Unit Protocol

Security Assessment

September 18th, 2020

By :
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Alex Papgeorgiou @ 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 indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

What isn’t a CertiK report?

- A statement about the overall bug free or vulnerability free nature of a piece of source code or any modules, technologies or code it interacts with.
- Guarantee or warranty of any sort regarding the intended functionality or security of any or all technology referenced in the report.
- An endorsement or disapproval of any company, team or technology.

Summaries

Project Summary

Project Name	Unit Protocol
Description	A decentralized borrowing protocol that allows using a variety of tokens as collateral.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository

Audit Summary

Delivery Date	Sep. 18, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Sep. 3rd, 2020 - Sep. 11th 2020

Vulnerability Summary

Total Issues	40
Total Critical	0
Total Major	0
Total Minor	11
Total Informational	29

Findings

ID	Title	Type	Severity
UNP-01	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-02	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-03	Potential for re-entrancy	Control Flow	Minor
UNP-04	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-05	Potential for re-entrancy	Control Flow	Minor
UNP-06	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-07	Unsafe division by zero	Arithmetic	Minor
UNP-08	Unnecessary relative import	Language Specific	Informational
UNP-09	Variable should be constant	Implementation	Minor
UNP-10	Unlabeled constants	Implementation	Informational
UNP-11	Possible integer overflow	Arithmetic	Minor
UNP-12	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-13	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-14	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-15	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-16	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-17	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-18	Unused return value, Inefficient greater-than comparison w/ zero	State Change, Performance	Minor, Informational
UNP-19	Unused return value, Inefficient greater-than comparison w/ zero	State Change, Performance	Minor, Informational
UNP-20	Unused return value, Inefficient greater-than comparison w/ zero	State Change, Performance	Minor, Informational

Findings (continued)

ID	Title	Type	Severity
UNP-21	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-22	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-23	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-24	Inefficient greater-than comparison w/ zero	Performance	Informational
UNP-25	Unused return value, Inefficient greater-than comparison w/ zero	State Change, Performance	Minor, Informational
UNP-26	Unused return value, Inefficient greater-than comparison w/ zero	State Change, Performance	Minor, Informational
UNP-27	Unused return value, Inefficient greater-than comparison w/ zero	State Change, Performance	Minor, Informational
UNP-28	Inefficient greater-than comparison w/ zero	Performance	Informational

UNP-01: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	Parameters.sol L161

Description:

The `Parameters.setInitialCollateralRatio` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-02: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	Parameters.sol L172

Description:

The `Parameters.setLiquidationRatio` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-03: Potential for re-entrancy

Type	Severity	Location
Control Flow	Minor	Vault.sol L158-L162

Description:

The `Vault.withdrawCol` function had an external call to the `ERC20SafeTransfer.safeTransferAndVerify` function that introduced the potential for re-entrancy due to ignoring the Solidity `Check Effects Interactions` pattern.

Recommendation:

We recommended applying all changes to state variables before making external calls, noting that if the transaction reverts, any changes made to state variables will be reverted as well.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-04: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	Vault.sol L205

Description:

The `Vault.chargeFee` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-05: Potential for re-entrancy

Type	Severity	Location
Control Flow	Minor	Vault.sol L216-L240

Description:

The `Vault.liquidate` function has external calls to the `ERC20SafeTransfer.safeTransferAndVerify` and `LiquidationSystem.liquidate` functions that introduced the potential for re-entrancy due to ignoring the Solidity `Check Effects Interactions` pattern.

Recommendation:

We recommended applying all changes to state variables before making external calls, nothing that if the transaction reverts, any changes made to state variables will be reverted as well, to store temporary copies of any required debt and collateral state variables for the user, subtract the user's debt from the overall debt of the asset, reset the user's debts and collateral for the asset, transfer the collateral to the liquidation system, and lastly liquidate the asset.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-06: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	Vault.sol L261

Description:

The `Vault.isContract` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the overall cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-07: Unsafe division by zero

Type	Severity	Location
Arithmetic	Minor	SafeMath.sol L27-L35

Description:

The `SafeMath.div` function did not provide a safe implementation and allowed for division by zero. As division by zero is considered a logic error, we pointed out that this is not considered a safe implementation.

