CERTIK

StarLink Miner Protocol

Security Assessment

January 18th, 2021

For :
StarLink

By :

Owan Li @ CertiK
guilong.li@certik.org

Bryan Xu @ CertiK
buyun.xu@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	StarLink Miner
Description	A defi miner protocol with SLN token
Platform	Ethereum; Solidity; Yul
Codebase	GitHub Repository
Commit	42e49305dad56c373b7b3602ed7c4a858b0022ea ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a 3c30da5a5e5062d375262c2b3947cf9a354b571c

Audit Summary

Delivery Date	Jan. 18th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 5, 2021 - Jan. 7, 2021

Vulnerability Summary

Total Issues	12
Total Critical	0
Total Major	0
Total Minor	3
Total Informational	9

Executive Summary

This report has been prepared for **Starlinkminer** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, 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.

File in Scope

Contract	SHA-256 Checksum
SLNToken.sol	2d95413bfd4bc665d42d6e05b39025efa0a3755baad8202b1fe61a699ad713f0
StarPools.sol	14ad734541e0fc9fca36ee39a2cb03cb29d51fea5024f9cf9a3b1c5294951f63
WordFund.sol	365b0396da906f39c160fe21dbc9cfa7e33c655b75c19e8893ac00e0d021267b

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract’s functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **StarLink** team or reported an issue.

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 3 minor vulnerabilities were identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate doumentation and specification outside of the source files, however the code comment coverage was minimal.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings

ID	Title	Type	Severity	Resolved
Exhibit-01	Unlocked Compiler Version Declaration	Language Sepcific	Informational	✓
Exhibit-02	Incorrect Naming Convention Utilization	Coding Style	Informational	❗
Exhibit-03	Incorrect Order of Layout Utilization	Coding Style	Informational	✓
Exhibit-04	Proper Usage of “public” and “external” type	Gas Optimization	Informational	✓
Exhibit-05	Lack of natspec comments	Optimization	Informational	❗
Exhibit-06	Use SafeMath	Mathematical Operations	Informational	❗
Exhibit-07	State variables that could be declared constant	Gas Optimization	Informational	✓
Exhibit-08	Missing Emit Events	Optimization	Minor	❗
Exhibit-09	Events Should Add Indexed Keyword	Optimization	Informational	❗
Exhibit-10	Missing Checks	Logical Issue	Minor	✓
Exhibit-11	Code Redundancy	Optimization	Informational	✓
Exhibit-12	Potential Overflow	Mathematical Operations	Minor	✓

Exhibit-01: Unlocked Compiler Version Declaration

Type	Severity	Location
Language Sepcific	Informational	SLNToken.sol, StarPools.sol, WordFund.sol, DebugLogs.sol

Description:

The compiler version utilized throughout the project uses the “^” prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts. Recommend the compiler version should be consistent throughout the codebase.

Recommendation:

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team heeded our advice and locked the version of their contracts at version 0.6.12, ensuring that compiler-related bugs can easily be narrowed down should they occur.
The recommendations were applied in commit ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a.

Exhibit-02: Incorrect Naming Convention Utilization

Type	Severity	Location
Coding Style	Informational	SLNToken.sol, StarPools.sol, WordFund.sol

Description:

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to: https://solidity.readthedocs.io/en/v0.6.0/style-guide.html#naming-conventions

Events should be named using the CapWords style.
Examples:
Events like: `addWord` , `biddingWord` , `harvestWord` , `releaseWord` , `setWordData` , `claimInvoked` , `factorReset` , `depositInvoked` , `withdrawInvoked` , `depositMoveInvoked` , `joinInvoked` , `quitInvoked` , `claimInvoked` , `claimMoveInvoked` , `claimValue`

Parameter shoud use mixedCase.
Examples:
Parameter like: `_poolid` , `_topoolid` , `_toaccount` , `_wordid` , `_wordlist`

Structs should be named using the CapWords style.
Examples:
Struct like: `worddata`

Recommendation:

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.

Alleviation:

No alleviation.

