CERTIK

Security Assessment

xDollar.fi

Dec 3rd, 2021

Table of Contents

Summary

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

- 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.

Overview

Project Summary

Project Name	xDollar.fi
Platform	Polygon
Language	Solidity
Codebase	https://github.com/xDollar-Finance/xDollar-new-contracts
Commit	034bf841dee1f9c601e5af1b77ae31bb24aad5f8 934f7defbc85492e098ff4ba423536c430fa6e65 e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

Audit Summary

Delivery Date	Dec 03, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● Major	17	0	0	1	15	1
● Medium	4	0	0	2	0	2
● Minor	9	0	0	3	0	6
● Informational	26	0	0	9	0	17
● Discussion	0	0	0	0	0	0

Audit Scope

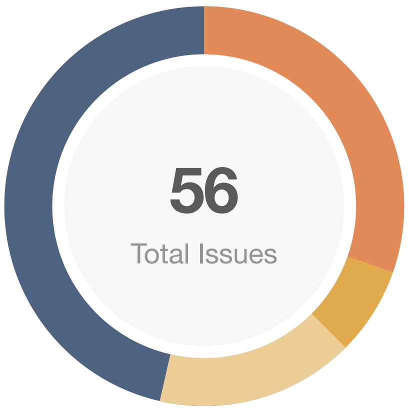
ID	File	SHA256 Checksum
AVI	Dependencies/AggregatorV3Interface.sol	b2ebef44df63d7e32405e0eb398a5868123c5f1800005c02d44d53cec38a28df
BMD	Dependencies/BaseMath.sol	fada1c3c95dcfa780a6d03bab5e1f329201a30ec94140b54818559f9e720cbc4
CCD	Dependencies/CheckContract.sol	aa32079b9f38a1669beb9fefe2e0af95fc543e1b717617713f00a648f0dc4b66
IER	Dependencies/IERC20.sol	64f3e9f771f7ba660ba11cf966318da692834288126f5b58601d8ba3ffc1a3fa
IEC	Dependencies/IERC2612.sol	83a6ac8c0f185342c500ada926f0dbe734a959382bbcbe8e1c69bcf7faa2454b
ITD	Dependencies/ITellor.sol	0627bd40d58674014ea1b2f8f2643a22fb10b609671a0f0bae6fc1e4f51e7bf8
LBD	Dependencies/LiquityBase.sol	21349deeb3c6378ea9d4cb3183b9ebee7d559dd64d36b7d44a8394ca1def5589
LMD	Dependencies/LiquityMath.sol	0d418289d7ba0ab052d1fff29b3c832bd809175d9bd69b8cd5552184a5e1bf17
LSM	Dependencies/LiquitySafeMath128.sol	93aa3e470a6e581ea7e515d2e34737f5e619af229637054ac09f0d303c0bddd2
ODD	Dependencies/Ownable.sol	645208d3053f1ee614b73776e9c638f4529062bfc3333fa06ba5663d9193405b
SMD	Dependencies/SafeMath.sol	caa5397440fd9a0988eb40c136bd7a58baad05012edcf244f6b586e167e531f6
TCD	Dependencies/TellorCaller.sol	941a091db174da098d6807615ae8e7576b22d2f1ade979be793070ac35872bc9
DDC	Dependencies/console.sol	fe7de02fbe78bf1af499331c9a5a404299a7141f0800e942e29b55c8c64029dc
IAP	Interfaces/IActivePool.sol	a5b69eac41d0290a9833dce7dad2f7ba333fbce41b117a98759f533052ce3207
IBO	Interfaces/IBorrowerOperations.sol	4f43b0d928ff5dfd6530cc0e69ae983bec4b9b6e1948409e7b3b5dbdac99da26
ICS	Interfaces/ICollSurplusPool.sol	77cb1dce3f9d2ff8ba14d3e5300840b60ed3ac7d05c7fba9f35a110b700b6ca1