Recommendation:

We suggested checking for zero in any client code that would use safe division beforehand and handling it gracefully in scope instead of allowing for division by zero in the `SafeMath.div` function and to restore the assertion in the `SafeMath.div` function to be inline with the OpenZeppelin `SafeMath` implementation it is based on.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-08: Unnecessary relative import

Type	Severity	Location
Language Specific	Informational	ChainlinkedUniswapOracle.sol L10

Description:

The `ChainlinkedUniswapOracle.sol` file has an import statement with a relative file path into the `node_modules` folder:

```
import { UniswapOracle, IUniswapV2Pair } from
 '../../node_modules/@keydonix/uniswap-oracle-contracts/source/UniswapOracle.sol';
```

Recommendation:

Since the project already depends on `@keydonix/uniswap-oracle-contracts`, consider refactoring the import path:

```
import { UniswapOracle, IUniswapV2Pair } from '@keydonix/uniswap-oracle-
contracts/source/UniswapOracle.sol';
```

Alleviation:

The recommendation was applied in commit 518a09081aadda6a383f9845837ed7045101e64f.

UNP-09: Variable should be constant

Type	Severity	Location
Implementation	Minor	ChainlinkedUniswapOracle.sol L23

Description:

The `ChainlinkedUniswapOracle.MIN_BLOCKS_BACK` state variable was not declared constant in order to prevent modification after initialization and reduce the overall cost of gas.

Recommendation:

We recommended adding the `constant` attribute to the `ChainlinkedUniswapOracle.MIN_BLOCKS_BACK` state variable.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-10: Unlabeled constants

Type	Severity	Location
Implementation	Informational	ChainlinkedUniswapOracleLP.sol L75, L76, L78, L83, L84, L86, L87

Description:

The `ChainlinkedUniswapOracleLP.assetToUsd` function made use of unlabeled constant magic numbers, which made the function difficult to review correctly.

Recommendation:

We recommended making a constant variable for each constant magic number used in the `ChainlinkedUniswapOracleLP.assetToUsd` function in order to clarify the logic of USD calculation code.

Alleviation:

While constant variables were created in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d, we suggest giving the constant variables more clearly-defined names in order to better portray their origin.

UNP-11: Possible integer overflow

Type	Severity	Location
Arithmetic	Minor	ChainlinkedUniswapOracleLP.sol L99

Description:

The `ChainlinkedUniswapOracleLP.sqrt` function implements the Babylonian method for calculating the square root of a supplied `uint x` parameter. The implementation used an initial iteration value of `z = (x + 1) / 2` which could result in an integer overflow if `x` is `uint(-1)` and allowed for division by zero in the calculation of `z = (x / z + z) / 2`, which would have caused the transaction to revert.

Recommendation:

While the value returned from the previous `ChainlinkedUniswapOracleLP.sqrt` implementation is valid for other values, we recommended that it should be refactored in order to prevent against division by zero.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-12: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerStandard.sol, L52-L58

Description:

The `VaultManagerStandard.deposit` function had inefficient greater-than (`>`) comparisons between an unsigned integers and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the overall cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-13: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerStandard.sol, L75

Description:

The `VaultManagerStandard.repay` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-14: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerStandard.sol, L107-L120

Description:

The `VaultManagerStandard.repayAllAndWithdraw` function had inefficient greater-than (`>`) comparisons between unsigned integers and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, consider converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-15: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswap.sol, L39

Description:

The `VaultManagerUniswap.spawned` modifier had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-16: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswap.sol, L83

Description:

The `VaultManagerUniswap.spawn` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-17: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswap.sol, L123

Description:

The `VaultManagerUniswap.depositAndBorrow` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-18: Unused return value, Inefficient greater-than comparison w/ zero

Type	Severity	Location
State Change	Minor	VaultManagerUniswap.sol, L175
Performance	Informational	VaultManagerUniswap.sol, L155-L172

Description:

The `VaultManagerUniswap.withdrawAndRepay` function had inefficient greater-than (`>`) comparisons between unsigned integers and the constant value of 0 and ignored the user debt value returned from the call to the `Vault.repay` function. While these issues will not lead to compromising the system, we pointed out that they should generally be avoided.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas and to emit all events before making external calls.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-19: Unused return value, Inefficient greater-than comparison w/ zero

Type	Severity	Location
State Change	Minor	VaultManagerUniswap.sol, L240
Performance	Informational	VaultManagerUniswap.sol, L210-L236

Description:

The `VaultManagerUniswap.withdrawAndRepayUsingCol` function had inefficient greater-than (`>`) comparisons between unsigned integers and the constant value of 0 and ignored the user debt value returned from the call to the `Vault.repay` function. While these issues will not lead to compromising the system, we pointed out that they should generally be avoided.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation for the inefficient comparisons was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d, but the user debt value returned from the call to the `Vault.repay` function is still ignored.