Exhibit-03: Incorrect Order of Layout Utilization

Type	Severity	Location
Coding Style	Informational	SLNToken.sol, StarPools.sol, WordFund.sol

Description:

Solidity defines an Order of Layout that should be followed. In general, inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Functions

refer to: https://docs.soliditylang.org/en/v0.6.0/style-guide.html?highlight=layout#order-of-layout

Recommendation:

See Exhibit 2.

Alleviation:

The team heeded our advice and fixed the order of layout.
The recommendations were applied in commit ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a.

Exhibit-04: Proper Usage of "public" and "external" type

Type	Severity	Location
Gas Optimization	Informational	SLNToken.sol, StarPools.sol, WordFund.sol

Description:

"public" functions that are never called by the contract could be declared "external" . When the inputs are arrays "external" functions are more efficient than "public" functions.
Examples:
Functions `setTeamAddress()` , `setPoolAddress()` , `tokensThisWeek()` , `claim()` , `setBlockedlist()` in contract `SLNToken` .
Functions `poolLength()` , `addPool()` , `setPool()` , `setLiveFactor()` , `deposit()` , `withdraw()` , `depositMove()` , `claim()` , `claimMove()` , `annualPerShare()` in contract `StarPools` .
Functions `setRelatedPoolId()` , `wordsLength()` , `addWords()` , `setData()` , `bidding()` , `harvest()` , `release()` , `claim()` in contract `WordFund` .

Recommendation:

Consider using the "external" attribute for functions never called from the contract.

Alleviation:

The team heeded our advice and used the "external" attribute for functions never called from the contract.
The recommendations were applied in commit ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a.

Exhibit-05: Lack of natspec comments

Type	Severity	Location
Optimization	Informational	SLNToken.sol, StarPools.sol, WordFund.sol

Description:

Contract code is missing natspec comments, which helps understand the code and all the functions' parameters.

Recommendation:

Please follow these style guides for adding natspec comments.
https://docs.soliditylang.org/en/v0.6.11/style-guide.html?highlight=natspec%23natspec

Alleviation:

No alleviation.

Exhibit-06: Use SafeMath

Type	Severity	Location
Mathematical Operations	Informational	SLNToken.sol, StarPools.sol, WordFund.sol

Description:

Many functions in the `grant` contract did not use SafeMath.

Example:
Functions `_makeSum()` , `_makeLastWeek()` , `tokensThisWeek()` , `_calcMintValue()` , `calcPoolValue()` , `teamClaim()` in contract `SLNToken` .
Functions `addPool()` , `_profitnow()` , `_quit()` , `_join()` , `balanceOf()` , `annualPerShare()` in contract `StarPools` .
Functions `lowAmount()` , `bidding()` , `harvest()` , `_ishold()` , `release()` in contract `WordFund` .

Recommendation:

We recommend to use SafeMath for calculations.

Alleviation:

No alleviation.

Exhibit-07: State variables that could be declared constant

Type	Severity	Location
Gas Optimization	Informational	WordFund.sol L43

Description:

Constant state variables should be declared constant to save gas.

```
    uint256 public biddingLockingPeriod = (4 hours);
    uint256 public positionLockingPeriod = (15 days);
```

Recommendation:

Consider to add the constant attributes to state variables that never change.

```
        uint256 public constant biddingLockingPeriod = (4 hours);
        uint256 public constant positionLockingPeriod = (15 days);
```

Alleviation:

The team heeded our advice and defined the un-changed variables as constants.
The recommendations were applied in commit ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a.