ID	File	SHA256 Checksum
ICI	Interfaces/ICommunityIssuance.sol	1b25e623b3db2a2d18dd30d347823e8640f601747a3632fb941d9bcdf84fd898
IDP	Interfaces/IDefaultPool.sol	068005ed4808f182ef628c2377192ad6d5b24fda0184acd8f3d1acfc8bdc8f8f
ILQ	Interfaces/ILQTYStaking.sol	380d6e754e4cdc7609446319c15324be93be1dacb322601ddaecf3108d006e5a
ILT	Interfaces/ILQTYToken.sol	6d54ecced315fda5a33ddeaf729f178fd55cb5d0990fa7157272fdf1efa2b9df
ILU	Interfaces/ILUSDToken.sol	0c4dfc856818151537469427d4e19e11821f6c57f8421af22c974eebd7e27658
ILB	Interfaces/ILiquityBase.sol	48697f434db39ab90174b90dd36ce654ec78e3e8ed169efb2dad119761fdab4a
ILC	Interfaces/ILockupContractFactory.sol	7e7c6a8d9f4dc43a6f02b1de70175a5964f7ef6e835492d426f8aa1854e20358
IPI	Interfaces/IPool.sol	f6698ee9ea04e1c0270c22aa98c6d592f64cdbd7f9d3293e6cc280aa19ce6a8a
IPF	Interfaces/IPriceFeed.sol	b30789ec4ee77a4bd502aee8f1ca7b3368fc9250eac83ab4fbd737fdd094d2c1
IST	Interfaces/ISortedTroves.sol	1756c28b2f3e6c8cc2a76cb041311862fe6471cc4f0aa09c46f76be16d714a6e
ISP	Interfaces/IStabilityPool.sol	91f7f57477f83611b6d58d31a993c09a2032cb4748cd183c8726c8d1246e2eb6
ISC	Interfaces/IStableCollActivePool.sol	bad04c0d694e979ef6d643ff8c39b0757b5c11b9f5bccdfad2c9559f0804b5e4
ISB	Interfaces/IStableCollBorrowerOperations.sol	ee7b63f22e8ab3b2da38fdfddddc0b1a0c7860d106c5f07204c5f3dc605b6ccf
ISM	Interfaces/IStableCollTroveManager.sol	3ad4088a6271ed21fd20ff6c4b7f6fe467badc2d858f59d604e892340f1357e5
ITC	Interfaces/ITellorCaller.sol	3e743e3da65e5a3333140807fee464bd32655b6bf945436c1fa78709ae9d9a63
ITM	Interfaces/ITroveManager.sol	1edab4efc913a9344a6e9430dccd394b393bb28dc1e985067dcfead5b76a37a9
ADL	LPRewards/Dependencies/Address.sol	05a6a49cf9cc82c283f36d65e20f1e16fbf850588cb3312ad3c52f15eb4b6a12

ID	File	SHA256 Checksum
SER	LPRewards/Dependencies/SafeERC20.sol	2bd09642c108993133303aa419f9edef8e94bcbab44411208e7e9e3da014c639
ILW	LPRewards/Interfaces/ILPTokenWrapper.sol	bcfedabf6b5ae1487f11d40856510c1464068c720bb4ff42c22f6ea0bf311b6a
IUI	LPRewards/Interfaces/IUnipool.sol	a0a344fec8abc86cfd17606b31d333fc9fd45d038faed69ef78b60eebf6d0d0e
USD	LPRewards/USDCUnipool.sol	fed3631a98521b2faaa318ae49dabfdd1b77dca08d475dd75ea46e46512336d3
ULP	LPRewards/Unipool.sol	d1ddbfe704546ac7230deca7c0d723be2885d616ca6552c9d64bb1af3e017476
CIL	LQTY/CommunityIssuance.sol	2a871920d5f6abdf829e4d45400cde8bf70f602a20dafbca2a18b12acee7ed9e
LQY	LQTY/LQTYStaking.sol	e88a1682adaa492aa56e70fcd82fc4277fd3a4f0b7ccea3b6447fbbc2db3ea70
LQL	LQTY/LQTYToken.sol	496f9c2d13ebaa20aa472d9fac77d71670ad2f7d0b66f6aec9b09f839add7c4b
LCL	LQTY/LockupContract.sol	9b5f1ae796c6e2b716f24967e7f7c4e5212648e06e8da04a43deb6fed330bdf4
LCF	LQTY/LockupContractFactory.sol	dd38a2cb3a50b13dfdd762f6a814af93f1945c2b98898a7fcad88867fa566612
SCA	StableColl/StableCollActivePool.sol	c1c1934ac82ac1c6665579008c53b9312e17c35ec00918c0532cad00e6a21b12
SCB	StableColl/StableCollBorrowerOperations.sol	93b98384a79e4c16b4830dd94fda5c88b3ce6ce7f2ae7fbf80ae627358f68277
SCT	StableColl/StableCollTroveManager.sol	150634df73df9918fcf4e3e31fd663adea0891872973c439f7c0224170b67e7b
APD	ActivePool.sol	2192de1cac7a5691bd2b38bea3a5bea1dac311ebb39695e01810eaf5a17de393
BOD	BorrowerOperations.sol	057091447ebc9a450b6992d7b2f06898443da055a25e3bf0d8acb778455aa433
CSP	CollSurplusPool.sol	8495af1e96df6a67bd22a0aa28573b1018f9368e70ec1bd72691ce2ddf885c05

