

# Aave Protocol V2

Date	September 2020
Lead Auditor	Bernhard Mueller
Co-auditors	Sergii Kravchenko

## 1 Executive Summary

This report presents the results of our engagement with Aave review version 2 of the Aave protocol. The review was conducted over 4 weeks, from September 8th, 2020 to October 9th 2020 by Bernhard Mueller and Sergii Kravchenko. A total of **35** person-days were spent.

## 2 Scope

Our review focused on the commit hash

**f756f44a8d6a328cd545335e46e7128939db88c4**. The list of files in scope can be found in the [Appendix](#). The auditor focused specifically on the changes and new features introduced with version 2 of the protocol.

### 2.1 Mitigations. Phase 1.

AAVE team provided a set of code changes resulting from multiple audits and their internal review. The resulting commit hash is

**60428846fead90d859bded4f1b8c7a4b95f15b4c**. Major changes include batching of flash loans, removing `repayWithCollateral` and `swapLiquidity` functionality, optimizing proxy behavior, minor fixes, gas optimization

changes, refactoring. A total of **2** person-days were spent to review them.



Some minor issues were found, but they were fixed in the next mitigations phase.

## 2.2 Mitigations. Phase 2.

After the first stage of mitigations, AAVE team provided final code changes. The final commit hash is

**750920303e33b66bc29862ea3b85206dda9ce786**. Most of the changes are refactoring, minor fixes, adding minor features, more gas optimization. Also, [issue 5.5](#) was addressed. A total of **2** person-days were spent to review the final changes.

## 2.3 Objectives

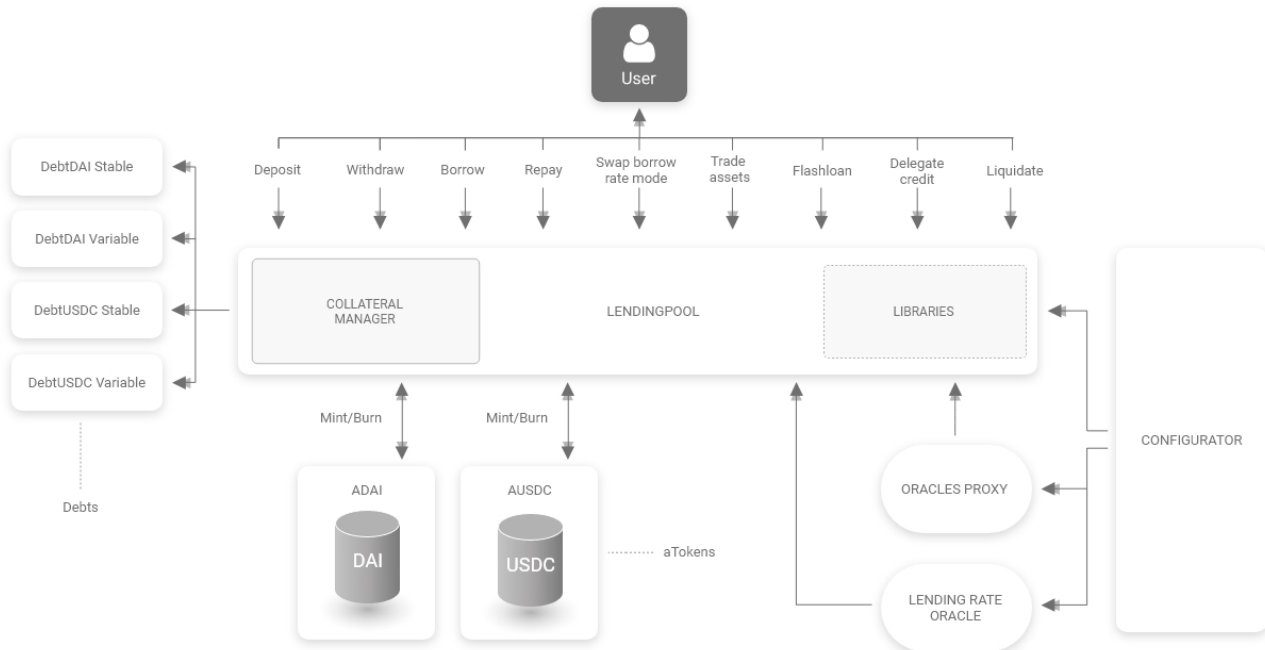
Together with the Aave team, we identified the following priorities for our review:

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](#).
3. Verify that the changes to smart contract architecture introduced in v2 did not introduce unexpected behavior or bugs.
4. Verify that the newly introduced features such as debt tokenization, collateral swap and flash loans v2 do not enable new attack vectors.

# 3 System Overview

Aave v2 retains most of the functionality of v1 while simplifying the protocol architecture and adding several new features. Code complexity has been reduced, reducing the gas footprint of all the actions by 15-20%.





Other important changes and new features include:

- **LendingPool** serves as the single point of entry for users while **LendingPoolCore** and **LendingPoolDataProvider** have been replaced with libraries.
- Funds that were previously stored in the **LendingPoolCore** contract are now stored in each specific aToken.
- Debt is now represented by tokens instead of internal accounting within the contracts. This simplifies internal accounting and allows users to take stable and variable rate loans of the same asset concurrently.
- Users are now able to swap out their collateral natively within the protocol, without the need to repay their loans.
- Aave v2 removes the reentrancy guard on flash loan which allows users to use flash loans with other features of Aave.

## 4 Security Specification

This 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.



**Actors**

The relevant actors are listed below with their respective abilities:

- Everyone
  - Can deposit or withdraw the liquidity.
  - Can borrow or repay debt.
  - Can take a flash loan.
  - Can liquidate an undercollateralized position.
  - Can swap liquidity or repay with collateral.
- AAVEAdmin
  - Manages reserves: adding, removing, freezing, enabling/disabling stable rate, allowing reserve to be used as collateral, allowing borrowing.
  - Configures reserves: sets ltv, reserve factor, liquidation threshold, liquidation bonus, reserve decimals, interest rate strategy.
  - Updates A/debt token implementation.
  - Can pause/unpause pool.

