

## A CONSENSYS DILIGENCE AUDIT REPORT

# Amp

Date	June 2020
Lead Auditor	Shayan Eskandari
Co-auditors	Valentin Wüstholtz

## 1 Executive Summary

This report presents the results of our engagement with **Flexa** to review **Amp Token** and **Flexa Collateral Manager**. Flexa is a payment network, using smart contract and Amp collateral to facilitate off-chain payments.

The review was conducted over the course of two weeks, from **June 9, 2020** to **June 19, 2020** by Shayan Eskandari and Valentin Wüstholtz. A total of **15** person-days were spent.

The review of the initial report fixes were performed from August 10, 2020 to August 14, 2020 by Shayan Eskandari.

## 2 Scope

Our review focused on the commit hash `4203e96d1138632a991d072d0c232fd8ba69c9e1` for `amp-contracts`, and `4203e96d1138632a991d072d0c232fd8ba69c9e1` for `flexa-collateral-manager`. The list of files in scope can be found in the [Appendix](#).

**Update:** The final review of the initial report fixes were done on the commit hash `aece0f6b24df6348221da548a815528a6633a20e` for `amp-token-contracts`, and `1c295c2ed5d3eef12e5992d96efb8d10d2d3` for `flexa-collateral-manager`.



## 2.1 Objectives

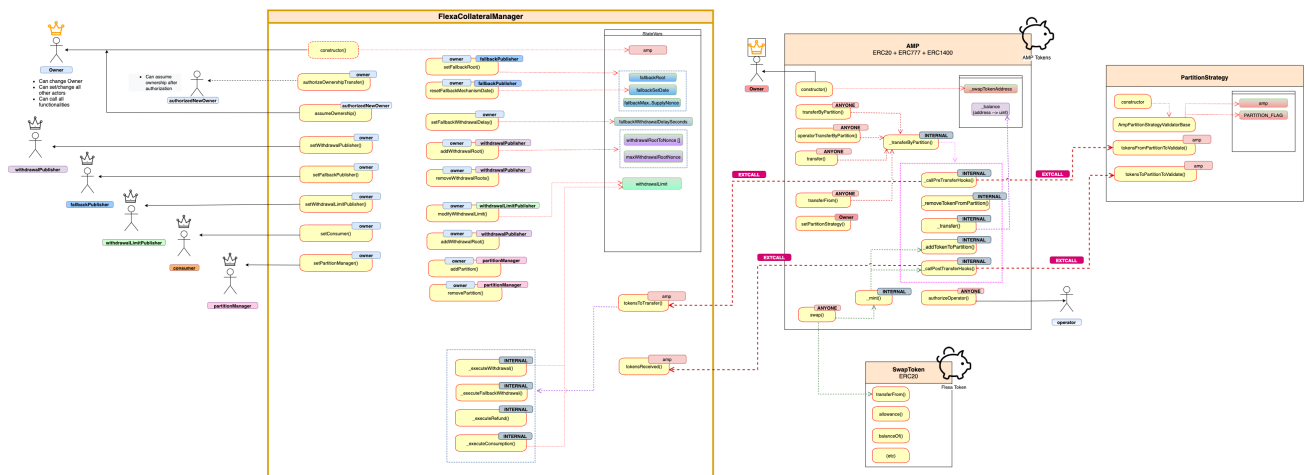
Together with the the **Flexa** team, we identified the following priorities for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our [Smart Contract Best Practices](#), and the [Smart Contract Weakness Classification Registry](#).

The second review was mainly a check for the fixes of the issues filed in the initial report. The rest of the text in this report reflects the initial review unless explicitly tagged as **Update**.

## 3 System Overview

The following figure is a visualization of the actors and the overview of Flexa Collateral Manager with Amp Token:



Many of the internal calls to view/pure functions and details regarding partitions are removed from this chart for more readability.

The Actors and their permissions in the system are described in [Security specification](#) section.

## 4 Security Specification



section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The

purpose of this section is to identify specific security properties that were validated by the audit team.

## 4.1 Actors

The relevant actors are listed below with their respective abilities:

### Flexa Collateral Manager

- **owner**
  - *Can change ownership*
  - *Can set and change all other actors in the system at any time*
  - *Can call all functionalities that other actors can call*
  - *Can update the delay period (time-lock) in which fallback mechanism is activated ( `fallbackWithdrawalDelaySeconds` )*
- **withdrawalPublisher**
  - *Can add Merkle Root for authorized token withdrawals*
    - Note that **the root is not verified** and can be an invalid value. Also the call to add root will remove the specified previous withdrawal roots in the smart contract.
  - *Can remove any of the previously added roots in the smart contract*
- **fallbackPublisher**
  - *Can set Fallback Merkle Root, which will update the fallback SetDate, MaxIncludedSupplyNonce and the root itself.*
    - Note that **the root is not verified** and can be an invalid value.
  - *Can reset fallback mechanism date, resulting in delay in fallback period without publishing new root, or to deactivate the fallback mechanism temporarily*
- **withdrawalLimitPublisher**
  - *Can modify the global withdrawal limit*
    - Note: setting this limit to 0, disables all withdrawals and breaks the executions (e.g. `_executeWithdrawal`, `_executeConsumption` )
    - The value of `withdrawalLimit` also is decreased after every consumer execution



