February 17, 2023

Security Audit for

LevelX

DEFIYIELD

Content

1. About DEFIYIELD

DeFiYield is one of the leading smart contract auditing providers focused on checking security of DeFi projects and the world's only crypto asset management dashboard featuring user protection against interactions with risky smart contracts.

Our first audits were conducted back in July 2020, shortly after the yield farming industry boomed, bringing impressive return opportunities for users. At the same time, scams happened every day, and users were not protected against them in any way. No one performed yield-farming-focused audits at the time and a lot of projects were launching without even doing proper internal audits. This is why DeFiYield took the lead and has been developing and pushing security standards in the community since then.

2. Audit Scope

2.1 Project

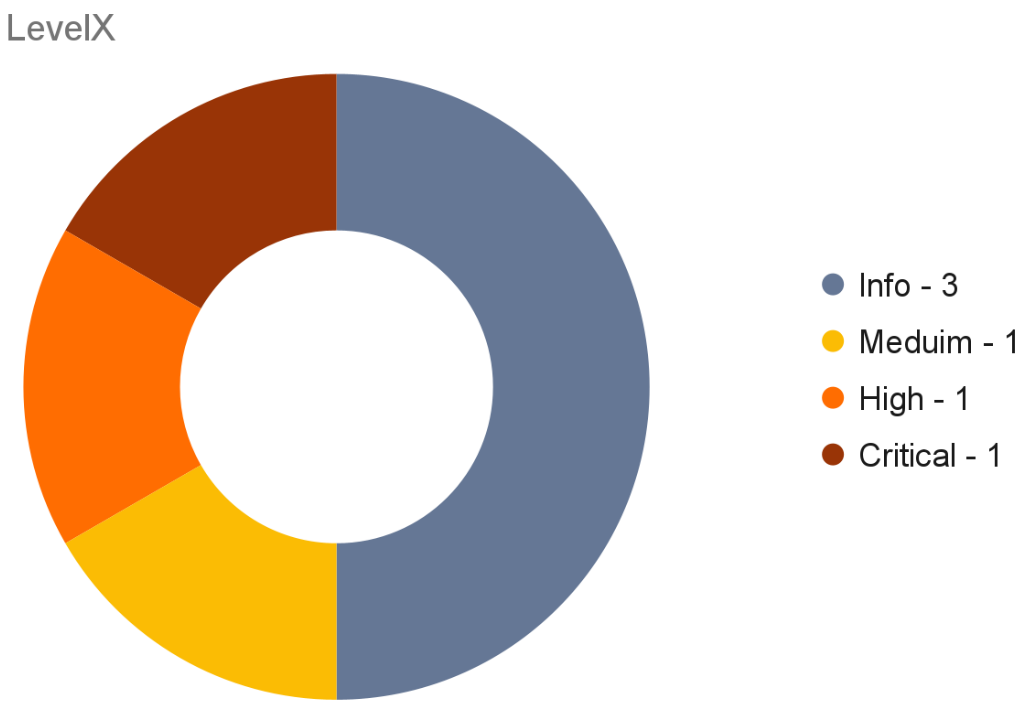
Project Name	LevelX
Blockchain	-
Language	Solidity
About	LevelX is an ERC20-based protocol that includes a rebase ERC20 token contract, reflection, and leveling functionality. Users can earn reward tokens through this system, and there is an additional ERC20 token-based staking contract available.

2.1 Contracts

4 (four) smart contracts from the GitHub commit: 0c4dbcf14088157039cac0be0e9458c30decd256 were analyzed for the presence of code vulnerabilities:

Solidity

- LevelXToken.sol
- LevelXWrapperToken.sol
- LevelXSeed.sol
- LevelXExchange.sol



Total Issues Found: 6

Related Smart Contract	ID	Issue name	Category	Severity	Status
LevelXToken.sol	209	Sell/Buy fee	DeFi-Specific	● Critical	Acknowledged
	198-d	Overprivileged role	DeFi-Specific	● High	Acknowledged
	179	Costly operations in a loop	Solidity Coding Best Practices	● Medium	Acknowledged
	-	Code styling	Solidity Coding Best Practices	● Info	Acknowledged

LevelXWrapperToken .sol	-	Code styling	Solidity Coding Best Practices	◉ Info	Acknowledged
LevelXSeed.sol	-	Code styling	Solidity Coding Best Practices	◉ Info	Acknowledged
LevelXExchange.sol	-	-	-		

✱ Acknowledged means that the issue was addressed to the project's team, but they don't define it as a problem:
a) the code was designed in the way intentionally;
b) the code doesn't cause any issues for security and functioning of the smart contract.

4. Methodology

4.1 Auditing approach

Both manual and automated audit techniques are applied to all audited contracts.

For automated analysis, we use DeFiYield' scanner. The Scanner was designed by DeFiYield for fast, automated checks of common smart contract weaknesses. Moreover, it detects DeFi-specific smart contract vulnerabilities and malicious functions that are the most frequent reasons for rug pulls and hacker attacks.

All issues found by the scanner get manually reviewed and validated with line-by-line code analysis.