ID	File	SHA256 Checksum
DPD	DefaultPool.sol	d373a5e8fd9648bcca5aab246217020d1a10f62b7ef7c3ec768399e22d3fe8ab
GPD	GasPool.sol	9aab938a8b7985e223e5e0d13bbd720d2a0e365706dd789668968d67ed8e9581
HHD	HintHelpers.sol	11ebcb20cf2f0634754596d1dc935f937387b0e3fed21c2d9aeccd159789104f
LUS	LUSDToken.sol	f3e167ab0f69db8d298ad768e7d6965c4e97434fccdf57a221481e7571093402
MTG	MultiTroveGetter.sol	49e0dd1154d00683c468515fa53184efa5ed29f6c26c4af90ee7d565366b935d
PFD	PriceFeed.sol	208345c3a762beeed64cdf2e8dbb9c81391573e2c5e9eaeae4ef3e8b0fe1f3ee
STD	SortedTroves.sol	480a8a4bb3f396e9279d9025843c388fd9d625749e780373d103290d17fbafc0
SPD	StabilityPool.sol	46d034b6e8c4b82f151845845809b73b316d7990cb033558f1a1f28329d67138
TMD	TroveManager.sol	d15a99650d68cdf68e15c2dce772ef64750fe4521cca659b7d90bc8e8436b8f9

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**17** (30.36%)
🟨	**Medium**	**4** (7.14%)
🟨	**Minor**	**9** (16.07%)
🟦	**Informational**	**26** (46.43%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
APD-01	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
BOD-01	Visibility Specifiers Missing	Language Specific	● Informational	✓ Resolved
BOD-02	Variable Declare as Immutable or Constant	Volatile Code	● Informational	Acknowledged
BOD-03	Optimizable Usage of `uint`	Gas Optimization	● Informational	Acknowledged
BOD-04	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	✓ Resolved
BOD-05	Potentially Unable To Close Trove When Holding Less Than Minted Amount	Volatile Code, Data Flow	● Informational	✓ Resolved
BOD-06	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
CIL-01	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	✓ Resolved
CIL-02	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
CIL-03	Unchecked Value of ERC-20 `transfer()` Call	Volatile Code	● Minor	Acknowledged
CSP-01	Missing Input Validation	Volatile Code	● Minor	✓ Resolved

ID	Title	Category	Severity	Status
CSP-02	Variable Could Be Declared as Constant	Gas Optimization	● Informational	ⓘ Acknowledged
CSP-03	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
DPD-01	Missing Input Validation	Volatile Code	● Minor	✓ Resolved
DPD-02	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
HHD-01	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
LBD-01	Redundant Named Return Variables	Gas Optimization	● Informational	✓ Resolved
LBD-02	Return Variable Utilization	Gas Optimization	● Informational	✓ Resolved
LCF-01	Unused Variable	Gas Optimization	● Informational	✓ Resolved
LCF-02	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
LCL-01	Unchecked Value of ERC-20 `transfer()` Call	Volatile Code	● Minor	✓ Resolved
LCL-02	Lack of input validation	Volatile Code	● Informational	✓ Resolved
LQL-01	Lack of input validation	Volatile Code	● Informational	✓ Resolved
LQL-02	Initial Token Distribution	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
LQL-03	Compares to a Boolean Constant	Gas Optimization	● Informational	✓ Resolved
LQL-04	Lack of sanity checks on ecrecover	Volatile Code	● Medium	✓ Resolved
LQL-05	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
LQL-06	Mismatch of Function Name and Implementation	Logical Issue	● Medium	✓ Resolved
LQL-07	Typo	Language Specific	● Informational	ⓘ Acknowledged

ID	Title	Category	Severity	Status
LQY-01	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	✓ Resolved
LQY-02	Unchecked Value of ERC-20 `transfer()` Call	Volatile Code	● Minor	✓ Resolved
LQY-03	Compares to a Boolean Constant	Gas Optimization	● Informational	✓ Resolved
LQY-04	Incorrect Naming Convention Utilization	Coding Style	● Informational	✓ Resolved
LQY-05	Centralization Risk	**Centralization / Privilege**	● **Major**	✓ Resolved
LQY-06	Typo	Coding Style	● Informational	✓ Resolved
LUS-01	Susceptible to Signature Malleability	Volatile Code	● Medium	ⓘ Acknowledged
LUS-02	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	ⓘ Acknowledged
LUS-03	Missing Validation Against Restricted Addresses in `mint()`	Volatile Code	● Minor	ⓘ Acknowledged
LUS-04	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
PFD-01	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
SPD-01	Optimizable Usage of `uint`	Gas Optimization	● Informational	ⓘ Acknowledged
SPD-02	Missing Input Validation	Volatile Code	● Minor	✓ Resolved
SPD-03	Checks-effect-interaction Pattern Violation	Logical Issue	● Minor	ⓘ Acknowledged
SPD-04	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
TCD-01	Missing Input Validation	Volatile Code	● Minor	✓ Resolved
TMD-01	Visibility Specifiers Missing	Language Specific	● Informational	ⓘ Acknowledged
TMD-02	Optimizable Usage of `uint`	Gas Optimization	● Informational	ⓘ Acknowledged