- **consumer**
  - Can execute consumption (consumption transfers)
    - Note that consumer is trusted, as in if consumer executes a transfer, he can spend up to `withdrawalLimit` which then all withdrawals will be impossible until `withdrawalLimitPublisher` modifies the `withdrawalLimit` to a number other than 0 to re-enable withdrawals.
- **partitionManager**
  - Can add & remove new partitions to the system
    - Removed partition will be disallowed from incoming transfers

All the above actors in this system are trusted in this system, meaning that they could misbehave and temporarily block other functionalities of the system, however they all can be replaced by `owner` as well.

**Update:** *consumer* was renamed to *directTransferer* to remove confusion. All associated actions were also renamed, such as *consume* → *directTransfer* and so on.

## Amp Token

- **owner**
  - Can set partition strategy addresses, linking PartitionStrategy contracts to specific prefixes in the system
- **operator**
  - Anyone can authorize an Operator for all their token balance or a specific partition.
  - Can transfer from user's balance (or the partition the operator is authorized for)
  - Any token holder is also his own operator

## 4.2 Important Security Properties & Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:



- Flexa (Actors in the system) are trusted, they can misbehave and update the contract in a way that the withdrawals are blocked.
- The Merkle Tree Roots published on the smart contract can be removed/replaced by the publisher actor, making the valid withdrawals invalid. However the premise is that the publisher will act honestly and is part of the system.
- There are some concerns about external call in `Amp.Swap()` regarding reentrancy or other malicious token implementations, however as `swapToken` here is previously deployed Flexa ERC20 token, we assume the token is trusted and does not have malicious intentions.
- It should be noted that ERC777 introduces the hooks that have been used for reentrancy attack vectors in other DApps that have interacted with the ERC777 smart contract.
- Partition Strategies are set by the Amp owner and we assume they are trusted, as there are external calls to the functions defined in these contracts.

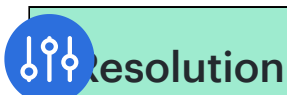
## 5 Issues

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

### 5.1 Eliminate assembly code by using ABI decode **Major**

✓ Fixed



All assembly code was replaced with proper use of `abi.decode()`.

## Description

There are several locations where assembly code is used to access and decode byte arrays (including uses inside loops). Even though assembly code was used for gas optimization, it reduces the readability (and future updatability) of the code.

## Examples

### code/amp-contracts/contracts/partitions/PartitionsBase.sol:L39-L44

```
assembly {
    flag := mload(add(_data, 32))
}
if (flag == CHANGE_PARTITION_FLAG) {
    assembly {
        toPartition := mload(add(_data, 64))
    }
}
```

### code/amp-contracts/contracts/partitions/PartitionsBase.sol:L43-L44

```
assembly {
    toPartition := mload(add(_data, 64))
}
```

Same code as above is also present here:

```
/flexa-collateral-manager/contracts/FlexaCollateralManager.sol#L1403
```

```
flexa-collateral-manager/contracts/FlexaCollateralManager.sol#L1407
```

### code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L1463-L1470

```
for (uint256 i = 116; i <= _operatorData.length; i = i + 32) {
    bytes32 temp;
    assembly {
        temp := mload(add(_operatorData, i))
    }
    proof[index] = temp;
    index++;
}
```



## Recommendation

As discussed in the mid-audit meeting, it is a good solution to use ABI decode since all uses of assembly simply access 32-byte chunks of data from user input. This should eliminate all assembly code and make the code significantly more clean. In addition, it might allow for more compact encoding in some cases (for instance, by eliminating or reducing the size of the flags).

This suggestion can be also applied to Merkle Root verifications/calculation code, which can reduce the for loops and complexity of these functions.

## 5.2 Ignored return value for transferFrom call Major ✓ Fixed

### Resolution

Fixed by adding a `require` to validate the success/failure of `transferFrom()`.

## Description

When burning swap tokens the return value of the `transferFrom` call is ignored. Depending on the token's implementation this could allow an attacker to mint an arbitrary amount of Amp tokens.