## 4.2 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:

- The pool uses the oracle to determine the price of the assets. If the oracle gets malicious, funds can be stolen.
- Any user can use the system without any whitelisting.
- The protocol is managed by AAVE Governance DAO.

## 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 Attacker can abuse `swapLiquidity` function to drain users' funds

**Medium** **✓ Fixed**

### Resolution

Solved by removing `swapLiquidity` functionality.

### Description

The `swapLiquidity` function allows liquidity providers to atomically swap their collateral. The function takes a `receiverAddress` argument that normally points to an `ISwapAdapter` implementation trusted by the user.

**code/contracts/lendingpool/LendingPoolCollateralManager.sol:L490-L517**



```

vars.fromReserveAToken.burn(
    msg.sender,
    receiverAddress,
    amountToSwap,
    fromReserve.liquidityIndex
);
// Notifies the receiver to proceed, sending as param the underlying already
ISwapAdapter(receiverAddress).executeOperation(
    fromAsset,
    toAsset,
    amountToSwap,
    address(this),
    params
);

vars.amountToReceive = IERC20(toAsset).balanceOf(receiverAddress);
if (vars.amountToReceive != 0) {
    IERC20(toAsset).transferFrom(
        receiverAddress,
        address(vars.toReserveAToken),
        vars.amountToReceive
    );

    if (vars.toReserveAToken.balanceOf(msg.sender) == 0) {
        _usersConfig[msg.sender].setUsingAsCollateral(toReserve.id, true);
    }

    vars.toReserveAToken.mint(msg.sender, vars.amountToReceive, toReserve.liquidityIndex);
}

```

However, since an attacker can pass any address as the `receiverAddress`, they can arbitrarily transfer funds from other contracts that have given allowances to the `LendingPool` contract (for example, another `ISwapAdapter`).

The `amountToSwap` is defined by the caller and can be very small. The attacker gets the difference between `IERC20(toAsset).balanceOf(receiverAddress)` value of `toAsset` and the `amountToSwap` of `fromToken`.

## Remediation

Ensure that no funds can be stolen from contracts that have granted allowances to the `LendingPool` contract.



medium

## Griefing attack by taking flash loan on behalf of user

## Description

When taking a flash loan from the protocol, the arbitrary `receiverAddress` address can be passed as the argument:

**code/contracts/lendingpool/LendingPool.sol:L547-L554**

```
function flashLoan(  
    address receiverAddress,  
    address asset,  
    uint256 amount,  
    uint256 mode,  
    bytes calldata params,  
    uint16 referralCode  
) external override {
```

That may allow anyone to execute a flash loan on behalf of other users. In order to make that attack, the `receiverAddress` should give the allowance to the `LendingPool` contract to make a transfer for the amount of `currentAmountPlusPremium`.

## Example

If someone is giving the allowance to the `LendingPool` contract to make a deposit, the attacker can execute a flash loan on behalf of that user, forcing the user to pay fees from the flash loan. That will also prevent the victim from making a successful deposit transaction.

## Remediation

Make sure that only the user can take a flash loan.

## 5.3 Interest rates are updated incorrectly Medium

### Resolution

This issue was independently discovered by the Aave developers and had already been fixed by the end of the audit.



The function `updateInterestRates()` updates the borrow rates of a reserve. Since the rates depend on the available liquidity they must be recalculated each time liquidity changes. The function takes the amount of liquidity added or removed as the input and is called ahead of minting or burning ATokens. However, in `LendingPoolCollateralManager` an interest rate update is performed *after* aTokens have been burned, resulting in an incorrect interest rate.

#### **code/contracts/lendingpool/LendingPoolCollateralManager.sol:L377-L382**

```
vars.collateralAtoken.burn(  
    user,  
    receiver,  
    vars.maxCollateralToLiquidate,  
    collateralReserve.liquidityIndex  
);
```

#### **code/contracts/lendingpool/LendingPoolCollateralManager.sol:L427-L433**

```
//updating collateral reserve  
collateralReserve.updateInterestRates(  
    collateral,  
    address(vars.collateralAtoken),  
    0,  
    vars.maxCollateralToLiquidate  
);
```

### **Recommendation**

Update interest rates before calling `collateralAtoken.burn()`.

## **5.4 Unhandled return values of transfer and transferFrom**

Medium

### **Resolution**

`safeTransferFrom` is now used instead of `transferFrom` in all locations.



20 implementations are not always consistent. Some implementations of `transfer` and `transferFrom` could return 'false' on failure instead of reverting. It



is safer to wrap such calls into `require()` statements to these failures. Unsafe `transferFrom` calls were found in the following locations:

### code/contracts/lendingpool/LendingPool.sol:L578

```
IERC20(asset).transferFrom(receiverAddress, vars.aTokenAddress, vars.amountToReceive);
```

### code/contracts/lendingpool/LendingPoolCollateralManager.sol:L407

```
IERC20(principal).transferFrom(receiver, vars.principalAToken, vars.actualAmount);
```

### code/contracts/lendingpool/LendingPoolCollateralManager.sol:L507-L511

```
IERC20(toAsset).transferFrom(
    receiverAddress,
    address(vars.toReserveAToken),
    vars.amountToReceive
);
```

## Recommendation

Check the return value and revert on `0` / `false` or use OpenZeppelin's `SafeERC20` wrapper functions.

## 5.5 Re-entrancy attacks with ERC-777 Minor

### Resolution

The issue was partially mitigated in `deposit` function by minting AToken before the transfer of the deposit token.

## Description

Some tokens may allow users to perform re-entrancy while calling the `transferFrom` function. For example, it would be possible for an attacker to



“borrow” a large amount of ERC-777 tokens from the lending pool by re-entering the `deposit` function from within `transferFrom`.

