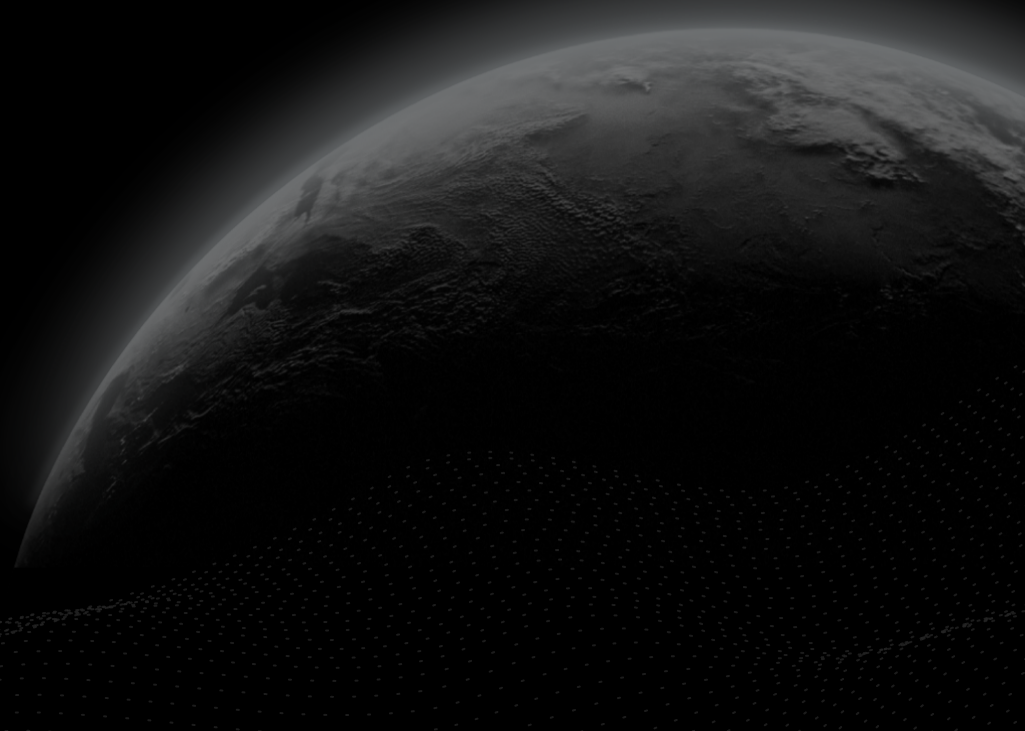
CERTIK

Security Assessment

LendeXe

CertiK Verified on Nov 16th, 2022



CertiK Verified on Nov 16th, 2022

LendeXe

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Lending	Ethereum	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 11/16/2022	N/A

CODEBASE

https://gitlab.com/l2921/lendexe-protocol/tree/8575a038a9e01152726ebb57501231ee706c07de

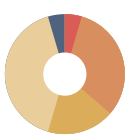
...View All

COMMITS

8575a038a9e01152726ebb57501231ee706c07de

...View All

Vulnerability Summary



22	13	0	0	9	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

1	Critical	1 Resolved	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
7	Major	5 Resolved, 2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
4	Medium	3 Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
9	Minor	4 Resolved, 5 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
1	Informational	1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | LENDEXE

CODEBASE | LENDEXE

Repository

https://gitlab.com/l2921/lendexe-protocol/tree/8575a038a9e01152726ebb57501231ee706c07de

Commit

8575a038a9e01152726ebb57501231ee706c07de

AUDIT SCOPE | LENDEXE

14 files audited ● 7 files with Acknowledged findings ● 1 file with Resolved findings ● 6 files without findings

