UniLend

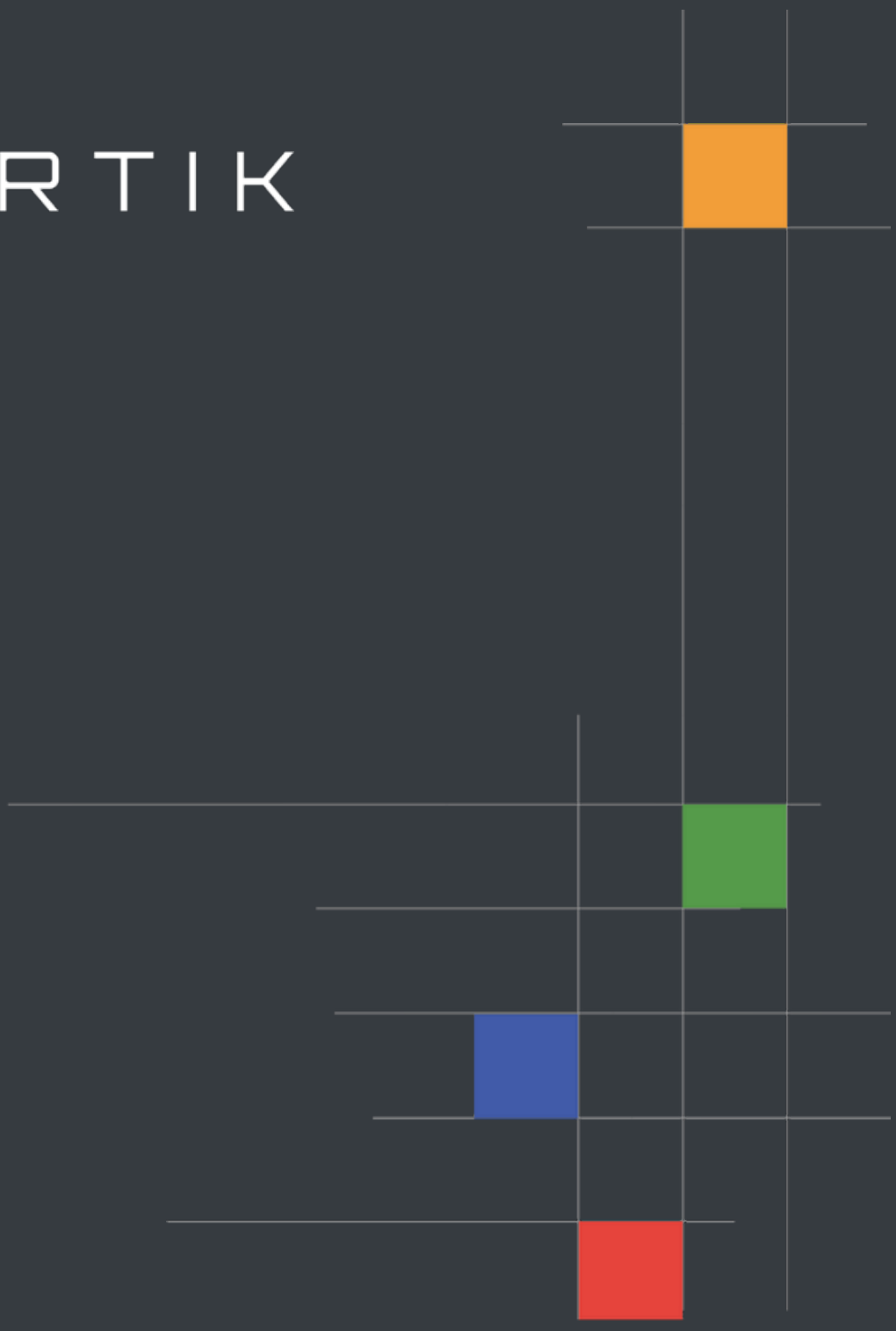
Flash Loan

Security Assessment

March 15th, 2021

Audited By:
Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Reviewed By:
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org



Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	UniLend - Flash Loan	
Description	A typical flash loan smart contract implementation	
Platform	Ethereum; Solidity, Yul	
Codebase	GitHub Repository	
Commits	1. d773f39309657d4965e383734f8b89bfaf5c7c21	2. 0e5cf20d34ea30ef26ca67f67bd2aaa2c04d36cc