## code/contracts/lendingpool/LendingPool.sol:L91-L118

```
function deposit(
    address asset,
    uint256 amount,
    address onBehalfOf,
    uint16 referralCode
) external override {
    _whenNotPaused();
    ReserveLogic.ReserveData storage reserve = _reserves[asset];

    ValidationLogic.validateDeposit(reserve, amount);

    address aToken = reserve.aTokenAddress;

    reserve.updateState();
    reserve.updateInterestRates(asset, aToken, amount, 0);

    bool isFirstDeposit = IAToken(aToken).balanceOf(onBehalfOf) == 0;
    if (isFirstDeposit) {
        _usersConfig[onBehalfOf].setUsingAsCollateral(reserve.id, true);
    }

    IAToken(aToken).mint(onBehalfOf, amount, reserve.liquidityIndex);

    //transfer to the aToken contract
    IERC20(asset).safeTransferFrom(msg.sender, aToken, amount);

    emit Deposit(asset, msg.sender, onBehalfOf, amount, referralCode);
}
```

Because the `safeTransferFrom` call is happening at the end of the `deposit` function, the deposit will be fully processed before the tokens are actually transferred.

So at the beginning of the transfer, the attacker can re-enter the call to withdraw their deposit. The withdrawal will succeed even though the attacker’s tokens have not yet been transferred to the lending pool. Essentially, the attacker is granted a flash-loan but without paying fees.



Additionally, after these calls, interest rates will be skewed because interest rate update relies on the actual current balance.

## Remediation

Do not whitelist ERC-777 or other re-entrable tokens to prevent this kind of attack.

### 5.6 Potential manipulation of stable interest rates using flash loans Minor


#### Resolution

This type of manipulation is difficult to prevent completely especially when flash loans are available. In practice however, attacks are mitigated by the following factors:

1. Liquidity providers attempting to increase users' stable rates would have to pay a high flash loan premium. Users could also immediately swap to variable interest meaning that the attack could result in a net loss for the LP. In practice, it is likely that this makes the attack economically unfeasible.
2. Under normal conditions, users would only gain a relatively small advantage by lowering their stable rate due to the design of the stable rate curve. If a user attempted to manipulate their stable rate during a liquidity crisis, Aave could immediately rebalance them and bring the rate back to normal.

Flash loans allow users to borrow large amounts of liquidity from the protocol. It is possible to adjust the stable rate up or down by momentarily removing or adding large amounts of liquidity to reserves.

#### LPs increasing the interest rate of borrowers

The function `rebalanceStableBorrowRate()` increases the stable interest rate of a user if the current liquidity rate is higher than the user's stable rate. A liquidity provider could trigger an artificial "liquidity crisis" in a reserve and increase  stable interest rates of borrowers by atomically performing the following steps:

1. Take a flash loan to take a large number of tokens from a reserve
2. Re-balance the stable rate of the emptied reserves' borrowers
3. Repay the flash loan (plus premium)
4. Withdraw the collateral and repay the flash loan

Individual borrowers would then have to switch to the variable rate to return to a lower interest rate.

### User borrowing at an artificially lowered interest rate

Users wanting to borrow funds could attempt to get a lower interest rate by temporarily adding liquidity to a reserve (which could e.g. be flash borrowed from a different protocol). While there's a check that prevents users from borrowing an asset while also adding a higher amount of the same asset as collateral, this can be bypassed rather easily by depositing the collateral from a different address (via smart contracts). Aave would then have to rebalance the user to restore an appropriate interest rate.

In practice, users would gain only a relatively small advantage here due to the design of the stable rate curve.

### Recommendation

This type of manipulation is difficult to prevent especially when flash loans are available. The safest option to prevent the first variant would be to restrict access to `rebalanceStableBorrowRate()` to admins. In any case, Aave should monitor the protocol at all times to make sure that interest rates are being rebalanced to sane values.

## 5.7 Code quality could be improved Minor

Some minor code quality improvements are recommended to improve readability.

**Explicitly set the visibility for of variables:**

**code/contracts/tokenization/StableDebtToken.sol:L23-L24**

```
mapping(address => uint40) _timestamps;  
uint40 _totalSupplyTimestamp;
```



## code/contracts/configuration/LendingPoolAddressesProviderRegistry.sol: L17-L18

```
mapping(address => uint256) addressesProviders;  
address[] addressesProvidersList;
```

### 5.8 Attacker can front-run delegator when changing allowance Minor

Users can grant allowances to borrow debt assets to other users using the `delegateAllowance` function. Similar to the classical ERC20 `approve` attack, it is possible for a malicious user to front-run the delegator when they attempt to change the allowance and borrow the sum of the old and new values.

Example scenario:

1. Bob creates an allowance of 100 DAI for Malice:

```
delegateBorrowAllowance(DAI, Malice, 100)
```

2. Later, Bob attempts to lower the allowance to 90:

```
delegateBorrowAllowance(DAI, Malice, 90)
```

3. Malice borrows a total of 190 DAI by first frontrunning Bob's second transaction borrowing 100 DAI and then borrowing another 90 DAI after Bob's transaction was mined.

### Recommendation

A commonly used way of preventing this attack is using `increaseAllowance()` and `decreaseAllowance()` functions specifically for increasing and decreasing allowances.

### 5.9 Description of flash loan function is inconsistent with code Minor

The function `flashLoan` in `LendingPool.sol` takes an argument `mode` that specifies the interest rate mode. If the mode is `ReserveLogic.InterestRateMode.NONE` the function call is treated as a flash loan, if not a normal borrow is executed.

However, inline comments in the function describe the behaviour as "If the transfer didn't succeed, the receiver either didn't return the funds, or didn't



approve the transfer". It is unclear how this relates to the actual code or why it is possible to specify a mode in the first place.

# Appendix 1 - MythX Scan

MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at [mythx.io](https://mythx.io).

