# Amp

## Security Assessment

**August 3, 2020**


Prepared For:
Dave McGregor | *Flexa*
dave@flexa.co

Zachary Kilgore | *Flexa*
z@flexa.co

Chris Pick | *Flexa*
cpick@flexa.co

Trevor Filter | *Flexa*
trev@flexa.co

Prepared By:
Robert Tonic | *Trail of Bits*
robert.tonic@trailofbits.com

Michael Colburn | *Trail of Bits*
michael.colburn@trailofbits.com

# Executive Summary

From July 13 through August 3, 2020, Flexa engaged Trail of Bits to review the security of their Solidity smart contracts, off-chain backend services, and Google Cloud infrastructure configuration. Trail of Bits conducted this assessment over the course of six person-weeks with two engineers, reporting 18 issues.

This report details only the findings relevant for the smart contract components. The smart contracts were reviewed over the duration of the larger assessment by one engineer working from the commit `aece0f6` in the `amptoken/amp-token-contracts` repository and `8d421c2` from the `flexahq/flexa-collateral-manager` repository.

**Week one:** We performed a mix of manual and automated review. We used Slither to evaluate the Solidity smart contracts. We then manually reviewed the analysis of the smart contracts, alongside repository familiarization activities. These efforts led to one finding, TOB-FLXA-001: Duplicate contract names may lead to errors.

**Week two:** We continued our manual review, focusing on the `FlexaCollateralManager` contract and its interactions with the `Amp` token contract. Additionally, we identified areas of the `Amp` contract that would make good candidates for property-based fuzzing with Echidna.

**Week three:** We completed our manual review of the collateral manager contract and its interactions with the Amp token. We also developed property tests for Echidna to fuzz several of the token operations in the Amp token (see Appendix D). We have also included an appendix detailing code quality recommendations (Appendix C).

**Assessment results:** Overall, we identified one low-severity finding in the smart contracts where duplicate contract names found throughout the system may have adverse effects for third-party integrations due to the production of incomplete build artifacts.

**Next steps:** Flexa should fix the duplicated Amp interface used in several contracts to facilitate easier integration with third-party tooling. Additionally, although only minimal issues were identified in the contract code, due to its complexity it would benefit from additional high-level documentation that describes various interactions between the contracts, the expected uses of partitions and planned strategies for them, and which aspects of various standards the contracts do and do not implement.

# Project Dashboard

### Application Summary

| | |
|---|---|
| Name | Amp Solidity contracts |
| Version | `amp-token-contracts` aece0f6b24df6348221da548a815528a6633a20e<br>`flexa-collateral-manager` 8d421c295c2ed5d3eef12e5992d96efb8d10d2d3 |
| Type | Solidity |
| Platforms | Ethereum |

### Engagement Summary

| | |
|---|---|
| Dates | July 13–August 3, 2020 |
| Method | Whitebox |
| Consultants Engaged | 1 |
| Level of Effort | 3 person-weeks |

### Vulnerability Summary

| | | |
|---|---|---|
| Total High-Severity Issues | 0 | |
| Total Medium-Severity Issues | 0 | |
| Total Low-Severity Issues | 1 | ■ |
| Total Informational-Severity Issues | 0 | |
| Total Undetermined-Severity Issues | 0 | |
| Total | 1 | |

### Category Breakdown

| | | |
|---|---|---|
| Patching | 1 | ■ |
| Total | 1 | |

## Code Maturity Evaluation

In the table below, we review the maturity of the codebase and the likelihood of future issues. In each area of control, we rate the maturity from strong to weak, or missing, and give a brief explanation of our reasoning.

| Category Name | Description |
|---|---|
| Access Controls | **Strong.** The contracts exposed various privileged operations. Appropriate access controls were in place for performing these operations, with the contract owner being able to delegate tightly scoped roles to particular addresses. |
| Arithmetic | **Satisfactory.** The contracts made consistent use of OpenZeppelin's `SafeMath` library functions to prevent overflows. Token balances were tracked and updated consistently across transfers and between partitions. |
| Assembly Use | **Not Applicable.** The contracts did not make use of any assembly code. |
| Centralization | **Satisfactory.** The token contract owner was only able to set partition strategies and otherwise could not exert undue influence over the token itself. The collateral manager contract had several roles for managing the contract. |
| Contract Upgradeability | **Not Applicable.** The contracts did not employ any upgradeability framework. |
| Function Composition | **Satisfactory.** Functions were broken down methodically into internal function calls. They were also organized in a logical way and adhered to a consistent coding style. |
| Front-Running | **Satisfactory.** The Amp token contract included the common `increaseAllowance` and `decreaseAllowance` functions to help mitigate the ERC20 race condition. No other areas where front-running may benefit an attacker were identified. |
| Monitoring | **Strong.** The contracts made very thorough use of events as these were required by other off-chain components operated by Flexa. |
| Specification | **Satisfactory.** The code had thorough comment coverage, and good high level documentation. Due to the complexity of the code and various standards implemented, this was essential for understanding the smart contracts. Additional high-level or formal specification would have been helpful. |

| | |
|---|---|
| Testing & Verification | **Satisfactory.** The repositories included tests for a variety of scenarios. |

# Engagement Goals

The engagement was scoped to provide a security assessment of the Flexa Collateral Manager and Amp token smart contracts.