ID	File	SHA256 Checksum
CCP	contracts/Comptroller.sol	5a5e08c5c23e11ee9c67f8011c0003a826cd53dc1f8f7734ce3de0308efa5eaf
LJM	contracts/LendexeJumpRateModel.sol	4700a46fb52672087218dd1d801771303b3484595a4343d6ea6678ed24145e01
LDK	contracts/LockDrop.sol	9e52ef058688a726a1a173cdc4f45f8adbedae2d891de022d3b0c345c774f118
ULC	contracts/Stablecoin/UltimateLoan.sol	6fd36f8f68c152e73a08f72fc6283a271565eb57e8351fb275ac23cabec06391
ULK	contracts/Stablecoin/UltimateLoanLock.sol	bdc8b38f274294000ba1d00b3e5b54d228d479b0f0fb13eb29e747ccfce13d8f
XSC	contracts/Stablecoin/XSD.sol	dedb3a007ae95a3d376de27a43f2f827ae39edc83810951c098264ea0a181d74
XSP	contracts/Stablecoin/XSDStabilizer.sol	20ada3dd71615581678c9615d83706194c4146d6b7ef7ae02883a71ee02b5fcb
SHT	contracts/SwapTools/SwapHelper.sol	1fedce0dd5e776eeb2b4f79ea9d6178c0e326c41497cfd6caf37e36a3f9491ad
CIK	contracts/ComptrollerInterface.sol	09760de54286f88174e8ab44caec3aade40fd66fce7ccf14e9fe7d5b5a94f109
CSK	contracts/ComptrollerStorage.sol	7f4de7438c75452226e7ffd7e104e400da2913b01ee37e2756d0fc71a48eaf4b
POK	contracts/PriceOracle.sol	9cf5b561db940852620af5132f050cfd0ec271dafda103d733208178fa1e70fd
XSK	contracts/Stablecoin/XSDInterface.sol	89f6a1ed0c05c5f9b3b24294285d58de71c5afc818149039b8bb06a92b6d5d05
ISS	contracts/SwapTools/ISwapRouter.sol	39fd92b50b0fc59343558340999f2f9cc7b24222a95568ff2851b6ffb208cf48
IUS	contracts/SwapTools/IUniswapV3SwapCallback.sol	171a9a692e71b6d532df655695b0b672bd8ea5dcca3b3363131700b45b0171c6

APPROACH & METHODS | LENDEXE

This report has been prepared for LendeXe to discover issues and vulnerabilities in the source code of the LendeXe project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

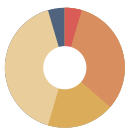
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | LENDEXE



22	1	7	4	9	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for LendeXe. Through this audit, we have uncovered 22 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged
CCP-01	**Centralization Risks In The Function `fixBadAccruals()`**	**Centralization / Privilege**	**Major**	**Resolved**
CCP-02	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
CCP-03	Lack Of Access Control Over `_setBorrowCapGuardian()` Function	Control Flow	Major	Resolved
CKP-01	SafeMath Not Used	Mathematical Operations	Minor	Acknowledged
CKP-02	Unchecked Return Value	Volatile Code	Minor	Resolved
LDK-01	Lack Of Sanity Check For `lockTime`	Volatile Code	Medium	Resolved
LDK-02	Lack Of Input Validation For `lockEndBlocks`	Volatile Code	Medium	Resolved
LDK-03	Unreasonable Fee Calculation	Logical Issue	Minor	Resolved
LJM-01	Mathematical Verification	Logical Issue	Minor	Acknowledged

ID	Title	Category	Severity	Status
SCP-01	Potential Arbitrage Attack	Control Flow	Major	Resolved
SCP-02	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
SHT-01	Lack Of Access Control Over `_setSwapRouter()` Function	Control Flow	Major	Resolved
SHT-02	Potential Sandwich Attacks	Logical Issue	Minor	Resolved
ULC-01	Lack Validation For `nftAmount`	Logical Issue	Major	Resolved
ULC-02	Invalid Validation	Logical Issue	Medium	Resolved
ULC-03	Potential Unable To Mint XSD Tokens	Logical Issue	Medium	Acknowledged
ULC-04	Logic Issue Of Function `setSupplyTokens()`	Logical Issue	Minor	Resolved
ULC-05	`lexeVault` Address Can Acquire `xLEXE`	Logical Issue	Minor	Acknowledged
XSC-01	Incorrect Function Visibility	Control Flow	Critical	Resolved
XSP-01	Potential Unable To Burn XSD Tokens	Logical Issue	Minor	Acknowledged
LDK-04	Logical Issue On Function `calculateTokenPortions()`	Logical Issue	Informational	Acknowledged