## A.1.1 Auditor Comments

The MythX analysis did not return any critical or high risk findings. Some notes on the results:

1. A single integer overflow was reported. However, this turned out to be an intentional overflow produced by solc: With optimisation enabled, `variable - 1` compiles to `ADD var, MAX_INT`.
2. All assertion violations were caused by accessing non-existent elements in arrays or structs. This is expected behaviour.
3. MythX reports some issues where state is accessed following external calls or multiple calls are executed within the same transaction. The reported issues were double-checked by the auditors and found not to cause any exploitable conditions.
4. Some minor best practice violations such as lack of explicit visibility specifiers on state variables were reported. We included them in code quality recommendations in the previous section of this report.

## A.1.2 MythX Output

Below is the result of the MythX scan. The raw MythX results were reviewed by an auditor to eliminate false positives. [Download the full PDF report here](#).

Title	Method	SWC	Passed
-------	--------	-----	--------



Title	Method	SWC	Passed
Solidity assert violation	Symbolic analysis, fuzzing (bytecode)	Symbolic analysis, fuzzing (bytecode)	X
MythX assertion violation (AssertionFailed event)	Symbolic analysis, fuzzing (bytecode)	Symbolic analysis, fuzzing (bytecode)	✓
Integer overflow in arithmetic operation	Symbolic analysis, fuzzing (bytecode)	Symbolic analysis, fuzzing (bytecode)	✓
Integer underflow in arithmetic operation	Symbolic analysis, fuzzing (bytecode)	Symbolic analysis, fuzzing (bytecode)	✓
Caller can redirect execution to arbitrary locations	Symbolic analysis, fuzzing (bytecode)	Symbolic analysis, fuzzing (bytecode)	✓
Caller can write to arbitrary storage locations	Symbolic analysis, fuzzing (bytecode)	Symbolic analysis, fuzzing (bytecode)	✓
Dangerous use of uninitialized storage variables	Solidity code analysis	Solidity code analysis	✓
Use of "tx.origin" as a part of authorization control	Solidity code analysis	Solidity code analysis	✓



Title	Method	SWC	Passed
Any sender can trigger SELFDESTRUCT	Symbolic analysis (bytecode)	Symbolic analysis (bytecode)	✓
Any sender can withdraw ETH from the contract account	Symbolic analysis (bytecode)	Symbolic analysis (bytecode)	✓
Delegatecall to a user-supplied address	Symbolic analysis (bytecode)	Symbolic analysis (bytecode)	✓
Call to a user-supplied address	Symbolic analysis (bytecode)	Symbolic analysis (bytecode)	✓
Unchecked return value from external call	Solidity code analysis	Solidity code analysis	✓
Block timestamp influences a control flow decision	Taint analysis (bytecode)	Taint analysis (bytecode)	✓
Environment variables influence a control flow decisions	Taint analysis (bytecode)	Taint analysis (bytecode)	✓
Loop over unbounded data structure	Solidity code analysis	Solidity code analysis	X
Implicit loop over unbounded data structure	Solidity code analysis	Solidity code analysis	✓
Usage of "continue" in "do-while"	Solidity code analysis	Solidity code analysis	✓
Multiple calls are executed in the same transaction	Static analysis (bytecode)	Static analysis (bytecode)	X





Title	Method	SWC	Passed
Persistent state read following external call	Static analysis, fuzzing (bytecode)	Static analysis, fuzzing (bytecode)	X
Persistent state write following external call	Static analysis, fuzzing (bytecode)	Static analysis, fuzzing (bytecode)	X
Account state accessed after call to user-defined address	Symbolic analysis (bytecode)	Symbolic analysis (bytecode)	✓
Return value of an external call is not checked	Static analysis (bytecode)	Static analysis (bytecode)	✓
Potential weak source of randomness	Solidity code analysis	Solidity code analysis	✓
Requirement violation	Fuzzing (bytecode)	Fuzzing (bytecode)	✓
Call with hardcoded gas amount	Solidity code analysis	Solidity code analysis	✓
Incorrect ERC20 implementation	Solidity code analysis	Solidity code analysis	✓
Outdated compiler version	Solidity code analysis	Solidity code analysis	✓
No or floating compiler version set	Solidity code analysis	Solidity code analysis	X
Use of right-to-left-override control character	Solidity code analysis	Solidity code analysis	✓
Shadowing of built-in symbol	Solidity code analysis	Solidity code analysis	✓



Title	Method	SWC	Passed
Incorrect constructor name	Solidity code analysis	Solidity code analysis	✓
State variable shadows another state variable	Solidity code analysis	Solidity code analysis	X
Local variable shadows a state variable	Solidity code analysis	Solidity code analysis	✓
Function parameter shadows a state variable	Solidity code analysis	Solidity code analysis	✓
Named return value shadows a state variable	Solidity code analysis	Solidity code analysis	✓
Unary operation without effect	Solidity code analysis	Solidity code analysis	✓
Unary operation directly after assignment	Solidity code analysis	Solidity code analysis	✓
Unused state variable	Solidity code analysis	Solidity code analysis	✓
Unused local variable	Solidity code analysis	Solidity code analysis	✓
Function visibility is not set	Solidity code analysis	Solidity code analysis	✓
State variable visibility is not set	Solidity code analysis	Solidity code analysis	✓
Use of deprecated functions: <code>callcode()</code> , <code>sha3()</code> , ...	Solidity code analysis	Solidity code analysis	✓



Title	Method	SWC	Passed
Use of deprecated global variables (msg.gas, ...)	Solidity code analysis	Solidity code analysis	✓
Use of deprecated keywords (throw, var)	Solidity code analysis	Solidity code analysis	✓
Incorrect function state mutability	Solidity code analysis	Solidity code analysis	✓

