LeadWallet

Smart Contract Audit
Final Report - LeadStake V2

IMMUNE BYTES

Audits

LEAD

April 24, 2021

Introduction

1. About LeadWallet

The Lead Wallet team is committed to providing a sophisticated yet simple crypto wallet application that will enable anyone (either newbie or expert) to store, send, receive, spend, exchange/swap crypto assets at users' convenience without the need to provide or store user data. Lead Wallet will enable users across the globe at any time to conveniently spend their cryptocurrency assets in exchange for what they've always wanted to have or buy. In addition, Lead Wallet will constantly research and provide excellent blockchain technology and cryptocurrency application scenarios that will further the adoption and use cases of cryptocurrencies.

Visit https://leadwallet.io/ to know more

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

LeadWallet team has provided documentation for the purpose of conducting the audit. The documents are:

1. https://docs.google.com/document/d/1TBHrBFwoCG0suUtd7N-8UNXIGnXbtK1E9SY3eX0MU8M/edit?usp=sharing
2. Whitepaper
   https://leadwallet.io/en/docs/White%20Paper%20v1_0_2.pdf

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: LeadWallet
- Languages: Solidity(Smart contract)
- Github commit hash for audit: 8cd5075099a47c72ed569b12730f6264312342a8
- Github commit hash for final audit: 3439537ad45f0a61abf6a117a5f06ab3061678d7
- Deployed contract (Ropsten): 0xd4dd6c929855fd06beafa42e285ec7dfb1399370
- Final Audit Deployed contract (Ropsten): 0xcdb90bcb03bffda78ca3fa2400ec6cacaa09827d

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	Admin/Owner Privileges	High	Medium	Low
Open	5	-	-	1
Closed	-	1	-	7

Admin/Owner Privileges

The **admin/owner** of **LeadStakeV2** smart contract has various privileges over the smart contract. These privileges can be misused either intentionally or unintentionally (in case admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

1. **In the LeadStakeV2 contract the owner has the right to change some crucial contract parameters.**
   The integer params that can be changed by admin are **taxRate**, **basicROI**, **secondaryROI**, **tertiaryROI**, **masterROI**, **basicPeriod**, **secondaryPeriod** and **tertiaryPeriod**.
   Any positive integer value (including zero) can be set as these parameters anytime by the owner.

2. **In the LeadStakeV2 contract the owner has the right to change the *feeTaker* address parameter.**
   Any ethereum address (including zero address) can be set as the **feeTaker** address anytime by the owner.

3. **Admin has the right to change the *bonusToken* for LeadStakeV2 contract.**
   In the LeadStakeV2 contract, as soon as the claimable bonus becomes zero, the owner can change **bonusToken** for the contract.
   Any ethereum address (including zero address) can be set as the **bonusToken** address by the owner.

4. **Admin has the right to claim any excess bonusToken present in the LeadStakeV2 contract.**
   In the LeadStakeV2 contract, the owner has the right to transfer out an amount of bonusTokens which is greater than the total claimable amount by all stakers.

5. **Admin has the right to call the activateMigration() function.**
   In the LeadStakeV2 contract, the owner has the right to set any ethereum address (excluding zero address) as the **migrationContract** address variable. This function can be called any number of times by the owner.

Recommendation:
Consider hardcoding predefined ranges or validations for input variables in privileged access functions. Also consider adding some governance for admin rights for smart contracts or use a multi-sig wallet as admin/owner address.

High severity issues

1. **The LeadStakeV2 contract won't work with USDT.**
   The LeadStakeV2 contract contains multiple statements which deal with the transfer of USDT tokens (either by USDT's **transfer**() or **transferFrom**() function). However these statements will most likely get failed/reverted because the USDT token does not completely follow the ERC20 token standard. USDT does not return the **boolean** value for **transfer**() and **transferFrom**().

   Recommendation:
   Consider using OpenZeppelin's SafeERC20 library contract to perform USDT token transfers.

   Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

Low severity issues

1. ***migrationContract* can be changed multiple times by the owner.**
   In the LeadStakeV2 contract, the **activateMigration**() function is used to set the **migrationContract** by owner. However, as per the function's implementation it can also be used to update the **migrationContract** address variable.

   Recommendation:
   Consider adding a check in **activateMigration()** function which stops the updating of the **migrationContract** variable if it is already set.

   Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

2. **Return value of Standard ERC20 functions should be handled explicitly.**
   At Line 684 and 940 of the LeadStakeV2 contract, standard ERC20 function calls are performed but the **returned value** of the function call is ignored. It is always recommended to handle the return value of ERC20 functions explicitly.