Note that the severity of this issue could have been Critical if [Flexa token](#) was any arbitrarily tokens. We quickly verified that Flexa token implementation would revert if the amount exceeds the allowance, however it might not be the case for other token implementations.

**code/amp-contracts/contracts/Amp.sol:L619-L620**

```
swapToken.transferFrom(_from, swapTokenGraveyard, amount);
```

## Recommendation



The code should be changed like this:

```
require(swapToken.transferFrom(_from, swapTokenGraveyard, amount));
```

## 5.3 No integration tests for the two main components Medium

✓ Fixed

### Resolution

`amp-contracts` added as a submodule to `collateral-manager` and full integration tests added

It is recommended to write test suites that achieve high code coverage to prevent missing obvious bugs that tests could cover.

### Description

The existing tests cover each of the two main components and each set of tests mocks the other component. While this is good for unit testing some issues might be missed without proper system/integration tests that cover all components.

### Recommendation

Consider adding system/integration tests for all components. As we've seen in the recent issues in multi-contract smart contract systems, it's becoming more crucial to have a full test suits for future changes to the code base. Not having inter-component tests, could result in issues in the next development and deployment cycles.

## 5.4 Potentially insufficient validation for operator transfers

Medium✓ Fixed

### Resolution



removing `operatorTransferByPartition` and simplifying the interfaces to only `transferByPartition`



This removes the existing `transferByPartition`, converting `operatorTransferByPartition` to it. The reason for this is to make the client interface simpler, where there is one method to transfer by partition, and that method can be called by either a sender wanting to transfer from their own address, or an operator wanting to transfer from a different token holder address. We found that it was redundant to have multiple methods, and the client convenience wasn't worth the confusion.

## Description

For operator transfers, the current validation does not require the sender to be an operator (as long as the transferred value does not exceed the allowance):

**code/amp-contracts/contracts/Amp.sol:L755-L759**

```
require(
    _isOperatorForPartition(_partition, msg.sender, _from) ||
    (_value <= _allowedByPartition[_partition][_from][msg.sender]),
    EC_53_INSUFFICIENT_ALLOWANCE
);
```

It is unclear if this is the intention or whether the logical `or` should be a logical `and`.

## Recommendation

Confirm that the code matches the intention. If so, consider documenting the behavior (for instance, by changing the name of function

`operatorTransferByPartition`).

## 5.5 Potentially missing nonce check Medium Acknowledged

### Resolution



Nothing was done here, as Dave M writes:

The first two are working as intended, and the third does check that the value is monotonically increasing.

## Description

When executing withdrawals in the collateral manager the per-address withdrawal nonce is simply updated without checking that the new nonce is one greater than the previous one (see Examples). It seems like without such a check it might be easy to make mistakes and causing issues with ordering of withdrawals.

## Examples

**code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L663-L664**

```
addressToWithdrawalNonce[_partition][supplier] = withdrawalRootNonce;
```

**code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L845-L846**

```
addressToWithdrawalNonce[_partition][supplier] = maxWithdrawalRootNonce;
```

**code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L1155-L1156**

```
maxWithdrawalRootNonce = _nonce;
```

## Recommendation

Consider adding more validation and sanity checks for nonces on per-address withdrawals.



## 5.6 Unbounded loop when validating Merkle proofs Medium

✓ Fixed

### Resolution

The loop was removed by switching to `abi.decode`.

### Description

It seems like the loop for validating Merkle proofs is unbounded. If possible it would be good to have an upper bound to prevent DoS-like attacks. It seems like the depth of the tree, and thus, the length of the proof could be bounded.


This could also simplify the decoding and make it more robust. For instance, in `_decodeWithdrawalOperatorData` it is unclear what happens if the data length is not a multiple of 32. It seems like it might result in out-of-bound reads.

**code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L1460-L1470**

```
uint256 proofNb = (_operatorData.length - 84) / 32;
bytes32[] memory proof = new bytes32[](proofNb);
uint256 index = 0;
for (uint256 i = 116; i <= _operatorData.length; i = i + 32) {
    bytes32 temp;
    assembly {
        temp := mload(add(_operatorData, i))
    }
    proof[index] = temp;
    index++;
}
```

### Recommendation

Consider enforcing a bound on the length of Merkle proofs.

Also note that if similar mitigation method as [issue 5.1](#) is used, this method  be replaced by a simpler function using ABI Decode, which does not

have any unbounded issues as the sizes of the hashes are fixed (or can be indicated in the passed objects)

## 5.7 Mitigation for possible reentrancy in token transfers

**Medium****✓ Fixed**

### Resolution

Fixed as recommended.

### Description

ERC777 adds significant features to the token implementation, however there are some known risks associated with this token, such as [possible reentrancy attack vector](#). Given that the Amp token uses hooks to communicate to Collateral manager, it seems that the environment is trusted and safe. However, a minor modification to the implementation can result in safer implementation of the token transfer.