ID	Title	Category	Severity	Status
TMD-03	Missing Emit Events	Coding Style	● Informational	Resolved
TMD-04	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	Acknowledged
TMD-05	Logic Flaw for Low LQTY Pool Participation	Volatile Code	● Medium	Acknowledged
TMD-06	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
TMD-07	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
ULP-01	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	Resolved
ULP-02	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
USD-01	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	Resolved
USD-02	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved

APD-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/ActivePool.sol (8663016): 43	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

BOD-01 | Visibility Specifiers Missing

Category	Severity	Location	Status
Language Specific	● Informational	projects/xDollar-Finance/BorrowerOperations.sol (8663016): 23~27	✓ Resolved

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

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

BOD-02 | Variable Declare as Immutable or Constant

Category	Severity	Location	Status
Volatile Code	● Informational	projects/xDollar-Finance/BorrowerOperations.sol (8663016): 33	ⓘ Acknowledged

Description

The `collToken` is set to `_wethTokenAddress` in the `setAddresses()` function. If only WETH is intended as collToken than it should be declared constant as we know the WETH address beforehand

Recommendation

We would recommend adding constant or immutable to linked variable to avoid any confusion what collateral token system is using.

BOD-03 | Optimizable Usage of `uint`

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/BorrowerOperations.sol (8663016): 43~68	ⓘ Acknowledged

Description

The uint in the linked structs is defaulting to uint256 but if there is no need of using 256bit for certain variables, there's a way to safe on gas by tight-packing the variables and using lower bits version of uint like uint128. This would safe on storage space and thus saving on gas.

Recommendation

We would recommend to reconsider usage of default uint in the linked structs and possibly using lower type bits of uint for tight-packing of variables.

BOD-04 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/BorrowerOperations.sol (8663016): 117, 184, 281, 302	✓ Resolved

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

BOD-05 | Potentially Unable To Close Trove When Holding Less Than Minted Amount

Category	Severity	Location	Status
Volatile Code, Data Flow	● Informational	projects/xDollar-Finance/BorrowerOperations.sol (8663016): 337	✓ Resolved

Description

Does any user who opened the trove and sent some LUSD to Stability Pool, can close the trove? Or will the transaction fail if a user don't have minted amount of LUSD when closing trove?

Alleviation

`[xdollar.fi team]`: If user don't have the amount of xUSD that is larger or equal to the debt, the closing trove txn fails.

BOD-06 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/BorrowerOperations.sol (8663016): 99	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

CIL-01 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/LQTY/CommunityIssuance.sol (8663016): 83, 116	✓ Resolved

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

CIL-02 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/LQTY/CommunityIssuance.sol (8663016): 66	Partially Resolved

Description

In the contract `CommunityIssuance`, the role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

CIL-03 | Unchecked Value of ERC-20 `transfer()` Call

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/LQTY/CommunityIssuance.sol (8663016): 124	ⓘ Acknowledged

Description

The linked `transfer()` invocations do not check the return value of the function call which should yield a `true` result in case of 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 openzeppelin implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

It is recommended to use SafeERC20 or make sure that the value returned from 'transfer()' is checked.

CSP-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/CollSurplusPool.sol (8663016): 44	✓ Resolved

Description

The assigned values to address type variables `_collTokenAddress` should be verified as non-zero values to prevent error.

Recommendation

Check that the addresses are not zero in the constructor, like below:

```
checkContract(_collTokenAddress)
```

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

CSP-02 | Variable Could Be Declared as Constant

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/CollSurplusPool.sol (8663016): 27	ⓘ Acknowledged

Description

`collToken` is expected to be WETH but it still accepts custom ERC20 address in setAddresses function. It should be made constant as WETH token address is known before deployment

Recommendation

We advise to change collToken to be a constant with WETH address

CSP-03 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/xDollar-Finance/CollSurplusPool.sol (8663016): 40	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

DPD-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/DefaultPool.sol (8663016): 40	✓ Resolved

Description

The assigned values to address type variables `_collTokenAddress` should be verified as non-zero values to prevent error.

Recommendation

Check that the addresses are not zero in the constructor, like below:

```
checkContract(_collTokenAddress)
```

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

DPD-02 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/DefaultPool.sol (8663016): 37	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

HHD-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/HintHelpers.sol (8663016): 24	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

LBD-01 | Redundant Named Return Variables

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/Dependencies/LiquityBase.sol (8663016): 67, 74, 78, 85	Resolved

Description

The linked code segments contain named return variables for functions that do not utilize them.

Recommendation

We advise the team to either remove or properly utilize the name variables.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LBD-02 | Return Variable Utilization

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/Dependencies/LiquityBase.sol (8663016): 95	✓ Resolved

Description

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation

We advise that the linked variables are either utilized or omitted from the declaration.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LCF-01 | Unused Variable

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/LQTY/LockupContractFactory.sol (8663016): 32	✓ Resolved

Description

The state variable `SECONDS_IN_ONE_YEAR` is never used.