## Appendix 2 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
File Name	SHA-1 Hash
-----	-----
contracts/configuration/LendingPoolAddressesProvider.sol	fc43ca392cefc784581336ccd50e6ab396d2e7b0
contracts/configuration/LendingPoolAddressesProviderRegistry.sol	5c762c93ad1601cf65b1b649e7e3dcaff67398b9
contracts/flashloan/base/FlashLoanReceiverBase.sol	b0c1d88eb7c62f1365ce8fc0d6570dcc12f1704a
contracts/lendingpool/DefaultReserveInterestRateStrategy.sol	5ea00bbb8cc5a2e23c9fbc8909fea7f72c359c27
contracts/lendingpool/LendingPool.sol	0e244177e35588087ec68aab10e3d6dcfba8fd3f
contracts/lendingpool/LendingPoolCollateralManager.sol	1bb03f18ec8695d9d576b6565e1806e2d9dfef45
contracts/lendingpool/LendingPoolConfigurator.sol	ee5a7e8e5a0c9979b6a80e6af4c773fe2932801c



File	SHA-1 hash
contracts/lendingpool/LendingPoolStorage.sol	45273cf021f4ca492ee31e4c3cb4fc505bb59113
contracts/libraries/configuration/ReserveConfiguration.sol	742681d53ec89c5f2bdb9c0f212817b7c00fe70b
contracts/libraries/configuration/UserConfiguration.sol	d83b66f4a86d6fd9d34e263d6239644aab2571fc
contracts/libraries/helpers/Errors.sol	2e3ef63a81e628ed80c98d7e8b444c013bd0445c
contracts/libraries/helpers/Helpers.sol	8e75dfb98a8f09a93f08c9c713ce64baa656cd77
contracts/libraries/logic/GenericLogic.sol	24bcec756ac5c7eee8efe9e15ac9c559fe9b09f3
contracts/libraries/logic/ReserveLogic.sol	c5378e263428b7517ed3160314753f4f894ee667
contracts/libraries/logic/ValidationLogic.sol	d1a8d5c6bcdfe18c4a5a21f20c5727661a16cf3d
contracts/libraries/math/MathUtils.sol	87d586353c9f343d2a4fb05f27ed82d9887331a5
contracts/libraries/math/PercentageMath.sol	362a7a01af3802c33c58d9450e33737592d14b38
contracts/libraries/math/SafeMath.sol	b6e2b436c949022d872611b1536e7ef66c4b157a
contracts/libraries/math/WadRayMath.sol	7bd3c5d5e1fa8decce70a8f7eb93ccff71270a82
contracts/libraries/openzeppelin-upgradeability/AdminUpgradeabilityProxy.sol	7d94d6437dd2ca13d1e122f15e10445ef0efc08a
contracts/libraries/openzeppelin-upgradeability/BaseAdminUpgradeabilityProxy.sol	56b10eef0d671c88b9508beca598f4a82a65d90f



File	SHA-1 hash
contracts/libraries/openzeppelin-upgradeability/BaseUpgradeabilityProxy.sol	9a4359f80fab985949ef6709e95ca73101cdfdda
contracts/libraries/openzeppelin-upgradeability/Initializable.sol	d222af72b8e085d77a713824e9dd4224731b0d55
contracts/libraries/openzeppelin-upgradeability/InitializableAdminUpgradeabilityProxy.sol	7a48b74de6eede6503c72ce131b1571e2cbd1beb
contracts/libraries/openzeppelin-upgradeability/InitializableUpgradeabilityProxy.sol	b50e5dde172728c8c44af9688a968cd79b0fd4c8
contracts/libraries/openzeppelin-upgradeability/Proxy.sol	d39b7a982109fb4d2ee216b0b1d8aeb0434b05c5
contracts/libraries/openzeppelin-upgradeability/UpgradeabilityProxy.sol	916c5c35658470a55a010eaa75326040dd574c1f
contracts/libraries/openzeppelin-upgradeability/VersionedInitializable.sol	5edbf435533e89b992dd7bfd7ce073953809cf08
contracts/misc/AaveProtocolTestHelpers.sol	8451fd59bc38e6c4dc3966bb2f3d872d20391f5f
contracts/misc/Address.sol	1de70a4c842e728cb55f72d267f2266248daba12
contracts/misc/ChainlinkProxyPriceProvider.sol	9e8107c03ed4a55e35281140b0c0241c0e9730d4
contracts/misc/Context.sol	4018ef02a339207d889bb298585c40b9e1a07cd1
contracts/misc/IERC20DetailedBytes.sol	5fc3557ea0f186849d0e7a5fdfd150bd9e0f8984
contracts/misc/SafeERC20.sol	126a50b6d07d10cca9b9713247d1e9f333dfae55



File	SHA-1 hash
contracts/misc/WalletBalanceProvider.sol	a11be6828e99c92be99b74859856ef4ffc8b1cb6
contracts/tokenization/AToken.sol	9def2fc2d5870f48530efa0f0a4cd007997eb1f
contracts/tokenization/IncentivizedERC20.sol	36048d0331176035e889e8e2616e48cc1acb6c0f
contracts/tokenization/StableDebtToken.sol	ff4b9475461bf03b108046d165c8e9e776ebe92c
contracts/tokenization/VariableDebtToken.sol	e5cc9d75a01081942810697bdcb6cff46f19a87f
contracts/tokenization/base/DebtTokenBase.sol	6c1310875fbfae2fbc92a2f5f10fee29c6436e03

## Appendix 3 - 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.3.1 Tests Suite

Below is the output generated by running the test suite:

#### AToken: Modifiers

- ✓ Tries to invoke mint not being the LendingPool
- ✓ Tries to invoke burn not being the LendingPool
- ✓ Tries to invoke transferOnLiquidation not being the LendingPool
- ✓ Tries to invoke transferUnderlyingTo not being the LendingPool