GLOBAL-01 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Minor		Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party `ISwapRouter` , and `IUniswapV3SwapCallback` , etc protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic requires interaction with `ISwapRouter` , `IUniswapV3SwapCallback` , etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

The team acknowledged this issue and they will constantly monitor the statuses of 3rd-parties.

CCP-01 CENTRALIZATION RISKS IN THE FUNCTION `fixBadAccruals()`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/Comptroller.sol: 1446~1449**	**Resolved**

Description

In the history of the compound protocol, the calculation of the Comp rewards becomes incorrect after proposal 062 `Split COMP rewards distribution and bug fixes` is executed.

In response to this problem, the function `fixBadAccruals()` has been temporarily added in proposal 065(`Correct Over-Accrued COMP`) and removed later.

The function `fixBadAccruals()` is a centralized function. It will update the users' unclaimed Comp rewards to decrease the incorrect rewards, and record the COMP debt in the variable `compReceivable` for the users whose unclaimed Comp rewards are not enough to decrease.

The function `fixBadAccruals()` is only used to handle the error caused by this upgrade migration in the Compound protocol, which is useless in the newly deployed Compound forks.

Refer to:

- https://compound.finance/governance/proposals/62
- https://compound.finance/governance/proposals/65

Recommendation

We recommend removing the function `fixBadAccruals()` .

Alleviation

The team heeded our advice and resolved this issue in commit `03b457cd44903229961e5feac52933899f7e161b` .

CCP-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/Comptroller.sol: 1074, 1102, 1123, 1192, 1226, 1301, 1328, 1341, 1354, 1378, 1394, 1410, 1422**	**Acknowledged**

Description

In the contract `Comptroller`, the role `admin` has authority over the following functions:

- function _setPriceOracle(), to set a new price oracle for the comptroller.
- function _setCloseFactor(), to set the `closeFactor` used when liquidating borrows.
- function _setCollateralFactor(), to set the `collateralFactor` for a market
- function _setLiquidationIncentive(), to set the `liquidationIncentive`.
- function _supportMarket(), to add and initialize the market to the markets mapping and set it as listed.
- function _setMarketBorrowCaps(), to the given borrow caps for the given `xToken` markets. Borrowing that brings total borrows to or above the borrowing cap will revert.
- function _setBorrowCapGuardian(), to change the Borrow Cap Guardian.
- function _setPauseGuardian(), to change the address of the Pause Guardian.
- function _setMintPaused(), to pause or unpause the mint.
- function _setBorrowPaused(), to pause or unpause the borrow.
- function _setTransferPaused(), to pause or unpause the transfer.
- function _setSeizePaused(), to pause or unpause the seize.
- function _setSwapHelperAddress(), to set the address for the contract `SwapHelperAddress`.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

The team acknowledged this issue and they will use a multi-signature wallet with ⅗ signers.

CCP-03 LACK OF ACCESS CONTROL OVER `_setBorrowCapGuardian()` FUNCTION

Category	Severity	Location	Status
Control Flow	Major	contracts/Comptroller.sol: 1341	Resolved

Description

The function `_setSwapHelperAddress()` is `external` and can be called by anyone as long as the contract is deployed.

Recommendation

We advise the client to set up access controls over the functions so only authorized users can call the function.

Alleviation

The team heeded our advice and resolved this issue in commit `65608d279194b641f371d13e496fcc3be42627d7`.