Recommendation

We recommend removing the unused state variable.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LCF-02 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/LQTY/LockupContractFactory.sol (8663016): 45	Partially Resolved

Description

In the contract `LockupContractFactory`, the role `owner` has the authority over the following functions:

- `setLQTYTokenAddress()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the LQTY token address of the project, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

LCL-01 | Unchecked Value of ERC-20 `transfer()` Call

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/LQTY/LockupContract.sol (8663016): 68	✓ Resolved

Description

The linked `transfer()` invocations do not check the return value of the function call which should yield a `true` result in case of 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 openzeppelin implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

It is recommended to use SafeERC20 or make sure that the value returned from 'transfer()' is checked.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LCL-02 | Lack of input validation

Category	Severity	Location	Status
Volatile Code	● Informational	projects/xDollar-Finance/LQTY/LockupContract.sol (8663016): 43~44	✓ Resolved

Description

The assigned values to address type variables `_lqtyTokenAddress` and `_beneficiary` should be verified as non-zero values to prevent error.

Recommendation

Check that the addresses are not zero in the constructor, like below:

```
require(_lqtyTokenAddress != address(0),"_lqtyTokenAddress is zero address!");
require(_beneficiary != address(0),"_beneficiary is zero address!");
```

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQL-01 | Lack of input validation

Category	Severity	Location	Status
Volatile Code	● Informational	projects/xDollar-Finance/LQTY/LQTYToken.sol (8663016): 114~120	✓ Resolved

Description

The assigned values to address type variables `_initialSetupAddress`, `_lpRewardsAddress`, `_multisigAddress`, `_ecosystemVestingAddress`, `_teamVestingAddress`, `_partnerVestingAddress` and `_treasuryAddress` should be verified as non-zero values to prevent error.

Recommendation

Check that the addresses are not zero in the constructor, like below:

```
require(_initialSetupAddress != address(0),"_initialSetupAddress is zero address!");
require(_lpRewardsAddress != address(0),"_lpRewardsAddress is zero address!");
require(_multisigAddress != address(0),"_multisigAddress is zero address!");
require(_ecosystemVestingAddress != address(0),"_ecosystemVestingAddress is zero
address!");
require(_teamVestingAddress != address(0),"_teamVestingAddress is zero address!");
require(_partnerVestingAddress != address(0),"_partnerVestingAddress is zero address!");
require(_treasuryAddress != address(0),"_treasuryAddress is zero address!");
```

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQL-02 | Initial Token Distribution

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/xDollar-Finance/LQTY/LQTYToken.sol (8663016): 144~160	ⓘ Acknowledged

Description

When the contract is deployed, the following actions will be executed:

- 1 million tokens are sent to the `_initialSetupAddress`.
- 10 million tokens are sent to the `_communityIssuanceAddress`.
- 0.05 million tokens are sent to the `_lpRewardsAddress`.
- 15 million tokens are sent to the `_ecosystemVestingAddress`.
- 7.5 million tokens are sent to the `_teamVestingAddress`.
- 4 million tokens are sent to the `_partnerVestingAddress`.
- 8.95 million tokens are sent to the `_treasuryAddress`.
- 17.5 million tokens are sent to the `_multisigAddress`.

This could be a centralization risk as the `owner` can distribute tokens without obtaining the consensus of the community.

Recommendation

We recommend the team to be transparent regarding the initial token distribution process.

Alleviation

`[xdollar.fi team]`: By design, will disclose all the information to community

LQL-03 | Compares to a Boolean Constant

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/LQTY/LQTYToken.sol (8663016): 353, 341	✓ Resolved

Description

Compares to a boolean constant. Example:

```
if (collTokenAddresses[collTokens[i].tokenAddress] == true) {
```

Recommendation

Consider removing the equality to the boolean constant.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQL-04 | Lack of sanity checks on ecrecover

Category	Severity	Location	Status
Volatile Code	● Medium	projects/xDollar-Finance/LQTY/LQTYToken.sol (8663016): 276	✓ Resolved

Description

Code lack sanity check for ecrecover. Raw ecrecover function will yield the zero address for any incorrect signature.

Recommendation

We would recommend adding require statement to check if the returned address from `ecrecover` isn't 0x0.

We would suggest using OpenZeppelin's ECDSA Library contract as it implements correctly recovering the address from the signature.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQL-05 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/xDollar-Finance/LQTY/LQTYToken.sol (8663016): 235, 239, 243	Partially Resolved

Description

In the contract `LQTYToken`, the role `owner` has the authority over the following functions:

- `addCommunityIssuanceAddress(address newCommunityIssuanceAddress)`
- `removeCommunityIssuanceAddress(address newCommunityIssuanceAddress)`
- `transferToNewCommunityIssuanceContract(address newCommunityIssuanceAddress, uint256 amount)`

Any compromise to the `owner` account may allow the hacker to take advantage of this and do the following:

- add community issuance address.
- remove community issuance address.
- transfer tokens to new community issuance address.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

LQL-06 | Mismatch of Function Name and Implementation

Category	Severity	Location	Status
Logical Issue	● Medium	projects/xDollar-Finance/LQTY/LQTYToken.sol (8663016): 328~330	✓ Resolved

Description

The `_isFirstYear()` function does not check whether the time passed a year or not, but rather checks whether the time passed half a year or not.