### Examples

In `Amp.sol --> _transferByPartition()`

**`code/amp-contracts/contracts/Amp.sol:L1152-L1177`**



```

require(
    _balanceOfByPartition[_from][_fromPartition] >= _value,
    EC_52_INSUFFICIENT_BALANCE
);

bytes32 toPartition = _fromPartition;
if (_data.length >= 64) {
    toPartition = _getDestinationPartition(_fromPartition, _data);
}

_callPreTransferHooks(
    _fromPartition,
    _operator,
    _from,
    _to,
    _value,
    _data,
    _operatorData
);

_removeTokenFromPartition(_from, _fromPartition, _value);
_transfer(_from, _to, _value);
_addTokenToPartition(_to, toPartition, _value);

_callPostTransferHooks(
    toPartition,

```

## Recommendation

It is suggested to move any condition check that is checking the balance to after the external call. However `_callPostTransferHooks` needs to be called after the state changes, so the suggested mitigation here is to move the `require` at line 1152 to after `_callPreTransferHooks()` function (e.g. line 1171).

## 5.8 Potentially inconsistent input validation Medium ✓ Fixed

### Resolution

`transferWithData` was removed as a resolution of another filed issue, the rest are documented properly.



The `msg.sender` cannot be authorized or revoked from being an operator for itself. This should also be clear from the natspec comments now.

## Description

There are some functions that might require additional input validation (similar to other functions):

## Examples

- `Amp.transferWithData` :  
`require(!_isOperator(msg.sender, _from), EC_58_INVALID_OPERATOR);` like in

### code/amp-contracts/contracts/Amp.sol:L699

```
require(!_isOperator(msg.sender, _from), EC_58_INVALID_OPERATOR);
```

- `Amp.authorizeOperatorByPartition` : `require(_operator != msg.sender);` like in

### code/amp-contracts/contracts/Amp.sol:L789

```
require(_operator != msg.sender);
```

- `Amp.revokeOperatorByPartition` : `require(_operator != msg.sender);` like in

### code/amp-contracts/contracts/Amp.sol:L800

```
require(_operator != msg.sender);
```

## Recommendation

Consider adding additional input validation.

## 5.9 ERC20 compatibility of Amp token using defaultPartition

Medium

✓ Fixed



## Resolution

This fix resulted in significant changes to the token allowance work flow. The new implementation of `balanceOf` represents the total balance of tokens at that address (across any partition), instead of only default partition.

The approve + allowance based operations were using a distinct global allowance mapping, while the rest of the ERC20 compat operations were using the partition state mappings with the default partition. This makes the allowance operations behave the same as the balance based operations.

## Description

It is somewhat unclear how the Amp token ensures ERC20 compatibility. While the `default` partition is used in some places (for instance, in function `balanceOf`) there are also separate fields for (aggregated) balances/allowances. This seems to introduce some redundancy and raises certain questions about when which fields are relevant.

## Examples

- `_allowed` is used in function `allowance` instead of `_allowedByPartition` with the default partition
- An `Approval` event should be emitted when approving the default partition

### code/amp-contracts/contracts/Amp.sol:L1494

```
emit ApprovalByPartition(_partition, _tokenHolder, _spender, _amount);
```

- `increaseAllowance()` VS. `increaseAllowanceByPartition()`



## commendation

After the mid-audit discussion, it was clear that the general `balanceOf` method (with no partition) is not needed and can be replaced with a `balanceOf` function that returns balance of the default partition, similarly for allowance, the general `increaseAllowance` function can simply call `increaseAllowanceByPartition` using default partition (same for `decreaseAllowance`).

## 5.10 Duplicate code better be moved to shared library Minor

✓ Fixed

### Resolution

aforementioned functions were moved to a shared library `PartitionUtils.sol`, which also fixed the inconsistency in function implementations.

### Description

There are some functionalities that the code is duplicated between different smart contracts.

### Examples

- `_getDestinationPartition()` is present in both `PartitionBase.sol` and `FlexaCollateralManager.sol`
  - Note that in `PartitionBase` the usage results in dead code in the contract.

**code/amp-contracts/contracts/Amp.sol:L1158-L1160**

```
if (_data.length >= 64) {  
    toPartition = _getDestinationPartition(_fromPartition, _data);  
}
```

**code/amp-contracts/contracts/partitions/PartitionsBase.sol:L33-L36**





```
toPartition = _fromPartition;
if (_data.length < 64) {
    return toPartition;
}
```

- `_splitPartition()` is present in `FlexaCollateralManager.sol`, `PartitionBase.sol` with slightly different implementations. One has an extra return value for `subPartition` which is not used in the code under audit

## Recommendation

Use a shared library for these functions, possibly `PartitionBased.sol` can be used in Collateral Manager.