CKP-01 | SAFEMATH NOT USED

Category	Severity	Location	Status
Mathematical Operations	Minor	contracts/LendexeJumpRateModel.sol; contracts/LockDrop.sol; contracts/Stablecoin/UltimateLoan.sol; contracts/Stablecoin/XSD.sol; contracts/Stablecoin/XSDStabilizer.sol	Acknowledged

Description

These expressions in the contracts do not check arithmetic overflow. Such unsafe math operations may cause unexpected behavior if unusual parameters are given.

Recommendation

We advise the client to use OpenZeppelin's SafeMath library for all of the mathematical operations.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

CKP-02 | UNCHECKED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/LockDrop.sol: 245, 444; contracts/Stablecoin/UltimateLoan.sol: 272, 297, 325, 330, 390; contracts/Stablecoin/UltimateLoanLock.sol: 38, 48~49; contracts/Stablecoin/XSDStabilizer.sol: 351, 395; contracts/SwapTools/SwapHelper.sol: 124~125	Resolved

Description

The linked functions invocations do not check the return value of the function call which returns a value in case of a proper call.

Recommendation

We would advise to check the return value of the function for intended values.

Alleviation

The team heeded our advice and resolved this issue in commit `e35e435fd99ea22f1ef90951bd91e70443e73c7e`.

LDK-01 | LACK OF SANITY CHECK FOR `lockTime`

Category	Severity	Location	Status
Volatile Code	Medium	contracts/LockDrop.sol: 133~137, 173~177, 364	Resolved

Description

There's no sanity check to validate if a `lockTime` is existing. If the `lockTime` does not exist, the user who locked assets will not get the rewards.

Recommendation

We recommend adding the sanity check to ensure the `timeLock` exists.

Alleviation

The team heeded our advice and resolved this issue in commit `7ef20bc65f8ee692a8fa8da551949a661ccc9b93`.

LDK-02 | LACK OF INPUT VALIDATION FOR `lockEndBlocks`

Category	Severity	Location	Status
Volatile Code	Medium	contracts/LockDrop.sol: 100	Resolved

Description

There is no validation to ensure the `lockEndBlocks[i]` is larger than the `_lockingPeriodEndBlock`.

Recommendation

We recommend adding the validation.

Alleviation

The team heeded our advice and resolved this issue in commit `65608d279194b641f371d13e496fcc3be42627d7`.

LDK-03 | UNREASONABLE FEE CALCULATION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/LockDrop.sol: 157~158, 203~204	Resolved

Description

As per the fee calculation logic, if the amount is less than 1, all the locked assets amount will be charged as fees.

```
157    uint256 actualAmount = (amount * 99) / 100;
158    uint256 fee = amount - actualAmount;
```

Recommendation

We recommend reviewing the logic again and ensuring it is intended.

Alleviation

The team heeded our advice and resolved this issue in commit `b5a9868176be20db564390ebe7e78212dc921417`.

LJM-01 | MATHEMATICAL VERIFICATION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/LendexeJumpRateModel.sol: 47~51	Acknowledged

Description

The function `getBorrowRate()` is using some algorithms. The Mathematical verification of these algorithms is not in the scope of this audit. The function logic will be checked based on the requirement documents.

Recommendation

We advise the client to revisit the design and ensure it is intended.

Alleviation

The team acknowledged this issue and provided the below medium post for reference.

Reference:

LendeXe Interest Rate Strategy

SCP-01 | POTENTIAL ARBITRAGE ATTACK

Category	Severity	Location	Status
Control Flow	Major	contracts/Stablecoin/UltimateLoan.sol: 214~217; contracts/Stablecoin/XSDStabilizer.sol: 313	Resolved

Description

The user can provide assets by calling the function provide() in the contract `UltimateLoan` and then directly call the function `burn()` / `burnShares()` in the contract `XSDStabilizer` to perform an arbitrage attack. The steps can be as follows,

1. call function `supply()` to lock xLEXE tokens as collateral, then receive XSD tokens, which is 150% of locked XLEXE tokens in value.
2. call the function `burn()` / `burnShares()` in the contract `XSDStabilizer` instead of the function `repay()` to burn XSD tokens will eventually receive 50% of the locked XLEXE tokens value in profit.

```
 240      (uint256 mintAmount, ) = stabilizer.mintShares(xsdAmount, supplyTokens);
//UL will gain double the xsd amount that was set as input because it locks an equal amount of LEXE
 241      uint256 portionOfOwnerMint = mul_(mintAmount, Exp({mantissa: 0.75e18}));
```

In an extreme case, the user can give up repaying the loan, and receive 50% of the locked `xLEXE` tokens value in profit.