Recommendation

We recommend making the function name and implementation consistent.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQL-07 | Typo

Category	Severity	Location	Status
Language Specific	● Informational	projects/xDollar-Finance/LQTY/LQTYToken.sol (8663016): 56	ⓘ Acknowledged

Description

```
string constant internal _NAME = "testDollar";
```

The above statement obviously uses a name in the test environment.

Recommendation

Consider updating all the constant parameters from the testing environment to the production environment.

LQY-01 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/LQTY/LQTYStaking.sol (8663016): 168, 220	✓ Resolved

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQY-02 | Unchecked Value of ERC-20 `transfer()` Call

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/LQTY/LQTYStaking.sol (8663016): 200, 247, 255	✓ Resolved

Description

The linked `transfer()` invocations do not check the return value of the function call which should yield a `true` result in case of 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 openzeppelin implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

It is recommended to use SafeERC20 or make sure that the value returned from 'transfer()' is checked.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQY-03 | Compares to a Boolean Constant

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/LQTY/LQTYStaking.sol (8663016): 175, 227, 332, 358, 363, 369, 402~405, 387~390	✓ Resolved

Description

Compares to a boolean constant. Example:

```
175  if (collTokenAddresses[collTokens[i].tokenAddress] == true) {
```

Recommendation

Consider removing the equality to the boolean constant.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

LQY-04 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/LQTY/LQTYStaking.sol (8663016): 131	✓ Resolved

Description

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant *meta* information that would otherwise not be immediately available.

Solidity defines a naming convention that should be followed.

- Contracts and libraries should be named using the CapWords style.
- Structs should be named using the CapWords style.
- Events should be named using the CapWords style.
- Functions should use mixedCase.
- Function arguments should use mixedCase.
- Local and State Variable Names should use mixedCase.
- Constants should be named with all capital letters with underscores separating words.
- Enums, in the style of simple type declarations, should be named using the CapWords style.

Reference: https://docs.soliditylang.org/en/latest/style-guide.html#naming-conventions

Recommendation

We advise the client to follow the Solidity naming convention. The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

```
function removeAddressesForColl(
    address _collTokenAddress,
    address _troveManagerAddress,
    address _borrowerOperationsAddress,
    address _activePoolAddress
)
```

Alleviation

`[xdollar.fi team]`: Obsolete - Filed removed

LQY-05 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/xDollar-Finance/LQTY/LQTYStaking.sol (8663016): 82, 90, 98 ~104, 131~137, 271, 283	✓ Resolved

Description

In the contract `LQTYStaking`, the role `owner` has the authority over the following functions:

- `setXdoTokenAddress(address _xdoTokenAddress)`
- `setXUSDTokenAddress(address _xUSDTokenAddress)`
- `addAddressesForColl(address _collTokenAddress, address _troveManagerAddress, address _borrowerOperationsAddress, address _activePoolAddress)`
- `RemoveAddressesForColl(address _collTokenAddress, address _troveManagerAddress, address _borrowerOperationsAddress, address _activePoolAddress)`

Any compromise to the `owner` account may allow the hacker to take advantage of this and do the following:

- set xdo token address.
- set xusdt token address.
- add addresses for coll
- remove addresses for coll

In the contract `LQTYStaking`, the accounts in the `troveManagerAddresses` have the authority over the following function:

- `increaseF_Coll(address collTokenAddress, uint256 _CollFee)`

Any compromise to the accounts in the `troveManagerAddresses` may allow the hacker to take advantage of this and do the following:

- `increaseF_Coll(address collTokenAddress, uint256 _CollFee)`

In the contract `LQTYStaking`, the accounts in `borrowerOperationsAddresses` have the authority over the following function:

- increase `CollFeePerLQTYStaked`

Any compromise to the accounts in the `borrowerOperationsAddresses` may allow the hacker to take advantage of this and do the following:

- increase `XUSDFeePerLQTYStaked`

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: Obsolete - Filed removed

LQY-06 | Typo

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/LQTY/LQTYStaking.sol (8663016)	✓ Resolved

Description

There are some comments mentioned `LUSD` instead of `XUSD`.

Alleviation

`[xdollar.fi team]`: Obsolete - Filed removed

LUS-01 | Susceptible to Signature Malleability

Category	Severity	Location	Status
Volatile Code	● Medium	projects/xDollar-Finance/LUSDToken.sol (8663016): 258	ⓘ Acknowledged

Description

The signature malleability is possible within the Elliptic Curve cryptographic system. An Elliptic Curve is symmetric on the X-axis, meaning two points can exist with the same `X` value. In the `r`, `s` and `v` representation this permits us to carefully adjust `s` to produce a second valid signature for the same `r`, thus breaking the assumption that a signature cannot be replayed in what is known as a replay-attack.