Exhibit-08: Missing Emit Events

Type	Severity	Location
Optimization	Minor	SLNToken.sol L44,L48, StarPools.sol L42,L51,L60, WordFund.sol L21, L25

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like : `setPoolAddress()` , `setRelatedPoolId` , `setTeamAddress()` , `addPool()` , `setPool()` , `setLiveFactor()`

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
        event SetPoolAddress(address indexed poolAddress);
        function setPoolAddress(address _poolAddress) public onlyOwner {
            poolAddress = _poolAddress;
            _approve(address(this), poolAddress, totalSupply()*835/1000);
            emit SetPoolAddress(poolAddress);
        }
```

Recommendation:

Change the condition to check inequality with zero, as it is more efficient regarding unsigned integer variables.

Alleviation:

No alleviation.

Exhibit-09: Events Should Add Indexed Keyword

Type	Severity	Location
Optimization	Informational	SLNToken.sol L37, StarPools.sol L82,L164,L296 WordFund.sol L53

Description:

Event definittions in contract `SLNToken` , `StarPools` and `WordFund` do not have `indexed` keyword.
The indexed parameters for logged events will allow you to search for these events using the indexed parameters as filters.

Examples:

```
    event claimInvoked(address from, uint256 poolid, uint256 wordid, address to);
```

Recommendation:

We recommend to add the `indexed` keyword.

```
    event claimInvoked(address indexed from, uint256 indexed poolid, uint256 indexed wordid,
address indexed to);
```

Alleviation:

No alleviation.

Exhibit-10: Missing Checks

Type	Severity	Location
Logical Issue	Minor	SLNToken.sol L17,L44,L129 StarPools.sol L14,L42,L51 WordFund.sol L21,L108

Description:

Function `setTeamAddress` , `setPoolAddress` , `constructor` in contract SLNToken, `constructor` in contract StarPools, `setPoolAddress` in contract WordFund are missing zero address checks.
Function `setBlockedlist()` in contract SLNToken, `setPool()` in contract StarPools, `bidding()` in contract WordFund in are missing parameter value checks.

Function `addPool()` in contract StarPools is missing parameter checks for `_LPToken` , same `_LPToken` can be used multiple times to add multiple pools.

Recommendation:

We recommend to add neccessary checks, for example:

```
        function setTeamAddress(address _teamAddress) public onlyOwner {
            require(_teamAddress != address(0), "zero address is not allowed!");
            teamAddress = _teamAddress;
        }
        function setBlockedlist(address _address, bool _blocked) public onlyOwner {
            require(blockedlist[_address] != _blocked);
            blockedlist[_address] = _blocked;
        }
        function setPool(uint256 _poolid, address _LPLimit, uint256 _start, uint256 _ending,
                    bool _paused) public onlyOwner returns (uint256) {
            require(_poolid < pools.length , "pool id is not existing!");
            pools[_poolid].LPLimit = _LPLimit;
            ...
        }
        function bidding(uint256 _wordid, uint256 _poolid, uint256 _value, address _referrer)
    public returns (uint256) {
        ...
            address LPAddress = getPoolLPToken(poolId);
            require(LPAddress != address(0) , "pool id is not existing!");
        ...
        }


        ...
```

Alleviation:

The team heeded our advice and added the checks.
The recommendations were applied in commit ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a.

Exhibit-11: Code Redundancy

Type	Severity	Location
Optimization	Informational	StarPools.sol L298

Description:

Return value for function `_setInviter()` is never used.

Recommendation:

Consider to remove un-used return value to reduce code redundancy.

Alleviation:

The team heeded our advice and removed the un-used return value.
The recommendations were applied in commit ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a.

Exhibit-12: Potential Overflow

Type	Severity	Location
Mathematical Operations	Minor	WordFund.sol L62

Description:

Variable `weight` in function `lowAmount()` is not checked for zero value.
This can lead to an overflow, due to the nature of the divide operation.

Recommendation:

Consider to add a check for the variable `weight` before devide operation.

```
    function lowAmount() public view returns (uint256) {
        uint256 weight;
        (,,,,weight,,,) = StarPools(poolAddress).pools(poolId);
        require (weight != 0, "weight is zero")
        return 100*(10**18)*(10**9)/weight;
    }
```

Alleviation:

The team heeded our advice and added the check.
The recommendations were applied in commit ac56f1dfb4fe37c7af01cbd50fd6b0cf9e0edf2a.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

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

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 instorage 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 and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

✓ : Issue resolved

❗ : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

❗✓ : Issue partially resolved. Not all instances of an issue was resolved.