Recommendation

We recommend refactoring the logic.

Alleviation

The team acknowledged this issue and they confirmed this is by design:

"The users must lock their xLexe or Lexe for a period(e.g. several months), then they get Lexe and use the UL. The profit is the reward to them."

SCP-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/Stablecoin/UltimateLoan.sol: 75~78, 101, 121, 126, 199; contracts/Stablecoin/UltimateLoanLock.sol: 41; contracts/Stablecoin/XSD.sol: 262, 283, 350, 365; contracts/Stablecoin/XSDStabilizer.sol: 426, 583, 593	Acknowledged

Description

In the contract `UltimateLoan`, the role `admin` has authority over the following functions:

- function setSupplyTokens(), set supply tokens, users supply these tokens to the market and receive XDS in exchange.
- function setStatus(), to pause or unpause the Ultimate Loan.
- function setDueDate(), to set the due date.
- function setULLock(), to set the address of the contract `UltimateLoanLock`.
- function setULPercentage(), to set the exchange rate of the contract `UltimateLoanLock`.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority.

In the contract `XSD`, the role `admin` has authority over the following functions:

- function _setStabilizer(), to set the address of the `XSDStabilizer` contract.
- function _delegateAdmin(), to delegate admin address to a different one.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority.

In the contract `XSD`, the role `Stabilizer` has authority over the following functions:

- function mint(), to mint any amount of `XSD` tokens to any `account` address.
- function burn(), to destroy any amount of `XSD` tokens for the `account` address.

Any compromise to the `Stabilizer` account may allow a hacker to take advantage of this authority.

In the contract `XSDStabilizer`, the role `admin` has authority over the following functions:

- function _unbanToken(), to unban a previously banned token.
- function _setUltimateLoanAddress(), to update the address of the `UltimateLoan` contract.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority.

In the contract `XSDStabilizer`, the role `ultimateLoanAddress` has authority over the following functions:

- function burnDirectly(), to burn XSD directly without handing out stablecoins.

Any compromise to the `ultimateLoanAddress` account may allow a hacker to take advantage of this authority.

In the contract `UltimateLoanLock`, the role `ultimateLoan` has authority over the following functions:

- function unlockUser(), to unlock `xLEXE` tokens and transfer to the user and `ultimateLoan` address.

Any compromise to the `ultimateLoan` account may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

The team acknowledged this issue and they will use a multi-signature wallet with ⅗ signers.

SHT-01 LACK OF ACCESS CONTROL OVER `_setSwapRouter()` FUNCTION

Category	Severity	Location	Status
Control Flow	Major	contracts/SwapTools/SwapHelper.sol: 27	Resolved

Description

The function `_setSwapRouter()` is `external` and can be called by anyone as long as the contract is deployed.

Recommendation

We advise the client to set up access controls over the functions so only authorized users can call the function.

Alleviation

The team heeded our advice and resolved this issue in commit `65608d279194b641f371d13e496fcc3be42627d7`.

SHT-02 | POTENTIAL SANDWICH ATTACKS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/SwapTools/SwapHelper.sol: 47~51, 53~55	Resolved

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction is attacked) a transaction to sell the asset.

The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- `swapRouter.exactInputSingle()`
- `swapRouter.exactInput()`

Recommendation

We recommend setting reasonable minimum output amounts based on token prices when calling the aforementioned functions.

Alleviation

The team heeded our advice and set a minimum amount of 3% in commit `028faea331beb8610055d87bad27e4fd896a0d6f`.