Audit Summary

Delivery Date	March 15th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	March 2nd, 2021 - March 15th, 2021

Vulnerability Summary

Total Issues	15
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	6
Total Informational	9

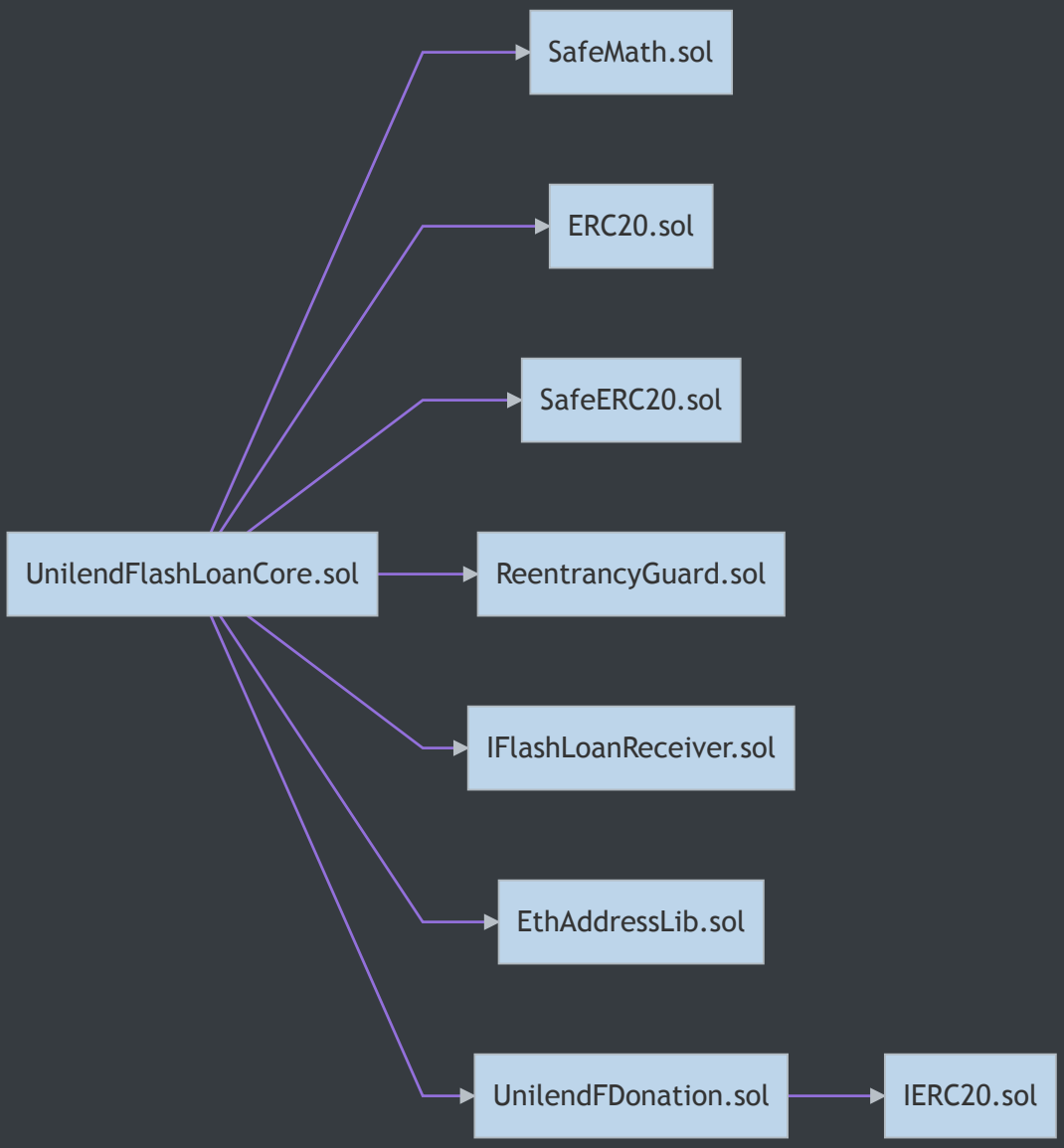
Executive Summary

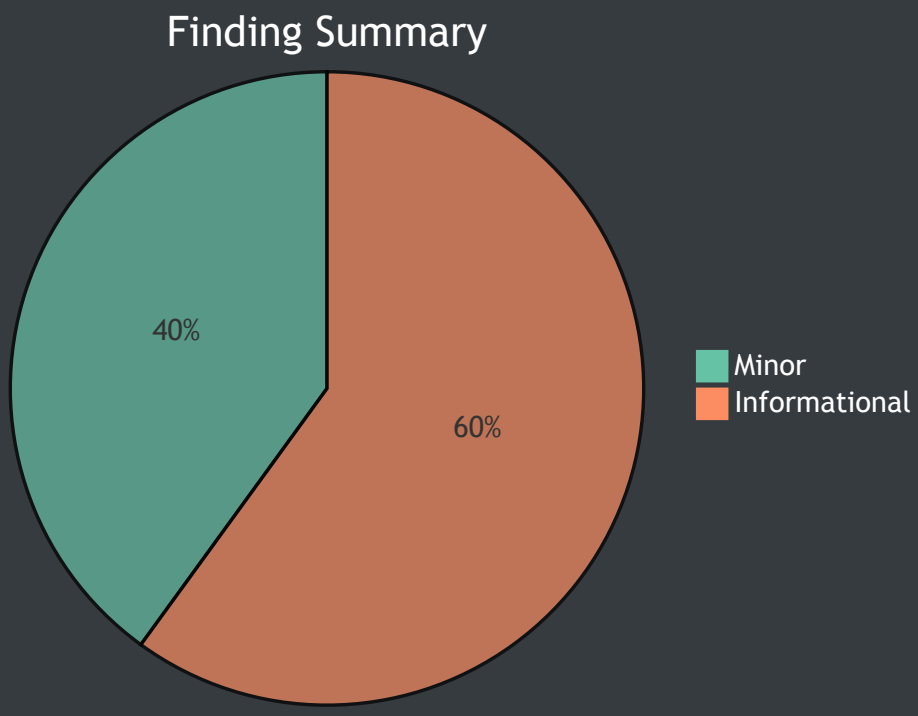
This section will represent the summary of the whole audit process once it has concluded.

Files In Scope

ID	Contract	Location
EAL	EthAddressLib.sol	contracts/EthAddressLib.sol
UFD	UnilendFDonation.sol	contracts/UnilendFDonation.sol
UFL	UnilendFlashLoanCore.sol	contracts/UnilendFlashLoanCore.sol

File Dependency Graph




Finding Summary
40%
60%
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
UFL-01M	Potential `admin`-less Contract	Volatile Code	Minor	✓
UFL-02M	Potential `distributor`-less Contract	Volatile Code	Minor	✓
UFL-03M	Inexistent Input Sanitization	Volatile Code	Minor	✓
UFL-04M	Redundant `else` Block	Gas Optimization	Informational	⟳
UFL-05M	Redundant Code Block	Gas Optimization	Informational	✓
UFL-06M	User-Defined Getters	Gas Optimization	Informational	✓
UFL-07M	Conditional Optimization	Gas Optimization	Informational	✓
UFL-08M	Potential Zero Flash Loan Fees	Volatile Code	Informational	✓

Static Analysis Findings

ID	Title	Type	Severity	Resolved
UFD-01S	Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call	Volatile Code	Minor	✓
UFD-02S	Potential Re-Entrancy	Volatile Code	Minor	✓
UFD-03S	Unlocked Compiler Version	Language Specific	Informational	✓
UFD-04S	Visibility Specifiers Missing	Language Specific	Informational	✓
UFL-01S	Usage of `transfer()` for sending Ether	Volatile Code	Minor	✓
UFL-02S	Contract Size	Language Specific	Informational	⟳
UFL-03S	Visibility Specifiers Missing	Language Specific	Informational	✓

UFL-01M: Potential `admin` -less Contract

Type	Severity	Location
Volatile Code	Minor	UnilendFlashLoanCore.sol L329-L331

Description:

The linked function can set the `admin` state variable equal to the zero address.

Recommendation:

We advise to add a `require` statement, checking the new admin address against the zero address, in case this is not intended functionality.

Alleviation:

The development team opted to consider our references and added a `require` statement to ensure that the `_admin` state variable is always non-zero.

UFL-02M: Potential `distributor` -less Contract

Type	Severity	Location
Volatile Code	Minor	UnilendFlashLoanCore.sol L337-L339

Description:

The linked function can set the `distributor` state variable equal to the zero address.