   Recommendation:
   Consider using the OpenZeppelin's SafeERC20 library contract or wrap the mentioned statements in **require** statements.

3. **setFeeTaker() function should check the validity of input address variable.**
   In the LeadStakeV2 contract the **setFeeTaker()** function is used by the owner to set the fee receiver address. This function does not check the validity of input address and also allows the zero address (0x00) to be set as the **feeTaker** address. The **feeTaker** address is used in the **addLiquidity**() function to transfer the respective platform fee. If the **feeTaker** value is set as zero address, all transactions for adding liquidity will get reverted.

   Recommendation:
   Consider adding a check for zero address input in the **setFeeTaker()** function.

   Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

4. **Address of the Lead Stake V1 contract should be taken as a parameter in *constructor()*.**
   In the **constructor**() of LeadStakeV2 contract the address of Lead Stake V1 contract is hardcoded to the ropsten contract address. This increases the room for human error at the time of contract deployment. Ideally most contract parameters should be taken as input in the contract's constructor.

   Recommendation:
   Consider taking the address of the Lead Stake V1 contract in **constructor**() as a parameter.

   Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

5. **No getter function present for *feeTaker* variable.**
   In the LeadStakeV2 contract, the **feeTaker** address variable is declared as a **private** state variable. The private variable in Solidity cannot be accessed outside of the contract directly. Since there is no getter function implemented for that private state variable, it will be difficult for other smart contracts/offchain elements to read the value of that variable.

   Recommendation:
   Consider implementing a getter function to read the **feeTaker** value.

Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

6. **Events should be emitted when crucial contract parameters are changed.**
   In the LeadStakeV2 contract multiple functions are present which updates the crucial parameters of the contract. These functions include **setTaxRate**(), **setFeeTaker**(), **changeROI**(), etc. However these functions do not emit any events to log the state changes. It is always recommended to emit necessary events whenever some crucial variables are changed.

   Recommendation:
   Consider implementing more events in the smart contract.

   Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

7. **Unnecessary input variable in *migrate*() function.**
   The **migrate**() function in the LeadStakeV2 contract takes an address variable **_unistakeMigrationContract** as input. Since the **migrationContract** address is already set in the contract this input parameter can be safely omitted.

   Recommendation:
   Consider omitting the input address variable.

   Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

8. **Inconsistent use of *now* and *block.timestamp*.**
   In the LeadStakeV2 contract at Line 692 and 715 **now** keyword is used while at 788 and 928 **block.timestamp** is used. Both of these keywords return the same value but it is always recommended to use either one of them throughout the contract.

   Amended (Apr 24th 2021): Issue was fixed by Lead Wallet team and is no longer present in the github commit 3439537ad45f0a61abf6a117a5f06ab3061678d7.

Automated Auditing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contracts.

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to them in real-time. We performed analysis using contract Library on the Kovan address of the SporeToken, SporeStake and LiquidityFarming contracts used during manual testing:

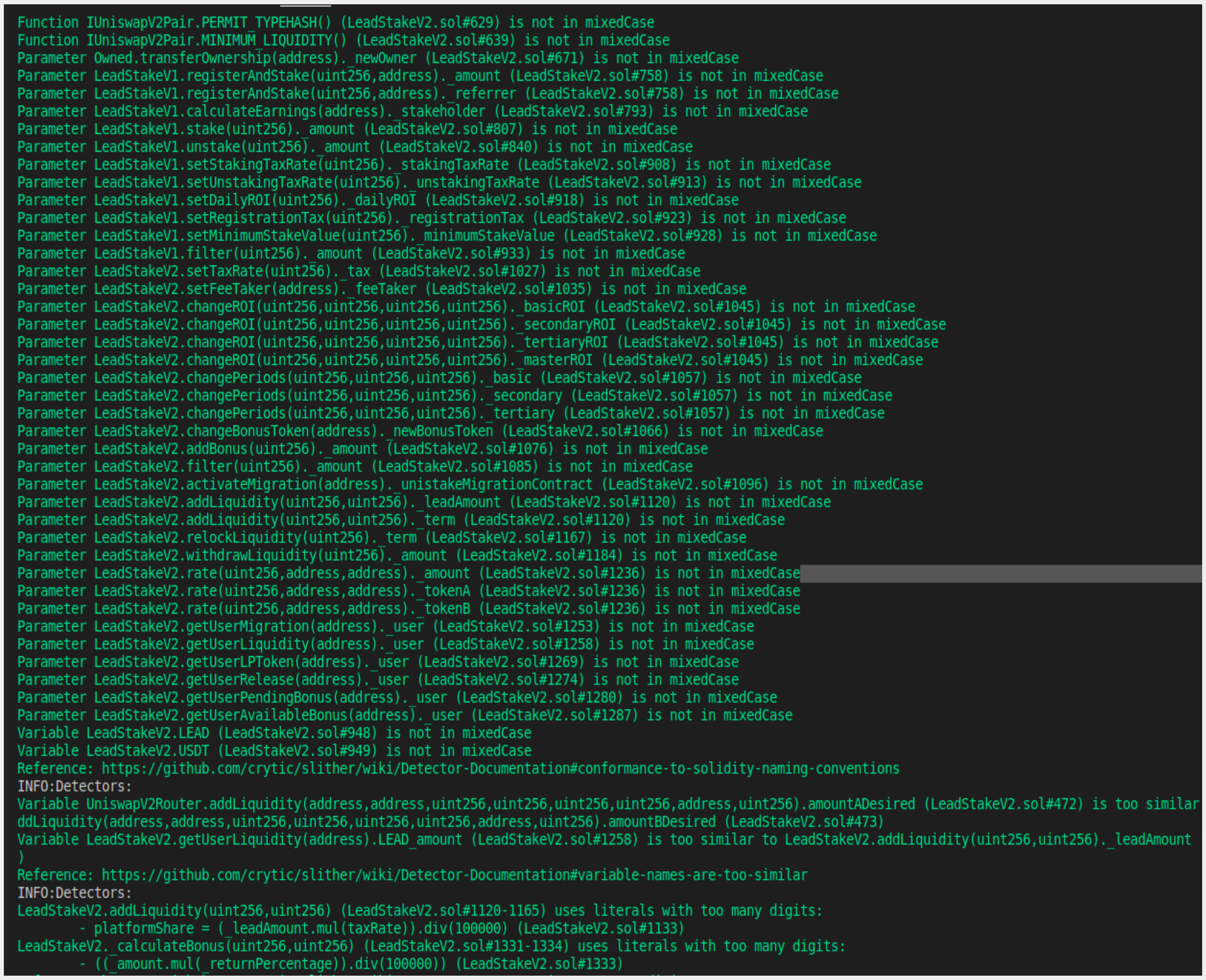
- LeadStakeV2: 0xd4dd6c929855fd06beafa42e285ec7dfb1399370
- LeadStakeV2 Final Audit: 0xcdb90bcb03bffda78ca3fa2400ec6cacaa09827d