ULC-01 | LACK VALIDATION FOR `nftAmount`

Category	Severity	Location	Status
Logical Issue	Major	contracts/Stablecoin/UltimateLoan.sol: 224	Resolved

Description

The input variable `nftAmount` is not validated in the function `supply()`, so the user can input the arbitrary `nft` amount to mint the maximum allowed XSD tokens.

Recommendation

We recommend refactoring the logic.

Alleviation

The team heeded our advice and resolved this issue in commit `65608d279194b641f371d13e496fcc3be42627d7`.

ULC-02 | INVALID VALIDATION

Category	Severity	Location	Status
Logical Issue	Medium	contracts/Stablecoin/UltimateLoan.sol: 363	Resolved

Description

The contract deployer can input himself as the admin to pass the validation.

```
349     constructor(
350         XSDStabilizer XSDStabilizer_,
351         address lexeVault_,
352         address payable admin_,
353         Lexe lexe_,
354         XToken xLEXE_,
355         XToken[] memory supplyTokens_,
356         uint256[] memory supplyTokenShares_,
357         XSDInterface xsd_,
358         PriceOracle oracle_
359     ) public {
360         // Creator of the contract is admin during initialization
361         admin = msg.sender;
362 
363         require(msg.sender == admin, "only admin may initialize the UL");
364 
365         // Set initial exchange rate
366         ULPercentage = uint256(50);
367         stabilizer = XSDStabilizer(XSDStabilizer_);
368         ...
```

Recommendation

We recommend reviewing the logic and fixing the issue.

Alleviation

The team heeded our advice and resolved this issue in commit `65608d279194b641f371d13e496fcc3be42627d7`.

ULC-03 | POTENTIAL UNABLE TO MINT XSD TOKENS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/Stablecoin/UltimateLoan.sol: 222	Acknowledged

Description

Users locked their `xLEXE` tokens in contract `UltimateLoanLock` for at least 3 months and expected to mint XDS tokens, however, they may not be able to mint XSD tokens due to the market share of XSD being too high.

```
242     require(
243         getHypotheticalLendexeShare(amount) < 0.6e18,
244         "Market share of XSD is too high!"
245     );
246
247
```

Recommendation

We recommend reviewing the logic again and ensuring there are enough XSD tokens to mint.

Alleviation

The team added a restriction in the function `setStatus()` in commit 41d57952a068898ecba7fc317028298a512b75ae, to ensure that UL cannot start when XSD's market share reaches 50%.

ULC-04 | LOGIC ISSUE OF FUNCTION `setSupplyTokens()`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Stablecoin/UltimateLoan.sol: 75~78	Resolved

Description

The array `supplyTokens` is not reset before new supply tokens are pushed, so it is impossible to update the supply tokens and shares if the function is called again.

```
87      supplyTokens.push(
88          supplyToken({
89              token: supplyTokens_[i],
90              share: Exp({mantissa: supplyTokenShares_[i]})
91          })
92      );
```

Recommendation

We recommend resetting the array `supplyTokens` in the function `supplyTokens()`.

Alleviation

The team heeded our advice and resolved this issue in commit `65608d279194b641f371d13e496fcc3be42627d7`, by setting the member `Length` to zero. This solution will work in the solidity versions below 0.6.0, but not work in solidity 0.6.0 and above.

ULC-05 | `lexeVault` ADDRESS CAN ACQUIRE `xLEXE`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Stablecoin/UltimateLoan.sol: 330	Acknowledged

Description

The `liquidate()` function calls the `xLEXE.transferFrom()` function with the `to` address specified as `lexeVault` for acquiring the `xLEXE` tokens. As a result, over time the `lexeVault` address will accumulate a significant portion of `xLEXE` tokens. If the `lexeVault` is an EOA (Externally Owned Account), the mishandling of its private key can have devastating consequences for the project as a whole.

Recommendation

Please make sure the deployer set the correct `lexeVault` address.