Issues covered:

- Unverified contracts;
- Unlimited minting to a malicious destination;
- Dangerous token migration;
- Pausing token transfers anytime for unlimited period;
- Pausing token transfer for limited period (defined in the contract);
- Pausing funds withdrawals (centralized pausing for any funds withdrawals);
- Pausing funds withdrawals with emergency withdrawal available;
- Proxy patterns;
- Funds lock with centralized control;
- State variables shadowing;
- Functions allowing anyone to destruct the contract;
- Uninitialized state variables;

- Uninitialized storage variables;
- Unprotected upgradeable contract;
- Functions that send Ether to arbitrary destination;
- Controlled delegatecall destination;
- Reentrancy vulnerabilities (theft of ethers);
- Unchecked tokens transfer;
- Weak PRNG;
- Detect dangerous enum conversion;
- Incorrect ERC20 interfaces;
- Incorrect ERC721 interfaces;
- Dangerous strict equalities;
- Contracts that lock ether;
- State variables shadowing from abstract contracts;
- Unused write;
- Misuse of Boolean constant;
- Constant functions using assembly code;
- Constant functions changing the state;
- Imprecise arithmetic operations order;
- Reentrancy vulnerabilities (no theft of ethers);
- Reused base constructor;
- Dangerous usage of tx.origin;
- Unchecked low-level calls;
- Unchecked send;
- Uninitialized local variables;
- Unused return values;
- Modifiers that can return the default value;
- Local variables used prior their declaration;
- Constructor called not implemented;
- Missing Events Access Control;
- Missing Events Arithmetic;
- Dangerous unary expressions;

- Missing Zero Address Validation;
- Benign reentrancy vulnerabilities;
- Reentrancy vulnerabilities leading to out-of-order Events;
- Dangerous usage of block.timestamp;
- Assert state change;
- Comparison to boolean constant;
- Un-indexed ERC20 event parameters;
- Function initializing state variables;
- Missing inheritance;
- Conformity to Solidity naming conventions;
- Incorrect Solidity version;
- Unimplemented functions;
- Unused state variables;
- Costly operations in a loop;
- Functions that are not used;
- Reentrancy vulnerabilities through send and transfer;
- Variable names are too similar;
- Conformance to numeric notation best practices;
- State variables that could be declared constant;
- Public function that could be declared external;
- Contract are different from filenames
- Funds lock with centralized control;
- Suspicious functions;
- Insufficient timelock for important contract changes;
- Overprivileged role:
    a. The privileged EOA can call a function that allows to withdraw all staked in the contract funds to a needed address;
    b. The privileged EOA can change address of token reward distribution;
    c. The privileged EOA can change the location of staked user funds.
- Unrestricted fee setting;

    a. withdrawal fee can be set up to 100%;
    b. user reward fee can be decreased;
    c. Team reward increased without any limitations in centralized way;
    d. Other protocol fees with unexpected security consequences).
- Using a singular exchange as a price source;
- Insufficient Validation;
- Uncollateralized share token minting;
- Unprotected function;
- Custom token standard;
- Logic bug;
- Missing requirement;
- Blacklisting;
- ERC20 transfer limit;
- Token drain vulnerability through ERC20 approval;
- Custom standard ERC20 functions;
- Blocking loop;
- Cooldown time on trading;
- Centralized token balance modification;
- Suspicious approval in Masterchef;
- Suspicious approval in token pair;
- Approval Objects Restriction;
- Payable function using delegatecall inside a loop;
- msg.value inside a loop;
- Arbitrary ERC20 send with permit;
- Privileged NFT transferFrom without approval;

4.2 Issue Classification by Severity

◉ **Critical**	Issues that can directly cause a loss of underlying funds with high probability. These issues must be removed ASAP.
◉ **High**	There is a possibility of negative impacts on funds managed by the smart contract when certain conditions come into action.
◉ **Medium**	Issues that affect contract functionality without causing financial losses, must be addressed by the developers.
◉ **Low**	The issues must be addressed to follow the best SC coding practice.
◉ **Info**	The issues refer to the best SC coding practice and don't cause any problems with using SCs. Their handling depends on the decision of the dev team.

5. Disclaimer

Please note that this audit is not financial advice. Conduct your own research before investing.

Review a few audits from multiple different audit providers as one audit doesn't guarantee all issues are detected and that there will be no security issues with the project analyzed in the future.

Moreover, it's important to consider that some of the information given is time-sensitive: the project can update its smart contract system, implementing new smart contracts and documentation. Therefore, always track changes influencing investing terms.