It raises no major concern for the contracts.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit section.



```
Function IUniswapV2Pair.PERMIT_TYPEHASH() (LeadStakeV2.sol#629) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (LeadStakeV2.sol#639) is not in mixedCase
Parameter Owned.transferOwnership(address)._newOwner (LeadStakeV2.sol#671) is not in mixedCase
Parameter LeadStakeV1.registerAndStake(uint256,address)._amount (LeadStakeV2.sol#758) is not in mixedCase
Parameter LeadStakeV1.registerAndStake(uint256,address)._referrer (LeadStakeV2.sol#758) is not in mixedCase
Parameter LeadStakeV1.calculateEarnings(address)._stakeholder (LeadStakeV2.sol#793) is not in mixedCase
Parameter LeadStakeV1.stake(uint256)._amount (LeadStakeV2.sol#807) is not in mixedCase
Parameter LeadStakeV1.unstake(uint256)._amount (LeadStakeV2.sol#840) is not in mixedCase
Parameter LeadStakeV1.setStakingTaxRate(uint256)._stakingTaxRate (LeadStakeV2.sol#908) is not in mixedCase
Parameter LeadStakeV1.setUnstakingTaxRate(uint256)._unstakingTaxRate (LeadStakeV2.sol#913) is not in mixedCase
Parameter LeadStakeV1.setDailyROI(uint256)._dailyROI (LeadStakeV2.sol#918) is not in mixedCase
Parameter LeadStakeV1.setRegistrationTax(uint256)._registrationTax (LeadStakeV2.sol#923) is not in mixedCase
Parameter LeadStakeV1.setMinimumStakeValue(uint256)._minimumStakeValue (LeadStakeV2.sol#928) is not in mixedCase
Parameter LeadStakeV1.filter(uint256)._amount (LeadStakeV2.sol#933) is not in mixedCase
Parameter LeadStakeV2.setTaxRate(uint256)._tax (LeadStakeV2.sol#1027) is not in mixedCase
Parameter LeadStakeV2.setFeeTaker(address)._feeTaker (LeadStakeV2.sol#1035) is not in mixedCase
Parameter LeadStakeV2.changeROI(uint256,uint256,uint256,uint256)._basicROI (LeadStakeV2.sol#1045) is not in mixedCase
Parameter LeadStakeV2.changeROI(uint256,uint256,uint256,uint256)._secondaryROI (LeadStakeV2.sol#1045) is not in mixedCase
Parameter LeadStakeV2.changeROI(uint256,uint256,uint256,uint256)._tertiaryROI (LeadStakeV2.sol#1045) is not in mixedCase
Parameter LeadStakeV2.changeROI(uint256,uint256,uint256,uint256)._masterROI (LeadStakeV2.sol#1045) is not in mixedCase
Parameter LeadStakeV2.changePeriods(uint256,uint256,uint256)._basic (LeadStakeV2.sol#1057) is not in mixedCase
Parameter LeadStakeV2.changePeriods(uint256,uint256,uint256)._secondary (LeadStakeV2.sol#1057) is not in mixedCase
Parameter LeadStakeV2.changePeriods(uint256,uint256,uint256)._tertiary (LeadStakeV2.sol#1057) is not in mixedCase
Parameter LeadStakeV2.changeBonusToken(address)._newBonusToken (LeadStakeV2.sol#1066) is not in mixedCase
Parameter LeadStakeV2.addBonus(uint256)._amount (LeadStakeV2.sol#1076) is not in mixedCase
Parameter LeadStakeV2.filter(uint256)._amount (LeadStakeV2.sol#1085) is not in mixedCase
Parameter LeadStakeV2.activateMigration(address)._unistakeMigrationContract (LeadStakeV2.sol#1096) is not in mixedCase
Parameter LeadStakeV2.addLiquidity(uint256,uint256)._leadAmount (LeadStakeV2.sol#1120) is not in mixedCase
Parameter LeadStakeV2.addLiquidity(uint256,uint256)._term (LeadStakeV2.sol#1120) is not in mixedCase
Parameter LeadStakeV2.relockLiquidity(uint256)._term (LeadStakeV2.sol#1167) is not in mixedCase
Parameter LeadStakeV2.withdrawLiquidity(uint256)._amount (LeadStakeV2.sol#1184) is not in mixedCase
Parameter LeadStakeV2.rate(uint256,address,address)._amount (LeadStakeV2.sol#1236) is not in mixedCase
Parameter LeadStakeV2.rate(uint256,address,address)._tokenA (LeadStakeV2.sol#1236) is not in mixedCase
Parameter LeadStakeV2.rate(uint256,address,address)._tokenB (LeadStakeV2.sol#1236) is not in mixedCase
Parameter LeadStakeV2.getUserMigration(address)._user (LeadStakeV2.sol#1253) is not in mixedCase
Parameter LeadStakeV2.getUserLiquidity(address)._user (LeadStakeV2.sol#1258) is not in mixedCase
Parameter LeadStakeV2.getUserLPToken(address)._user (LeadStakeV2.sol#1269) is not in mixedCase
Parameter LeadStakeV2.getUserRelease(address)._user (LeadStakeV2.sol#1274) is not in mixedCase
Parameter LeadStakeV2.getUserPendingBonus(address)._user (LeadStakeV2.sol#1280) is not in mixedCase
Parameter LeadStakeV2.getUserAvailableBonus(address)._user (LeadStakeV2.sol#1287) is not in mixedCase
Variable LeadStakeV2.LEAD (LeadStakeV2.sol#948) is not in mixedCase
Variable LeadStakeV2.USDT (LeadStakeV2.sol#949) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable UniswapV2Router.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (LeadStakeV2.sol#472) is too similar
ddLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (LeadStakeV2.sol#473)
Variable LeadStakeV2.getUserLiquidity(address).LEAD_amount (LeadStakeV2.sol#1258) is too similar to LeadStakeV2.addLiquidity(uint256,uint256)._leadAmount
)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
LeadStakeV2.addLiquidity(uint256,uint256) (LeadStakeV2.sol#1120-1165) uses literals with too many digits:
        - platformShare = (_leadAmount.mul(taxRate)).div(100000) (LeadStakeV2.sol#1133)
LeadStakeV2._calculateBonus(uint256,uint256) (LeadStakeV2.sol#1331-1334) uses literals with too many digits:
        - ((_amount.mul(_returnPercentage)).div(100000)) (LeadStakeV2.sol#1333)
```

Concluding Remarks

While conducting the audits of LeadWallet smart contract, it was observed that the contracts contain Admin/Owner Privileges, High and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High and Low severity issues should be resolved by LeadWallet developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the LeadWallet platform or its product neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.