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

XSC-01 | INCORRECT FUNCTION VISIBILITY

Category	Severity	Location	Status
Control Flow	Critical	contracts/Stablecoin/XSD.sol: 231~235, 308~312, 328~332	Resolved

Description

The visibility of the function `transfer()` is public, which allows anyone to transfer tokens from the `from` address to the `to` address.

The visibility of the function `approve()` is public, which allows anyone to set allowance of `spender` over the `owner's` token.

The visibility of the function `spendAllowance()` is public, which allows anyone to update the `owner's` allowance for `spender` based on the spent `amount`.

Recommendation

We recommend updating the visibility of `transfer()`, `approve()` and `spendAllowance()` to internal.

Alleviation

The team heeded our advice and resolved this issue in commit `b82e64abcd34d935b62e24925f98cfc81b984551`.

XSP-01 | POTENTIAL UNABLE TO BURN XSD TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Stablecoin/XSDStabilizer.sol: 322~325, 376~380	Acknowledged

Description

If the supply token status is marked as `BLACKLISTED` or `BANNED` , users are possibly unable to burn the XDS tokens to get the supply tokens.

Recommendation

We recommend reviewing the logic and ensuring it is intended.

Alleviation

The team acknowledged this issue and they stated the user is not forced to reclaim the same `Stablecoin` as he has supplied.

LDK-04 | LOGICAL ISSUE ON FUNCTION `calculateTokenPortions()`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/LockDrop.sol: 364	Acknowledged

Description

The function `calculateTokenPortions()` in the contract `LockDrop` should be called successfully daily. Otherwise, users who locked assets will potentially lose their rewards. We want to check with the team for more detail about the mechanism that can ensure the function `calculateTokenPortions()` run successfully daily.

Recommendation

We recommend the client review the logic.

Alleviation

The team acknowledged this issue and they stated they will ensure successful calls.

OPTIMIZATIONS | LENDEXE

ID	Title	Category	Severity	Status
LDK-05	Duplicated Assignment	Logical Issue	Optimization	Resolved
LDK-06	Missing Error Messages	Coding Style	Optimization	Resolved
LDK-07	Missing Validation For Array Length	Logical Issue	Optimization	Resolved

LDK-05 | DUPLICATED ASSIGNMENT

Category	Severity	Location	Status
Logical Issue	Optimization	contracts/LockDrop.sol: 114~116	Resolved

Description

`vt.lockEndBlock` is assigned the same value twice.

```
111     for (uint256 i = 0; i < _lockTimes.length; i++) {
112          uint8 distributionTime = uint8(totalSupply[i] / tokensPerDay);
113          VestingModel storage vt = vestingModels[_lockTimes[i]];
114          vt.lockEndBlock = lockEndBlocks[i];
115          vt.totalSupply = totalSupply[i];
116          vt.lockEndBlock = lockEndBlocks[i];
117          vt.distributionTime = distributionTime;
118      }
```

Recommendation

Consider removing duplicated one.

Alleviation

The team heeded our advice and resolved this issue in commit `e35e435fd99ea22f1ef90951bd91e70443e73c7e`.

LDK-06 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Optimization	contracts/LockDrop.sol: 160, 206	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

The team heeded our advice and resolved this issue in commit `2cd5e08a6de1c8f996ddd02deb748d5a38e06ea8`.

LDK-07 | MISSING VALIDATION FOR ARRAY LENGTH

Category	Severity	Location	Status
Logical Issue	Optimization	contracts/LockDrop.sol: 100~102	Resolved

Description

There is no validation between `_lockTimes.length` , `lockEndBlocks.length` , `totalSupply.length` and `_xTokenAddresses.length` in constructor. And there is no validation between `amounts.length` and `_xTokenAddresses.length` in function `lockAssets()` .

Recommendation

Consider adding the validation.

Alleviation

The team heeded our advice and resolved this issue in commit `55b1c9c4ece1aa80f17c019c949690faa482000f` .

APPENDIX | LENDEXE

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.