6. Findings

6.1 The LevelXToken.sol Contract

Address	-
Contract overview	ERC20 Rebase, Reflection token.
Owner	-
Privileged functions	Owner Role: - updateBuyFee - updateSellFee - updateFeeLiquidityCut - updateLiquidityRecipient - updateMinimumFeeBalanceToSwap - updateMinimumRewardBalanceToSwap - addRewardToken - updateRewardBankroll - updateRewardPath - updateBurnAmountToBumpLevel - updateMinimumBalanceForRewards - updateBaselineTotalActiveSupply - updateNextRebaseRatePerEpoch - updateExpansionToRewardFactor - updateAutoClaim - updateExcludeFromTransferPenalty - updateExcludeFromTradeFee - updateExcludeFromRewardsDefaultBehavior - claimOnBehalfOf

Issues found

Sell/Buy fee

Severity	◉ Critical
SCW ID	209
Description	The contract owner can set the buy and the sell fees up to 100%. All trading activity via LVLX/BNB pair could be blocked in a centralized way.
Location	- updateBuyFee(uint256 _buyFee) - updateSellFee(uint256 _sellFee)
Recommendations	We recommend using limitations for the fee setters. From our perspective, it shouldn't be higher than 20%. But if this is the protocol requirement, it should be well documented. Also, we recommend using a timelock contract as an owner role for this type of setter.
Status	Acknowledged: "The owner of both the contract and its upgrade beacon (it has been deployed as an upgradable contract) are a multisig requiring 3 out of 4 signatures, and there are 2 people of each team (GrowthDeFi / Emp.money) in the signing roster. Here is the multisig address 0xdaa425fF33C00aD8AeaF689776CF8556Bad263b8 and the upgradable beacon address 0xC121B103690927a41C7EcF744Cc41bEa7f6853E3"

Overprivileged role

Severity	◉ High

SCW ID	198-d
Description	A privileged address can change critical protocol parameters. We noticed, that some setters don't include any input requirements. Thus, if an uncontrolled value is set, it could lead to unexpected behavior of the protocol.
Location	- updateMinimumFeeBalanceToSwap(uint256 _minimumFeeBalanceToSwap) - updateMinimumRewardBalanceToSwap(uint256 _minimumRewardBalanceToSwap) - updateBurnAmountToBumpLevel(uint256 _burnAmountToBumpLevel)
Recommendations	All critical protocol setters should include input requirements and be well documented. Also, we recommend using a timelock contract as an owner role for this type of setter.
Status	Acknowledged: "The owner of both the contract and its upgrade beacon (it has been deployed as an upgradable contract) are a multisig requiring 3 out of 4 signatures, and there are 2 people of each team (GrowthDeFi / Emp.money) in the signing roster. Here is the multisig address 0xdaa425fF33C00aD8AeaF689776CF8556Bad263b8 and the upgradable beacon address 0xC121B103690927a41C7EcF744Cc41bEa7f6853E3"

Costly operations in a loop

Severity	● Medium
SCW ID	179
Description	Costly loop operations can waste gas and lead to out-of-gas errors.
Location	- function updateMinimumBalanceForRewards(uint256

	_minimumBalanceForRewards, bool _forceUpdateAll)
Recommendations	Rewrite the function with a batch functionality.
Status	Acknowledged: “That functionality is there only for convenience and is not critical. We can use a script to perform the state update in batches, outside the contract, if necessary.”

Code duplication

Severity	◉ info
SCW ID	-
Description	The code could be optimized
Location	- _updateEpoch()
Recommendations	We have noticed code duplication in lines “597-605” and “623-631”. This functionality could be implemented in a separate function to avoid duplication.
Status	Acknowledged

Code styling

Severity	◉ info
SCW ID	-
Description	Lack of NatSpec comments and wrong order of functions.
Location	-
Recommendations	Follow the Solidity style guide recommendations. https://docs.soliditylang.org/en/v0.8.17/style-guide.html

Status	Acknowledged

6.2 The LevelXWrapperToken.sol Contract

Address	-
Contract overview	ERC20 standard token-based staking contract. Allows depositing in the native LevelX token and minting shares from the target contract.
Owner	-
Privileged functions	No privileged role.

Issues found

Code styling

Severity	◉ info
SCW ID	-
Description	Lack of NatSpec comments and wrong order of functions.
Location	-
Recommendations	Follow the Solidity style guide recommendations. https://docs.soliditylang.org/en/v0.8.17/style-guide.html
Status	Acknowledged

6.3 The LevelXSeed.sol Contract

Address	-
Contract overview	The token sale contract.
Owner	-
Privileged functions	No privileged role.

Issues found

Code styling

Severity	◉ info
SCW ID	-
Description	Lack of NatSpec comments and wrong order of functions.
Location	-
Recommendations	Follow the Solidity style guide recommendations. https://docs.soliditylang.org/en/v0.8.17/style-guide.html
Status	Acknowledged

6. Conclusion

We conducted a security audit on the LevelX protocol and found no security issues within our scope, apart from some centralization issues. We have shared some suggestions for improvement with the LevelX team, and they have already implemented the most critical ones by implementing the multi-sig wallet contract as an owner of the LevelX Token contract and UpgradeableBeacon. This has reduced the centralization risks, but we recommend users and the community monitor the activity of the multi-sig wallet and its signers.

In case the contract is upgradable, some medium and info severity issues could be addressed in the future. We also recommend making the smart contract documentation public on the GitHub repository, since the NatSpec comments are missing. To facilitate this, information about the smart contract functionality should be included in the readme.md file.