#### AToken: Permit

- ✓ Checks the domain separator
- ✓ Get aDAI for tests (87ms)
- ✓ Reverts submitting a permit with 0 expiration (48ms)
- ✓ Submits a permit with maximum expiration length (56ms)
- ✓ Cancels the previous permit (67ms)



- ✓ Tries to submit a permit with invalid nonce
- ✓ Tries to submit a permit with invalid expiration (previous to the cur)
- ✓ Tries to submit a permit with invalid signature
- ✓ Tries to submit a permit with invalid owner

#### AToken: Transfer

- ✓ User 0 deposits 1000 DAI, transfers to user 1 (166ms)
- ✓ User 0 deposits 1 WETH and user 1 tries to borrow, but the aTokens rec
- ✓ User 1 sets the DAI as collateral and borrows, tries to transfer every

#### LendingPool SwapDeposit function

- ✓ Should not allow to swap if from equal to
- ✓ Should not allow to swap if from or to reserves are not active (212ms)
- ✓ Deposits WETH into the reserve (189ms)
- ✓ User tries to swap more then he can, revert expected (66ms)
- ✓ User tries to swap more then available on the reserve (305ms)
- ✓ User tries to swap correct amount (753ms)
- ✓ User tries to drop HF below one (396ms)
- ✓ Should set usage as collateral to false if no leftovers after swap (75
- ✓ Should not allow to swap if to reserve are freezed (76ms)

#### LendingPoolConfigurator

- ✓ Deactivates the ETH reserve (69ms)
- ✓ Rectivates the ETH reserve (39ms)
- ✓ Check the onlyAaveAdmin on deactivateReserve
- ✓ Check the onlyAaveAdmin on activateReserve
- ✓ Freezes the ETH reserve (40ms)
- ✓ Unfreezes the ETH reserve (44ms)
- ✓ Check the onlyAaveAdmin on freezeReserve
- ✓ Check the onlyAaveAdmin on unfreezeReserve
- ✓ Deactivates the ETH reserve for borrowing (39ms)
- ✓ Activates the ETH reserve for borrowing (272ms)
- ✓ Check the onlyAaveAdmin on disableBorrowingOnReserve
- ✓ Check the onlyAaveAdmin on enableBorrowingOnReserve
- ✓ Deactivates the ETH reserve as collateral (45ms)
- ✓ Activates the ETH reserve as collateral (43ms)
- ✓ Check the onlyAaveAdmin on disableReserveAsCollateral
- ✓ Check the onlyAaveAdmin on enableReserveAsCollateral
- ✓ Disable stable borrow rate on the ETH reserve (41ms)
- ✓ Enables stable borrow rate on the ETH reserve (41ms)
- ✓ Check the onlyAaveAdmin on disableReserveStableRate
- ✓ Check the onlyAaveAdmin on enableReserveStableRate
- ✓ Changes LTV of the reserve (46ms)
- ✓ Check the onlyAaveAdmin on setLtv
- ✓ Changes the reserve factor of the reserve (42ms)
- ✓ Check the onlyLendingPoolManager on setReserveFactor
- ✓ Changes liquidation threshold of the reserve (48ms)
- ✓ Check the onlyAaveAdmin on setLiquidationThreshold
- ✓ Changes liquidation bonus of the reserve (42ms)
- ✓ Check the onlyAaveAdmin on setLiquidationBonus





- ✓ Check the onlyAaveAdmin on setLiquidationBonus
- ✓ Check the onlyAaveAdmin on setReserveDecimals
- ✓ Check the onlyAaveAdmin on setLiquidationBonus
- ✓ Reverts when trying to disable the DAI reserve with liquidity on it (

#### LendingPool. repayWithCollateral()

- ✓ It's not possible to repayWithCollateral() on a non-active collateral
- ✓ User 1 provides some liquidity for others to borrow (315ms)
- ✓ User 2 deposit WETH and borrows DAI at Variable (217ms)
- ✓ It is not possible to do reentrancy on repayWithCollateral() (170ms)
- ✓ User 2 tries to repay his DAI Variable loan using his WETH collateral
- ✓ User 3 deposits WETH and borrows USDC at Variable (228ms)
- ✓ User 3 repays completely his USDC loan by swapping his WETH collateral
- ✓ Revert expected. User 3 tries to repay with his collateral a currency
- ✓ User 3 tries to repay with his collateral all his variable debt and pa
- ✓ User 4 tries to repay a bigger amount that what can be swapped of a pa
- ✓ User 5 deposits WETH and DAI, then borrows USDC at Variable, then disa
- ✓ User 5 tries to repay his USDC loan by swapping his WETH collateral, s

#### LendingPool. repayWithCollateral() with liquidator

- ✓ User 1 provides some liquidity for others to borrow (354ms)
- ✓ User 5 liquidate User 3 collateral, all his variable debt and part of
- ✓ User 3 deposits WETH and borrows USDC at Variable (377ms)
- ✓ User 5 liquidates half the USDC loan of User 3 by swapping his WETH co
- ✓ Revert expected. User 5 tries to liquidate an User 3 collateral a curi
- ✓ User 5 liquidates all the USDC loan of User 3 by swapping his WETH co
- ✓ User 2 deposit WETH and borrows DAI at Variable (349ms)
- ✓ It is not possible to do reentrancy on repayWithCollateral() (509ms)
- ✓ User 5 tries to liquidate User 2 DAI Variable loan using his WETH co
- ✓ User 5 liquidates User 2 DAI Variable loan using his WETH collateral,
- ✓ User 2 tries to repay remaining DAI Variable loan using his WETH colla
- ✓ Liquidator tries to repay 4 user a bigger amount that what can be swa
- ✓ User 5 deposits WETH and DAI, then borrows USDC at Variable, then disa
- ✓ Liquidator tries to liquidates User 5 USDC loan by swapping his WETH c