UNP-20: Unused return value, Inefficient greater-than comparison w/ zero

Type	Severity	Location
State Change	Minor	VaultManagerUniswap.sol, L269
Performance	Informational	VaultManagerUniswap.sol, L260-L266

Description:

The `VaultManagerUniswap._depositAndBorrow` function had inefficient greater-than (`>`) comparisons between unsigned integers and the constant value of 0 and made a call to the `Vault.borrow` function without taking the returned user debt value into account. While these issues will not lead to compromising the system, we pointed out that they should generally be avoided.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas, determining if the user debt value returned from the call to the `Vault.borrow` function is necessary and incorporating it into the system in some way, or emitting an event to use the value.

Alleviation:

The recommendation for the inefficient comparisons was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d, but the user debt value returned from the call to the `Vault.borrow` function is still ignored.

UNP-21: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswap.sol, L307

Description:

The `VaultManagerUniswap._ensureCollateralization` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-22: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswapLP.sol, L39

Description:

The `VaultManagerUniswapLP.spawned` modifier had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-23: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswapLP.sol, L83

Description:

The `VaultManagerUniswapLP.spawn` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-24: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswapLP.sol, L124

Description:

The `VaultManagerUniswapLP.depositAndBorrow` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-25: Unused return value, Inefficient greater-than comparison w/ zero

Type	Severity	Location
State Change	Minor	VaultManagerUniswapLP.sol, L177
Performance	Informational	VaultManagerUniswapLP.sol, L157-L174

Description:

The `VaultManagerUniswapLP.withdrawAndRepay` function had inefficient greater-than (`>`) comparisons between unsigned integers and the constant value of 0 and ignored the user debt value returned from the call to the `Vault.repay` function. While these issues will not lead to compromising the system, we pointed out that they should generally be avoided.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-26: Unused return value, Inefficient greater-than comparison w/ zero

Type	Severity	Location
State Change	Minor	VaultManagerUniswapLP.sol, L241
Performance	Informational	VaultManagerUniswapLP.sol, L211-L237

Description:

The `VaultManagerUniswapLP.withdrawAndRepayUsingCol` function had inefficient greater-than (`>`) comparisons between unsigned integers and the constant value of 0 and ignored the user debt value returned from the call to the `Vault.repay` function. While these issues will not lead to compromising the system, we pointed out that they should generally be avoided.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.

UNP-27: Unused return value, Inefficient greater-than comparison w/ zero

Type	Severity	Location
State Change	Minor	VaultManagerUniswapLP.sol, L270
Performance	Informational	VaultManagerUniswapLP.sol, L261-L267

Description:

The `VaultManagerUniswapLP._depositAndBorrow` function had inefficient greater-than (`>`) comparisons between unsigned integers and the constant value of 0 and ignored the user debt value returned from the call to the `Vault.borrow` function. While these issues will not lead to compromising the system, we pointed out that they should generally be avoided.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas and determining if the user debt value returned from the call to the `Vault.borrow` function is necessary and incorporating it into the system in some way, or emitting an event to use the value.

Alleviation:

The recommendations for the inefficient comparisons were applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d, but the user debt value returned from the call to the `Vault.borrow` function is still ignored.

UNP-28: Inefficient greater-than comparison w/ zero

Type	Severity	Location
Performance	Informational	VaultManagerUniswapLP.sol, L309

Description:

The `VaultManagerUniswapLP._ensureCollateralization` function had an inefficient greater-than (`>`) comparison between an unsigned integer and the constant value of 0.

Recommendation:

As unsigned integers are restricted to the non-negative range, we recommended converting the comparison to inequality in order to optimize the cost of gas.

Alleviation:

The recommendation was applied in commit 0a0b0c0b017545ef8f6812d71f80746feb7b8c0d.