Recommendation:

We advise to add a `require` statement, checking the new distributor against the zero address, in case this is not intended functionality.

Alleviation:

The development team opted to consider our references and added a `require` statement to ensure that the `distributor` state variable is always non-zero.

UFL-03M: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	Minor	UnilendFlashLoanCore.sol L360

Description:

The linked functions fail to check the input `address` values.

Recommendation:

We advise to add `require` statements, checking against the zero address.

Alleviation:

The development team opted to consider our references and added a `require` statement to ensure that the input address is different from the zero address.

UFL-04M: Redundant `else` Block

Type	Severity	Location
Gas Optimization	Informational	UnilendFlashLoanCore.sol L69-L71

Description:

The linked `else` block is redundant.

Recommendation:

We advise to omit the `else` block and directly use the `return` statement after the `if` block.

Alleviation:

The development team acknowledged this exhibit but opted to omit the recommended fix.

UFL-05M: Redundant Code Block

Type	Severity	Location
Gas Optimization	Informational	UnilendFlashLoanCore.sol L166-168

Description:

The linked `else` block is redundant, as it returns the default value of the `uint256` data type.

Recommendation:

We advise to remove the linked code block, as a failed execution of the `if` block will automatically return zero.

Alleviation:

The development team opted to consider our references and removed the redundant code block.

UFL-06M: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	UnilendFlashLoanCore.sol L268, L272, L287, L294, L303

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The development team opted to consider our references and removed the redundant functions.

UFL-07M: Conditional Optimization

Type	Severity	Location
Gas Optimization	Informational	UnilendFlashLoanCore.sol L315

Description:

The linked `for` loop redundantly looks-up the `_reserves` array on every iteration.

Recommendation:

We advise to introduce a local variable, assign it the `length` member of the `_reserves` array and use that variable instead.

Alleviation:

The development team acknowledged this exhibit but opted to store the entire array to `memory`.

UFL-08M: Potential Zero Flash Loan Fees

Type	Severity	Location
Volatile Code	Informational	UnilendFlashLoanCore.sol L346-L351

Description:

The linked function can set the flash loan fees to zero.

Recommendation:

We advise to add a `require` statement, creating a lower and upper bound for the potential fee values.

Alleviation:

The development team opted to consider our references and added two `require` statements, bounding the fee state variables within a desired range.

UFD-01S: Unchecked Value of ERC-20 `transfer()` / `transferFrom()` Call

Type	Severity	Location
Volatile Code	Minor	UnilendFDonation.sol L74, L107

Description:

The linked `transfer()` / `transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation:

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation:

The development team opted to consider our references and used `SafeERC20`'s `safeTransfer()` and `safeTransferFrom()` functions for the linked statements.

UFD-02S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	Minor	UnilendFDonation.sol L113

Description:

The linked function update the state of the contract after external calls.

Recommendation:

We advise to apply the Checks-Effects-Interactions pattern by moving the statement in L113 before the `if` block in L106-L110.

Alleviation:

The development team opted to consider our references and applied the Checks-Effects-Interactions pattern.

UFD-03S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	UnilendFDonation.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Lines: L1

Description: The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation: We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation: The development team opted to consider our references and locked the contract to Solidity compiler version `0.6.2`.

UFD-04S: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	UnilendFDonation.sol L14

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

Lines: L14

Description: The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation: Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation: The development team opted to consider our references and added an explicit visibility specifier to the linked state variable.

UFL-01S: Usage of `transfer()` for sending Ether

Type	Severity	Location
Volatile Code	Minor	UnilendFlashLoanCore.sol L380

Description:

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300` . This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation:

We advise that the linked `.transfer()` and `.send()` calls are substituted with the utilization of the `sendValue()` function from the `Address.sol` implementation of OpenZeppelin either by directly importing the library or copying the linked code.

Alleviation:

The development team opted to consider our references and introduced the pattern implemented in the aforementioned library.

UFL-02S: Contract Size

Type	Severity	Location
Language Specific	Informational	UnilendFlashLoanCore.sol General

Description:

Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet.

Recommendation:

We advise to remove redundant code.

Alleviation:

The development team acknowledged this exhibit.

UFL-03S: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	UnilendFlashLoanCore.sol L178

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references and added an explicit visibility specifier to the linked state variable.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.