amplify Token** smart contracts.

The code for the audit was taken from following the official link:

Note	Date	Commit hash
Version 1	October 26	https://github.com/amplify-labs/contracts/blob/46076504ecbee10de72df4880684d40460fceb33/protocol/contracts/AMPT/AMPT.sol

Issues Found

A. Contract – Amplify Token

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Informational issues

No issues were found.



Functional test

Function Names	Testing results
allowance	Passed
approve	Passed
balanceOf	Passed
transfer	Passed
transferFrom	Passed
delegate	Passed
delegateBySig	Passed
getCurrentVotes	Passed

Automated Tests

Slither

```
INFO:Detectors:
AMPT._writeCheckpoint(address,uint32,uint96,uint96) (AMPT.sol#276-287) uses a dangerous strict equality:
  - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (AMPT.sol#279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
AMPT.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (AMPT.sol#175-184) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(now <= expiry,AMPT::delegateBySig: signature expired) (AMPT.sol#182)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
AMPT.getChainId() (AMPT.sol#310-314) uses assembly
  - INLINE ASM (AMPT.sol#312)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Constant AMPT.totalSupply (AMPT.sol#29) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
AMPT.slitherConstructorConstantVariables() (AMPT.sol#18-316) uses literals with too many digits:
  - totalSupply = 1000000000e18 (AMPT.sol#29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
delegate(address) should be declared external:
  - AMPT.delegate(address) (AMPT.sol#162-164)
delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
  - AMPT.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (AMPT.sol#175-184)
getPriorVotes(address,uint256) should be declared external:
  - AMPT.getPriorVotes(address,uint256) (AMPT.sol#203-235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Mythril

```
enderphan@enderphan AMPT % myth a AMPT.sol
The analysis was completed successfully. No issues were detected.
```

SOLHINT LINTER

```
AMPT.sol
14:1  error    Compiler version ^0.6.12 does not satisfy the ^0.5.8 semver requirement  compiler-version
20:5  warning  Constant name must be in capitalized SNAKE_CASE                        const-name-snakecase
23:5  warning  Constant name must be in capitalized SNAKE_CASE                        const-name-snakecase
26:5  warning  Constant name must be in capitalized SNAKE_CASE                        const-name-snakecase
29:5  warning  Constant name must be in capitalized SNAKE_CASE                        const-name-snakecase
180:9 warning  Error message for require is too long                                reason-string
181:9 warning  Error message for require is too long                                reason-string
182:9 warning  Error message for require is too long                                reason-string
182:17 warning  Avoid to make time-based decisions in your business logic            not-rely-on-time
204:9 warning  Error message for require is too long                                reason-string
248:9 warning  Error message for require is too long                                reason-string
249:9 warning  Error message for require is too long                                reason-string
312:9 warning  Avoid to use inline assembly. It is acceptable only in rare cases    no-inline-assembly
```

Results

No 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 the Amplify Token platform. We performed our audit according to the procedure described above.

The audit showed no issues.



Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Amplify Token** platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **Amplify Token** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report November, 2021

For



Amplify

<https://ampt.finance>



QuillAudits

📍 Canada, India, Singapore, United Kingdom

🌐 audits.quillhash.com

✉️ audits@quillhash.com