Specifically, we sought to answer the following questions:

- Are errors handled appropriately within the contracts?
- Do the various token standards that the smart contracts fully or partially implement conflict with each other?
- Are balances tracked properly across token partitions?
- Is the swap token, Flexacoin, integrated properly?
- Can a malicious user gain unauthorized access to funds or other privileged functionality?

# Coverage

**Error Handling.** We performed automated review with [Slither](https://github.com/crytic/slither) and conducted manual review to ensure that all return values were being checked and handled properly. We also looked for instances where reverts could trap the contract, especially any areas where overflow might cause `SafeMath` to trigger a revert.

**Authentication.** We reviewed the authentication used in the smart contracts. We reviewed the various roles used by the contracts and ensured that access controls were set appropriately for administrative functionality.

**Authorization.** We reviewed the authorization used smart contracts. In addition to standard token allowances, Amp has the concept of an operator that allows a user to grant an actor full control of their balances, either globally or configured by partition. We reviewed the token transfer permissions to ensure users were able to move only the tokens they were intended to.

**Data Validation.** We reviewed the data validation performed within the smart contracts. Our automated and manual analyses focused on the unpacking and validation of data used for partitions and partition strategies, as well as the operator data used to signal various operations within the collateral manager contract.

**Logging.** We focused on ensuring events were emitted for compliance with standards, when administrative actions were performed, and where needed by the collateral manager contract to support off-chain components.

# Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

## Short Term

+ **Refactor the `IAmp` smart contract into its own stand-alone contract and import the interface when needed.** This will prevent problems associated with Truffle and avoid a potential point of confusion. (TOB-FLXA-001)

## Long Term

+ **Ensure that all newly introduced contracts have unique names.** This will ensure that build tools produce complete build artifacts, and prevent problems with integrating the third-party tooling that relies on them. (TOB-FLXA-001)

# Findings Summary

| # | Title | Type | Severity |
|---|---|---|---|
| 1 | Duplicate contract names may lead to errors | Patching | Low |

## 1. Duplicate contract names may lead to errors

Severity: Low
Type: Patching
Target: `amp-contracts/contracts`

Difficulty: Low
Finding ID: TOB-FLXA-001

**Description**
The codebase contains multiple contracts with the same name. This pattern is error-prone and will make Truffle generate incorrect compilation artifacts, which could break third-party integrations.

The duplicates are:

- `IAmp`
  - `mocks/MockCollateralPool.sol`
  - `partitions/HolderCollateralPartitionValidator.sol`
  - `mocks/ExampleCollateralManager.sol`
  - `partitions/HolderCollateralPartitionValidator.sol`
  - `partitions/CollateralPoolPartitionValidator.sol`
  - `partitions/HolderCollateralPartitionValidator.sol`

**Exploit Scenario**
Bob develops a third-party application that reads the Amp token's compilation artifacts, but the app can't read the artifacts correctly and does not work.

**Recommendation**
Short term, refactor `IAmp` into its own stand-alone contract and import the interface when needed. This will prevent problems associated with Truffle, and avoid a potential point of confusion.

Long term, ensure that all newly introduced contracts have unique names. This will ensure that build tools produce complete build artifacts and prevent problems with integrating the third-party tooling that relies on them.

# A. Vulnerability Classifications

| Vulnerability Classes | |
|---|---|
| **Class** | **Description** |
| Access Controls | Related to authorization of users and assessment of rights |
| Auditing and Logging | Related to auditing of actions or logging of problems |
| Authentication | Related to the identification of users |
| Configuration | Related to security configurations of servers, devices, or software |
| Cryptography | Related to protecting the privacy or integrity of data |
| Data Exposure | Related to unintended exposure of sensitive information |
| Data Validation | Related to improper reliance on the structure or values of data |
| Denial of Service | Related to causing system failure |
| Error Reporting | Related to the reporting of error conditions in a secure fashion |
| Patching | Related to keeping software up to date |
| Session Management | Related to the identification of authenticated users |
| Timing | Related to race conditions, locking, or order of operations |
| Undefined Behavior | Related to undefined behavior triggered by the program |

| Severity Categories | |
|---|---|
| **Severity** | **Description** |
| Informational | The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth |
| Undetermined | The extent of the risk was not determined during this engagement |
| Low | The risk is relatively small or is not a risk the customer has indicated is important |
| Medium | Individual user’s information is at risk, exploitation would be bad for |

| | |
|---|---|
| | client's reputation, moderate financial impact, possible legal implications for client |
| High | Large numbers of users, very bad for client's reputation, or serious legal or financial implications |

| **Difficulty Levels** | |
|---|---|
| **Difficulty** | **Description** |
| Undetermined | The difficulty of exploit was not determined during this engagement |
| Low | Commonly exploited, public tools exist or can be scripted that exploit this flaw |
| Medium | Attackers must write an exploit, or need an in-depth knowledge of a complex system |
| High | The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue |

# B. Code Maturity Classifications

| Code Maturity Classes | |
|---|---|
| **Category Name** | **Description** |
| Access Controls | Related to the authentication and authorization of components. |
| Arithmetic | Related to the proper use of mathematical operations and semantics. |
| Assembly Use | Related to the use of inline assembly. |
| Centralization | Related to the existence of a single point of failure. |
| Upgradeability | Related to contract upgradeability. |
| Function Composition | Related to separation of the logic into functions with clear purpose. |
| Front-Running | Related to resilience against front-running. |
| Key Management | Related to the existence of proper procedures for key generation, distribution, and access. |
| Monitoring | Related to use of events and monitoring procedures. |
| Specification | Related to the expected codebase documentation. |
| Testing & Verification | Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.). |

| Rating Criteria | |
|---|---|
| **Rating** | **Description** |
| Strong | The component was reviewed and no concerns were found. |
| Satisfactory | The component had only minor issues. |
| Moderate | The component had some issues. |
| Weak | The component led to multiple issues; more issues might be present. |
| Missing | The component was missing. |

| | |
|---|---|
| Not Applicable | The component is not applicable. |
| Not Considered | The component was not reviewed. |
| Further Investigation Required | The component requires further investigation. |

# C. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- We noted one `TODO` included inline in the Amp token contract (however it is typo'd as "`TOOD`"). This `TODO` should be addressed, and then removed appropriately. Furthermore, inline `TODO`s should be lifted out of the code and into a project issue tracker for completion.

# D. Echidna Property Testing

Trail of Bits used Echidna, our property-based testing framework, to test for logic errors in the Solidity components of Amp.

Trail of Bits developed a custom Echidna testing harness for the Amp token that implements the ERC20 standard along with features from several other token standards. This harness initializes the token and mints an appropriate amount of tokens for three users. It then executes a random sequence of API calls from the Amp contract in an attempt to cause anomalous behavior.

This harness includes tests of ERC20 invariants (e.g., capped `totalSupply`, `balanceOf` correctness), and ERC20 edge cases (e.g., transferring tokens to oneself and transferring zero tokens). Upon completion of the engagement, these harnesses and their related tests will be delivered to the Flexa team.

Figure D.1 shows the Solidity source code used to define, initialize, and test the Amp contract. The script defines a token contract used as the single component of the Amp contract to test. An example of how to run this test with Echidna is shown in Figure D.2.

```
import "../amp/Amp.sol";

contract CryticInterface {
    address internal crytic_owner = address(0x41414141);
    address internal crytic_user = address(0x42424242);
    address internal crytic_attacker = address(0x43434343);
}

contract CryticAmp is CryticInterface, Amp {

       uint _value;
       uint initialTotalSupply;

       constructor() public Amp(address(0x1), "Amp Token", "AMP") {
       _value = 10000000000;
       initialTotalSupply = 3*_value;

        _mint(crytic_user, crytic_user, _value);
        _mint(crytic_owner, crytic_owner, _value);
        _mint(crytic_attacker, crytic_attacker, _value);

       }

    function crytic_totalSupply_consistent_ERC20Properties() public returns (bool) {
```

```
        return this.balanceOf(crytic_owner) + this.balanceOf(crytic_user) +
this.balanceOf(crytic_attacker) <= this.totalSupply();
    }

    function crytic_revert_transfer_to_zero_ERC20PropertiesTransferable() public returns
(bool) {
       return this.transfer(address(0x0), this.balanceOf(msg.sender));
    }

    ...
}
```

*Figure D.1: Amp token test harness initialization and example property tests.*

```
ethsec@6f70e4109877:/flexa$ echidna-test contracts/crytic/crytic_amp.sol --contract CryticAmp
--config contracts/crytic/crytic_amp.yaml
Analyzing contract: /flexa/contracts/crytic/crytic_amp.sol:CryticAmp
crytic_self_transferFrom_to_other_ERC20PropertiesTransferable: passed! 🎉
crytic_totalSupply_consistent_ERC20Properties: passed! 🎉
crytic_zero_always_empty_ERC20Properties: passed! 🎉
crytic_self_transfer_ERC20PropertiesTransferable: passed! 🎉
crytic_self_transferFrom_ERC20PropertiesTransferable: passed! 🎉
crytic_revert_transferFrom_to_zero_ERC20PropertiesTransferable: passed! 🎉
crytic_supply_constant_ERC20PropertiesNotMintableNotBurnable: passed! 🎉
crytic_revert_transfer_to_zero_ERC20PropertiesTransferable: passed! 🎉
crytic_less_than_total_ERC20Properties: passed! 🎉

Seed: 653818515397280526
```

*Figure D.2: An example run of Echidna with the `crytic_amp.sol` test harness, including test results.*