## 5.11 Additional validation for `canReceive` Minor ✓ Fixed

### Resolution

Added proper checks and merged `_canReceive()` with `canReceive()`.

## Description

For `FlexaCollateralManager.tokensReceived` there is validation to ensure that only the Amp calls the function. In contrast, there is no such validation for `canReceive` and it is unclear if this is the intention.

## Examples

**code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L492-L493**

```
require(msg.sender == amp, "Invalid sender");
```

## Recommendation

Consider adding a conjunct `msg.sender == amp` in function `_canReceive`.



## code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L468-L470

```
function _canReceive(address _to, bytes32 _destinationPartition) internal view  
    return _to == address(this) && partitions[_destinationPartition];  
}
```

### 5.12 Update to Solidity 0.6.10 Minor ✓ Fixed

#### Resolution

Updated to 0.6.10 .

#### Description

Due to an [issue found](#) in 0.6.9, it is recommended to update the compiler version to latest version 0.6.10.

### 5.13 Discrepancy between code and comments Minor ✓ Fixed

#### Description

There are some discrepancies between (uncommented) code and the documentations comment:

#### Examples

## code/amp-contracts/contracts/Amp.sol:L459-L462

```
// Indicate token verifies Amp, ERC777 and ERC20 interfaces  
ERC1820Implementer._setInterface(AMP_INTERFACE_NAME);  
ERC1820Implementer._setInterface(ERC20_INTERFACE_NAME);  
// ERC1820Implementer._setInterface(ERC777_INTERFACE_NAME);
```

## code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L268-L279



```
/**
 * @notice Indicates a supply refund was executed
 * @param supplier Address whose refund authorization was executed
 * @param partition Partition from which the tokens were transferred
 * @param amount Amount of tokens transferred
 */
event SupplyRefund(
    address indexed supplier,
    bytes32 indexed partition,
    uint256 amount,
    uint256 indexed nonce
);
```

## Recommendation

Consider updating either the code or the comment.

### 5.14 Several fields could potentially be private Minor

Acknowledged

#### Resolution

Comment from Flexa team:

We audited the suggested fields, and determined that we would like them to be public for transparency and/or functionality reasons.

## Description

Several fields in Amp could possibly be private:

## Examples

- swapToken :

**code/amp-contracts/contracts/Amp.sol:L261**



```
ISwapToken public swapToken;
```

- `swapTokenGraveyard` :

### code/amp-contracts/contracts/Amp.sol:L268

```
address public constant swapTokenGraveyard = 0x0000000000000000000000000000000000000000000000000000000000000000;
```

- `collateralManagers` :

### code/amp-contracts/contracts/Amp.sol:L236

```
address[] public collateralManagers;
```

- `partitionStrategies` :

### code/amp-contracts/contracts/Amp.sol:L248

```
bytes4[] public partitionStrategies;
```

The same hold for several fields in `FlexaCollateralManager` . For instance:

- `partitions` :

### code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L78

```
mapping(bytes32 => bool) public partitions;
```

- `nonceToSupply` :

### code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L144

```
mapping(uint256 => Supply) public nonceToSupply;
```

- `withdrawalRootToNonce` :



### code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L163

```
mapping(bytes32 => uint256) public withdrawalRootToNonce;
```

## Recommendation

Double-check that you really want to expose those fields.

### 5.15 Several fields could be declared immutable Minor

Acknowledged

#### Resolution

Comment from Flexa team:

We tried to add this, but found that it made validating the contract on Etherscan impossible. We have added comments to a reader of the contract indicating the fields are immutable after deployment, though.

## Description

Several fields could be declared immutable to make clear that they never change after construction:

## Examples

- `Amp._name` :

**code/amp-contracts/contracts/Amp.sol:L129**

```
string internal _name;
```

- `Amp._symbol` :

**code/amp-contracts/contracts/Amp.sol:L134**



```
string internal _symbol;
```

- `Amp.swapToken` :

### **code/amp-contracts/contracts/Amp.sol:L261**

```
ISwapToken public swapToken;
```

- `FlexaCollateralManager.amp` :

### **code/flexa-collateral-manager/contracts/FlexaCollateralManager.sol:L73**

```
address public amp;
```

## **Recommendation**

Use the `immutable` annotation in Solidity (see [Immutable](#)).

# **Appendix 1 - Artifacts**

This section contains some of the artifacts generated during our review by automated tools, the test suite, etc. If any issues or recommendations were identified by the output presented here, they have been addressed in the appropriate section above.