#### LendingPool FlashLoan function

- ✓ Deposits WETH into the reserve (116ms)
- ✓ Takes WETH flashloan with mode = 0, returns the funds correctly (135ms)
- ✓ Takes an ETH flashloan with mode = 0 as big as the available liquidity
- ✓ Takes WETH flashloan, does not return the funds with mode = 0. (revert
- ✓ Takes a WETH flashloan with an invalid mode. (revert expected)
- ✓ Caller deposits 1000 DAI as collateral, Takes WETH flashloan with mode
- ✓ tries to take a very small flashloan, which would result in 0 fees (re
- ✓ tries to take a flashloan that is bigger than the available liquidity
- ✓ tries to take a flashloan using a non contract address as receiver (re
- ✓ Deposits USDC into the reserve (136ms)
- ✓ Takes out a 500 USDC flashloan, returns the funds correctly (224ms)
- ✓ Takes out a 500 USDC flashloan with mode = 0, does not return the func
- ✓ Caller deposits 5 WETH as collateral, Takes a USDC flashloan with mode
- ✓ Caller deposits 1000 DAI as collateral, Takes a WETH flashloan with m
- ✓ Caller takes a WETH flashloan with mode = 1 (200ms)





✓ Caller takes a WETH flashloan with mode = 1 (200ms)

#### LendingPoolAddressesProvider

✓ Test the accessibility of the LendingPoolAddressesProvider (258ms)

#### LendingPool liquidation - liquidator receiving aToken

- ✓ LIQUIDATION - Deposits WETH, borrows DAI/Check liquidation fails because of health factor (100ms)
- ✓ LIQUIDATION - Drop the health factor below 1 (89ms)
- ✓ LIQUIDATION - Tries to liquidate a different currency than the loan pool (100ms)
- ✓ LIQUIDATION - Tries to liquidate a different collateral than the borrowed asset (100ms)
- ✓ LIQUIDATION - Liquidates the borrow (761ms)
- ✓ User 3 deposits 1000 USDC, user 4 1 WETH, user 4 borrows - drops HF, liquidates the borrow (100ms)

#### LendingPool liquidation - liquidator receiving the underlying asset

- ✓ It's not possible to liquidate on a non-active collateral or a non active borrow (100ms)
- ✓ LIQUIDATION - Deposits WETH, borrows DAI (469ms)
- ✓ LIQUIDATION - Drop the health factor below 1 (77ms)
- ✓ LIQUIDATION - Liquidates the borrow (602ms)
- ✓ User 3 deposits 1000 USDC, user 4 1 WETH, user 4 borrows - drops HF, liquidates the borrow (100ms)
- ✓ User 4 deposits 1000 LEND - drops HF, liquidates the LEND, which results in a flashloan (100ms)

#### Pausable Pool

- ✓ User 0 deposits 1000 DAI. Configurator pauses pool. Transfers to user 0 (100ms)
- ✓ Deposit (82ms)
- ✓ Withdraw (149ms)
- ✓ DelegateBorrowAllowance (51ms)
- ✓ Borrow (47ms)
- ✓ Swap liquidity (51ms)
- ✓ Repay (47ms)
- ✓ Repay with collateral (48ms)
- ✓ Flash loan (56ms)
- ✓ Liquidation call (501ms)
- ✓ SwapBorrowRateMode (335ms)
- ✓ RebalanceStableBorrowRate (45ms)
- ✓ setUserUseReserveAsCollateral (131ms)

#### LendingPool: Borrow negatives (reverts)

- ✓ User 0 deposits 1000 DAI, user 1 deposits 1 WETH as collateral and tries to borrow 1000 DAI (revert expected) (100ms)
- ✓ User 0 deposits 1000 DAI, user 1 deposits 1 WETH as collateral and tries to borrow 1000 DAI (revert expected) (100ms)

#### LendingPool: Borrow/repay (stable rate)

- ✓ User 0 deposits 1000 DAI, user 1 deposits 1 WETH as collateral and borrows 1000 DAI (100ms)
- ✓ User 1 tries to borrow the rest of the DAI liquidity (revert expected) (100ms)
- ✓ User 1 repays the half of the DAI borrow after one year (490ms)
- ✓ User 1 repays the rest of the DAI borrow after one year (474ms)
- ✓ User 0 withdraws the deposited DAI plus interest (376ms)
- ✓ User 1 deposits 1000 DAI, user 2 tries to borrow 1000 DAI at a stable rate (100ms)
- ✓ User 0 deposits 1000 DAI, user 1,2,3,4 deposit 1 WETH each and borrow 1000 DAI (100ms)
- ✓ User 0 deposits 1000 DAI, user 1 deposits 2 WETH and borrow 100 DAI at a stable rate (100ms)



## LendingPool: Borrow/repay (variable rate)

- ✓ User 2 deposits 1 DAI to account for rounding errors (335ms)
- ✓ User 0 deposits 1000 DAI, user 1 deposits 1 WETH as collateral and bor
- ✓ User 1 tries to borrow the rest of the DAI liquidity (revert expected)
- ✓ User 1 tries to repay 0 DAI (revert expected) (156ms)
- ✓ User 1 repays a small amount of DAI, enough to cover a small part of
- ✓ User 1 repays the DAI borrow after one year (470ms)
- ✓ User 0 withdraws the deposited DAI plus interest (308ms)
- ✓ User 1 withdraws the collateral (298ms)
- ✓ User 2 deposits a small amount of WETH to account for rounding errors
- ✓ User 0 deposits 1 WETH, user 1 deposits 100 LINK as collateral and bor
- ✓ User 1 tries to repay 0 ETH (150ms)
- ✓ User 2 tries to repay everything on behalf of user 1 using uint(-1) (1
- ✓ User 3 repays a small amount of WETH on behalf of user 1 (355ms)
- ✓ User 1 repays the WETH borrow after one year (332ms)
- ✓ User 0 withdraws the deposited WETH plus interest (286ms)
- ✓ User 1 withdraws the collateral (289ms)
- ✓ User 2 deposits 1 USDC to account for rounding errors (318ms)
- ✓ User 0 deposits 1000 USDC, user 1 deposits 1 WETH as collateral and bo
- ✓ User 1 tries to borrow the rest of the USDC liquidity (revert expected
- ✓ User 1 repays the USDC borrow after one year (337ms)
- ✓ User 0 withdraws the deposited USDC plus interest (272ms)
- ✓ User 1 withdraws the collateral (275ms)
- ✓ User 1 deposits 1000 DAI, user 3 tries to borrow 1000 DAI without any
- ✓ user 3 deposits 0.1 ETH collateral to borrow 100 DAI; 0.1 ETH is not e
- ✓ user 3 withdraws the 0.1 ETH (284ms)
- ✓ User 1 deposits 1000 USDC, user 3 tries to borrow 1000 USDC without a
- ✓ user 3 deposits 0.1 ETH collateral to borrow 100 USDC; 0.1 ETH is not
- ✓ user 3 withdraws the 0.1 ETH (279ms)
- ✓ User 0 deposits 1000 DAI, user 6 deposits 2 WETH and borrow 100 DAI a