Recommendation

To fix this we would recommend adding check from EIP-2, point 2 (https://eips.ethereum.org/EIPS/eip-2), and also check for the v value to ensure the off-chain library is properly used. Look into `ecrecoverFromSig` function from SWC-117 (https://swcregistry.io/docs/SWC-117).

OpenZeppelin ECDSA library contract contains proper implementation for recovering address from the signature that isn't prone to signature malleability. We suggest importing that and using it in the contract.

Alleviation

`[CertiK]`: `ecrecover()` is commonly adopted in the Elliptic Curve cryptographic system, and this function also has security concern due to its implementation. We would like to recommend to adopt the openzeppelin library https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol for ECDSA functionalities.

Reference:

- https://docs.openzeppelin.com/contracts/2.x/api/cryptography
- http://coders-errand.com/malleability-ecdsa-signatures/

LUS-02 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/LUSDToken.sol (8663016): 292~293, 301, 309, 321~322	ⓘ Acknowledged

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function

LUS-03 | Missing Validation Against Restricted Addresses in `mint()`

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/LUSDToken.sol (8663016): 136~139	ⓘ Acknowledged

Description

LUSD Token impose restriction to certain addresses during transfer of the token. This set of restriction is not checked during mint function.

Recommendation

We would recommend to add `_requireValidRecipient` to the `mint()` function to impose the same restrictions as for the `transfer` functions.

LUS-04 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/LUSDToken.sol (8663016): 87, 112	Partially Resolved

Description

In the contract `LUSDToken`, the role `owner` has the authority over the following function:

- `addAddressesForColl()`
- `removeAddressesForColl()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and change the status of the following three sensitive variables.

- `troveManagerAddresses[_troveManagerAddress]`
- `stabilityPoolAddresses[_stabilityPoolAddress]`
- `borrowerOperationsAddresses[_borrowerOperationsAddress]`

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

PFD-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/PriceFeed.sol (8663016): 87	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

SPD-01 | Optimizable Usage of `uint`

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/StabilityPool.sol (8663016): 173~189	ⓘ Acknowledged

Description

The uint in the linked structs is defaulting to uint256 but if there is no need of using 256bit for certain variables, there's a way to safe on gas by tight-packing the variables and using lower bits version of uint like uint128. This would safe on storage space and thus saving on gas.

Recommendation

We would recommend to reconsider usage of default uint in the linked structs and possibly using lower type bits of uint for tight-packing of variables.

SPD-02 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/StabilityPool.sol (8663016): 283	✓ Resolved

Description

The assigned values to address type variables `_collTokenAddress` should be verified as non-zero values to prevent error.

Recommendation

Check that the addresses are not zero in the constructor, like below:

```
checkContract(_collTokenAddress)
```

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

SPD-03 | Checks-effect-interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	● Minor	projects/xDollar-Finance/StabilityPool.sol (8663016): 338, 385, 410, 413, 364, 366	ⓘ Acknowledged

Description

In the function `provideToSP()` and `withdrawFromSP()`, state variables are changed after the external transfer function call.

Recommendation

It is recommended to follow checks-effects-interactions pattern for cases like this.

It shields public functions from re-entrancy attacks. It's always a good practice to follow this pattern. `checks-effects-interaction` pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

We also recommend the client to consider the below code snippets as references and do adjustments based on the project needs..

```
function provideToSP(uint _amount, address _frontEndTag) external override {
    ...
    // Update front end stake
    uint compoundedFrontEndStake = getCompoundedFrontEndStake(frontEnd);
    uint newFrontEndStake = compoundedFrontEndStake.add(_amount);
    _updateFrontEndStakeAndSnapshots(frontEnd, newFrontEndStake);
    emit FrontEndStakeChanged(frontEnd, newFrontEndStake, msg.sender);

    uint newDeposit = compoundedLUSDDeposit.add(_amount);
    _sendLUSDtoStabilityPool(msg.sender, _amount);

    _updateDepositAndSnapshots(msg.sender, newDeposit);
    emit UserDepositChanged(msg.sender, newDeposit);
    ...
}
```

```
function withdrawFromSP(uint _amount) external override {
    ...
    // Update front end stake
    uint compoundedFrontEndStake = getCompoundedFrontEndStake(frontEnd);
    uint newFrontEndStake = compoundedFrontEndStake.sub(LUSDtoWithdraw);
    _updateFrontEndStakeAndSnapshots(frontEnd, newFrontEndStake);
    emit FrontEndStakeChanged(frontEnd, newFrontEndStake, msg.sender);

    uint newDeposit = compoundedLUSDDeposit.sub(LUSDtoWithdraw);
    _sendLUSDToDepositor(msg.sender, LUSDtoWithdraw);

    // Update deposit
    _updateDepositAndSnapshots(msg.sender, newDeposit);
    emit UserDepositChanged(msg.sender, newDeposit);
    ...
}
```

SPD-04 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/StabilityPool.sol (8663016): 275	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

TCD-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	projects/xDollar-Finance/Dependencies/TellorCaller.sol (8663016): 23	✓ Resolved

Description

The given input is missing the check for the non-zero address.