## **A.1.1 Harvey**

As part of the audit, we performed several fuzzing campaigns using Harvey, our in-house [greybox fuzzer](#) for smart contracts, to check 8 custom properties. In order to fuzz the entire contract system, we used Flexa's existing deployment scripts to set up an initial state for the fuzzer containing the following contracts:

- Amp
- FlexaCollateralManager
- ERC1820Registry
- MockFXC
- HolderCollateralPartitionValidator
- CollateralPoolPartitionValidator

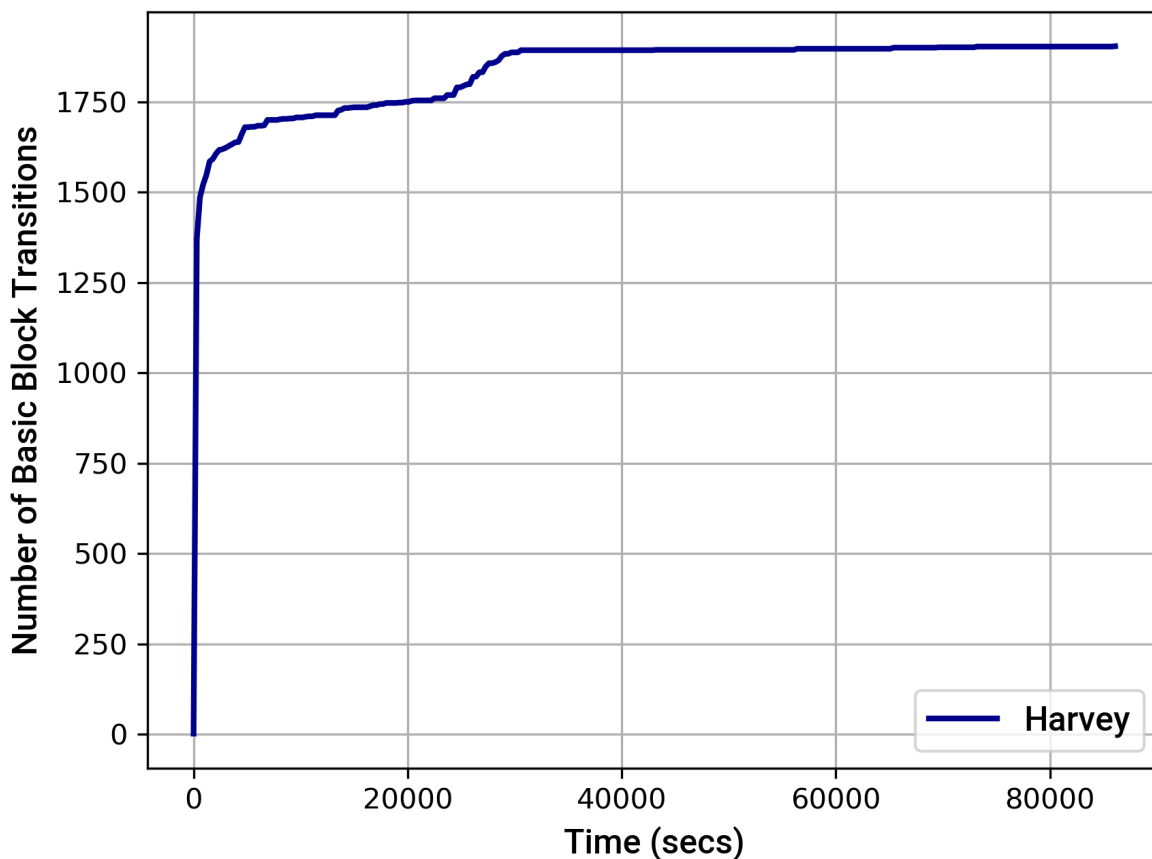


We extended the deployment scripts to distribute MockFXC tokens to several known users such that they could interact with the system after approving the Amp contract. We also made a small number of changes to the code to improve the effectiveness of the fuzzer. Our final 24-hour fuzzing campaign was able to detect 3 property violations that were reviewed as part of the audit.