## LendingPool: credit delegation

- ✓ User 0 deposits 1000 DAI, user 0 delegates borrowing of 1 WETH on var
- ✓ User 4 trying to borrow 1 WETH stable on behalf of user 0, revert expe
- ✓ User 0 delegates borrowing of 1 WETH to user 4, user 4 borrows 3 WETH
- ✓ User 0 delegates borrowing of 1 WETH on stable to user 2, user 2 borro
- ✓ User 0 delegates borrowing of 1 WETH to user 2 with wrong borrowRateM

## LendingPool: Deposit

- ✓ User 0 Deposits 1000 DAI in an empty reserve (299ms)
- ✓ User 1 deposits 1000 DAI after user 1 (299ms)
- ✓ User 0 deposits 1000 USDC in an empty reserve (296ms)
- ✓ User 1 deposits 1000 USDC after user 0 (439ms)
- ✓ User 0 deposits 1 WETH in an empty reserve (300ms)
- ✓ User 1 deposits 1 WETH after user 0 (298ms)
- ✓ User 1 deposits 0 ETH (revert expected) (148ms)
- ✓ User 1 deposits 0 DAI (121ms)
- ✓ User 1 deposits 100 DAI on behalf of user 2, user 2 tries to borrow 0

## LendingPool: Rebalance stable rate



- ✓ User 0 tries to rebalance user 1 who has no borrows in progress (revert)
- ✓ User 0 deposits 1000 DAI, user 1 deposits 5 ETH, borrows 600 DAI at a
- ✓ User 1 borrows another 200 at stable, user 0 tries to rebalance but th
- ✓ User 1 borrows another 200 at stable, user 0 tries to rebalance but th
- ✓ User 1 borrows another 100 at stable, user 0 tries to rebalance but th
- ✓ User 2 deposits ETH and borrows the remaining DAI, causing the stable
- ✓ User 2 borrows the remaining DAI (usage ratio = 100%). User 0 rebalan

#### LendingPool: Usage as collateral

- ✓ User 0 Deposits 1000 DAI, disables DAI as collateral (526ms)
- ✓ User 1 Deposits 2 ETH, disables ETH as collateral, borrows 400 DAI (re
- ✓ User 1 enables ETH as collateral, borrows 400 DAI (517ms)
- ✓ User 1 disables ETH as collateral (revert expected) (154ms)

#### LendingPool: Swap rate mode

- ✓ User 0 tries to swap rate mode without any variable rate loan in progr
- ✓ User 0 tries to swap rate mode without any stable rate loan in progres
- ✓ User 0 deposits 1000 DAI, user 1 deposits 2 ETH as collateral, borrows
- ✓ User 1 borrows another 100 DAI, and swaps back to variable after one y

#### LendingPool: Redeem negative test cases

- ✓ Users 0 Deposits 1000 DAI and tries to redeem 0 DAI (revert expected)
- ✓ Users 0 tries to redeem 1100 DAI from the 1000 DAI deposited (revert e
- ✓ Users 1 deposits 1 WETH, borrows 100 DAI, tries to redeem the 1 WETH c

#### LendingPool: withdraw

- ✓ User 0 Deposits 1000 DAI in an empty reserve (301ms)
- ✓ User 0 withdraws half of the deposited DAI (267ms)
- ✓ User 0 withdraws remaining half of the deposited DAI (262ms)
- ✓ User 0 Deposits 1000 USDC in an empty reserve (293ms)
- ✓ User 0 withdraws half of the deposited USDC (400ms)
- ✓ User 0 withdraws remaining half of the deposited USDC (264ms)
- ✓ User 0 Deposits 1 WETH in an empty reserve (295ms)
- ✓ User 0 withdraws half of the deposited ETH (268ms)
- ✓ User 0 withdraws remaining half of the deposited ETH (260ms)
- ✓ Users 0 and 1 Deposit 1000 DAI, both withdraw (1063ms)
- ✓ Users 0 deposits 1000 DAI, user 1 Deposit 1000 USDC and 1 WETH, borrow

#### Stable debt token tests

- ✓ Tries to invoke mint not being the LendingPool
- ✓ Tries to invoke burn not being the LendingPool

#### Upgradeability

- ✓ Tries to update the DAI Atoken implementation with a different address
- ✓ Upgrades the DAI Atoken implementation (44ms)
- ✓ Tries to update the DAI Stable debt token implementation with a differ
- ✓ Upgrades the DAI stable debt token implementation (52ms)
- ✓ Tries to update the DAI variable debt token implementation with a dif
- ✓ Upgrades the DAI variable debt token implementation (51ms)




```
Variable debt token tests
  ✓ Tries to invoke mint not being the LendingPool
  ✓ Tries to invoke burn not being the LendingPool
```

```
215 passing (2m)
```

## Appendix 4 - 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 be 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  as 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