Recommendation

We advise adding the check for the passed-in values to prevent unexpected error:

```
constructor (address _tellorMasterAddress) public {
     require(_tellorMasterAddress != address(0), "_tellorMasterAddress is address(0)");
     tellor = ITellor(_tellorMasterAddress);
}
```

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

TMD-01 | Visibility Specifiers Missing

Category	Severity	Location	Status
Language Specific	● Informational	projects/xDollar-Finance/TroveManager.sol (8663016): 26~28	ⓘ Acknowledged

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.

TMD-02 | Optimizable Usage of `uint`

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/xDollar-Finance/TroveManager.sol (8663016): 124~198	ⓘ Acknowledged

Description

The uint in the linked structs is defaulting to uint256 but if there is no need of using 256bit for certain variables, there's a way to safe on gas by tight-packing the variables and using lower bits version of uint like uint128. This would safe on storage space and thus saving on gas.

Recommendation

We would recommend to reconsider usage of default uint in the linked structs and possibly using lower type bits of uint for tight-packing of variables.

TMD-03 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/TroveManager.sol (8663016): 277	✓ Resolved

Description

"The function that affects the status of sensitive variables should be able to emit events as notifications to collateral token set.

Recommendation

Consider adding events for sensitive actions, and emit them in the function.

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

TMD-04 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/TroveManager.sol (8663016): 411, 955, 1195, 1243, 1306, 1312, 1366, 1441	ⓘ Acknowledged

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function

TMD-05 | Logic Flaw for Low LQTY Pool Participation

Category	Severity	Location	Status
Volatile Code	● Medium	projects/xDollar-Finance/TroveManager.sol (8663016): 2	ⓘ Acknowledged

Description

Fee rate can be inflated by the attacker but the attack is limited by the cost of their own operations. When the base fee is incremented, the new rate is applied to their redeems, and the fees are then distributed among LQTY pool stakers.

In the attack scenario, assuming the LQTY circulating supply took the dominating proportion of the stake in the LQTY pool, the cost of moving the fees could be received by the attackers. An attacker could utilize a flash loan and stake it to recover the paid fees, then redeem LUSD and ultimately increase the base rate.

Recommendation

We would recommend a faster base fee decay speed in order to make the attack even more expensive for an attacker.

We advise the client to revisit the design and implementation of the staking module and set a faster base fee decay speed, which can increase the cost of attack and therefore prevent the attack happen.

TMD-06 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/xDollar-Finance/TroveManager.sol (8663016): 308	Partially Resolved

Description

In the contract `TroveManager.sol` in the commit 934f7defbc85492e098ff4ba423536c430fa6e65, the role `feeAdminAddress` has the authority over the following function:

- `setRedemptionFeePoolParams()`

Any compromise to the `feeAdminAddress` account may allow the hacker to take advantage of this and modify the `redemptionFeePoolAddress` and `redemptionFeePoolRate` and thus break the entire staking and project system.

Recommendation

We advise the client to carefully manage the `feeAdminAddress` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

TMD-07 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	projects/xDollar-Finance/TroveManager.sol (8663016): 236	Partially Resolved

Description

The role `owner` has the authority over the following function:

- `setAddresses()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the addresses of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

ULP-01 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/LPRewards/Unipool.sol (8663016): 195~197, 238	✓ Resolved

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

ULP-02 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/xDollar-Finance/LPRewards/Unipool.sol (8663016): 95	Partially Resolved

Description

In the contract `USDCUnipool` and `Unipool`, the role `owner` has the authority over the following function:

- `setParams()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the parameters of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

USD-01 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/xDollar-Finance/LPRewards/USDCUnipool.sol (8663016): 197~199, 213, 240	✓ Resolved

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function

Alleviation

`[xdollar.fi team]`: The client heeded the advice and fixed the issue in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

USD-02 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/xDollar-Finance/LPRewards/USDCUnipool.sol (8663016): 97	Partially Resolved

Description

In the contract `USDCUnipool` and `Unipool`, the role `owner` has the authority over the following function:

- `setParams()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and set the parameters of the projects, which may cause the break of the entire system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

`[xdollar.fi team]`: The client implemented `notLocked` modifier to add additional control layer to the sensitive function in the commit e4add2bce5c8914066bdaa4ae0c98f110ec4d01a

`[CertiK]`: The modifier `notLocked` works when the `unlockFunction()`/`lockFunction()` are properly invoked by the role `owner`

Appendix

Finding Categories

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.

Gas Optimization

Gas Optimization findings 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.

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.

Volatile Code

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

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

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

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

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

About

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.