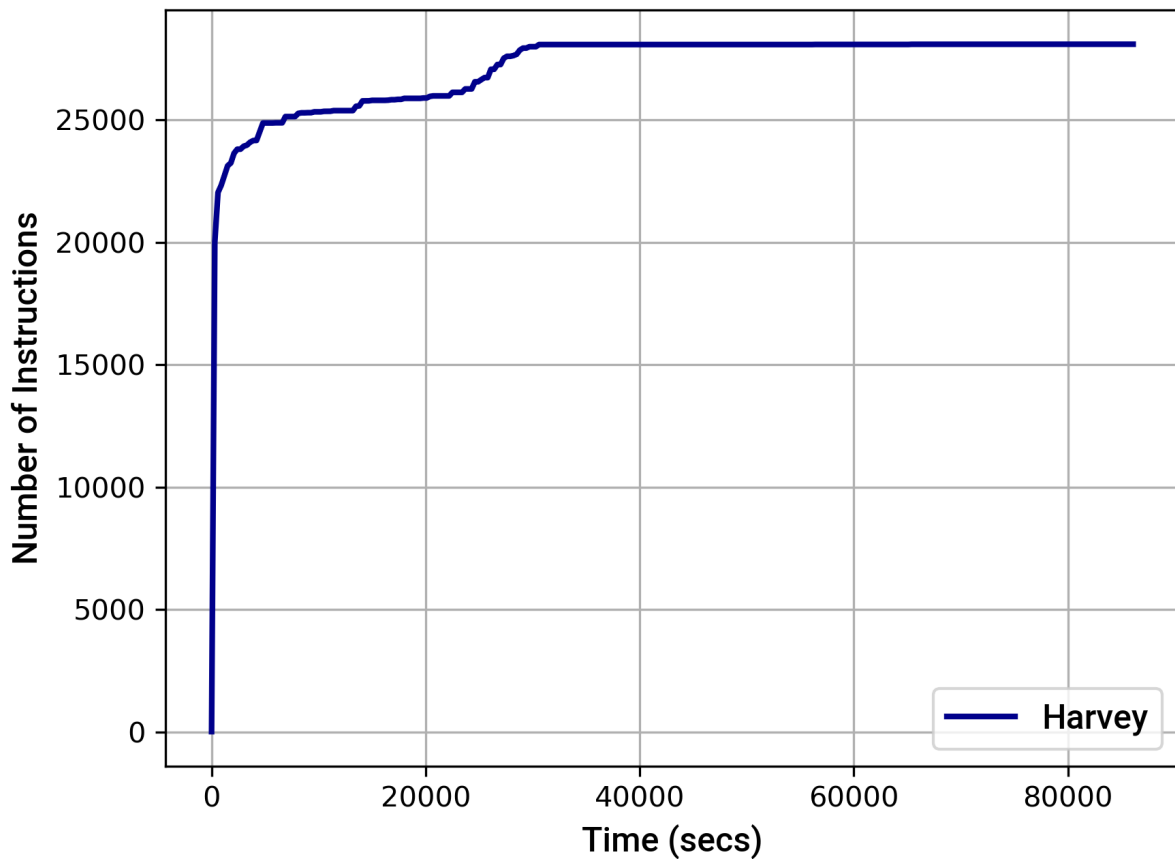
Note that—to support the auditor—several properties check if certain (not necessarily bad/dangerous) states are feasible and violations do not necessarily indicate issues with the code.

The graphs below provide an indication of the instruction and basic block transition coverage achieved by Harvey over time. After 24 hours, Harvey achieved the following coverage:

- EVM instruction coverage: 28090
- Path coverage: 7747
- EVM basic block transition coverage: 1904



## 1.2 Tests Coverage



Below is the coverage output generated by running the test suite:

File	% Stmts	% Branch	% Funcs	% Lines	Unc d L
contracts/	99.44	90.79	100	99.44	
Amp.sol	99.44	90.79	100	99.44	1571
contracts/ codes/	100	100	100	100	
ErrorCode s.sol	100	100	100	100	
contracts/ erc1820/	88.89	100	80	88.89	
ERC1820Cl ient.sol	80	100	66.67	80	53
ERC1820I mplement er.sol	100	100	100	100	



File	% Stmts	% Branch	% Funcs	% Lines	Unc d L
contracts/ extensions /	100	100	100	100	
IAmpToken sRecipient.sol	100	100	100	100	
IAmpToken sSender.sol	100	100	100	100	
contracts/ mocks/	88.33	78.26	82.86	88.98	
ExampleC ollateralMa nager.sol	89.02	71.88	87.5	89.53	... 305, 10
MockAmp TokensRec ipient.sol	85.71	100	75	87.5	29
MockAmp TokensSen der.sol	85.71	100	75	87.5	29
MockColla teralPool.s ol	84.21	83.33	66.67	85	58,71
MockERC2 OInteracto r.sol	100	100	100	100	
MockFXC.s ol	100	100	100	100	



File	% Stmts	% Branch	% Funcs	% Lines	Unc d L
MockPartit ionBase.sol	100	100	100	100	
contracts/ partitions/	95	75	86.67	95.24	
AmpPartiti onStrateg yValidator Base.sol	80	100	50	80	88
CollateralP oolPartitio nValidator. sol	100	80	100	100	
HolderColl ateralPartit ionValidat or.sol	100	71.43	100	100	
IAmpPartit ionStrateg yValidator. sol	100	100	100	100	
PartitionsB ase.sol	90.91	75	100	92.31	35
<b>All files</b>	<b>94.83</b>	<b>84.21</b>	<b>91.67</b>	<b>94.96</b>	

File	% Stmts	% Branch	% Funcs	% Lines	Unc d L
contracts/	88.78	83.65	89.13	89.22	
FlexaCollat eralManag er.sol	88.78	83.65	89.13	89.22	... 51,958



File	% Stmts	% Branch	% Funcs	% Lines	Unc d L
contracts/ amp/	50	100	60	50	
IAmp.sol	100	100	100	100	
IAmpToken Recipient.sol	100	100	100	100	
IAmpToken Sender.sol	100	100	100	100	
MockAmp. sol	50	100	60	50	23,25
contracts/ erc1820/	100	100	100	100	
ERC1820Cl ient.sol	100	100	100	100	
<b>All files</b>	<b>87.38</b>	<b>83.65</b>	<b>86.54</b>	<b>87.85</b>	

It's important to note that "100% test coverage" is not a silver bullet. Our review also included a inspection of the test suite to ensure that testing included important edge cases.

## Appendix 2 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
flexa-collateral- manager/contracts/FlexaCollateralManag er.sol	416586f57559b9c673ebde0 6121f314d9f781c42



File	SHA-1 hash
flexa-collateral-manager/contracts/amp/IAmp.sol	cf5b55544a4aa60e86461e5f59b067218e1b5e1f
flexa-collateral-manager/contracts/amp/IAmpTokensRecipient.sol	bc6989130031ab842d44b07fb4869631743f86f8
flexa-collateral-manager/contracts/amp/IAmpTokensSender.sol	87c1435d51fbb6ab35a63adb7f9d89432edfe724
flexa-collateral-manager/contracts/erc1820/ERC1820Client.sol	0efb9dca16afe6da2b8b4b25408b985e1ef289b0
amp-contracts/contracts/Amp.sol	d7c402dcd9b9edf88d7cec739932d1f6c2259437
flexa-collateral-manager/contracts/FlexaCollateralManager.sol	416586f57559b9c673ebde06121f314d9f781c42
flexa-collateral-manager/contracts/amp/IAmp.sol	cf5b55544a4aa60e86461e5f59b067218e1b5e1f
flexa-collateral-manager/contracts/amp/IAmpTokensRecipient.sol	bc6989130031ab842d44b07fb4869631743f86f8
flexa-collateral-manager/contracts/amp/IAmpTokensSender.sol	87c1435d51fbb6ab35a63adb7f9d89432edfe724
flexa-collateral-manager/contracts/erc1820/ERC1820Client.sol	0efb9dca16afe6da2b8b4b25408b985e1ef289b0
amp-contracts/contracts/erc1820/ERC1820Client.sol	e99262a96ee7e3d055055cffe4168e8497ce2b0



File	SHA-1 hash
amp-contracts/contracts/erc1820/ERC1820Implementer.sol	b82a2caee3db82521bcf3d84412441db9d1c139a
amp-contracts/contracts/partitions/IAmpPartitionStrategyValidator.sol	6e55bfed60d5175b9adef6aa33822c3c078ee93
amp-contracts/contracts/partitions/PartitionsBase.sol	12b764eac1f6f5059d201a08a049c03989127538
amp-contracts/contracts/codes/ErrorCodes.sol	f6e6c7dcc9dff16ec98086ab82684650cc886100

## Appendix 3 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to



relied upon as investment advice, is not an endorsement of this project or

team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

**PURPOSE OF REPORTS** The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

**LINKS TO OTHER WEB SITES FROM THIS WEB SITE** You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

**TIMELINESS OF CONTENT** The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.

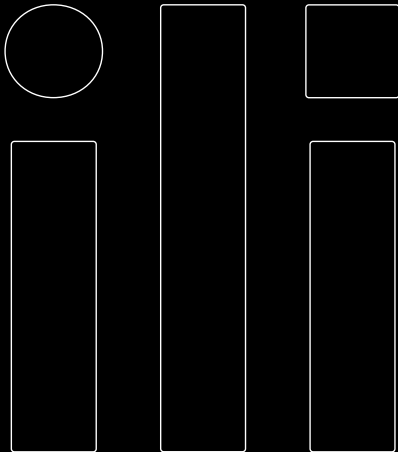




# Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

CONTACT US



AUDITS

FUZZING

SCRIBBLE

BLOG

TOOLS

RESEARCH

ABOUT

CONTACT

CAREERS

PRIVACY  
POLICY

## Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

Email\*

e-mail address



POWERED BY



CONSENSYS

