SPADE SOLIDITY AUDITS

Toad Network

07th June, 2021

For :

Toad Network

Twitter:
@SpadeAudits

Telegram:
t.me/spadeaudits

Disclaimer

Spade Solidity 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 Spade to perform a security review.

Spade Solidity 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.

Spade Solidity 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.

Spade Solidity 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. Spade Solidity’s position is that each company and individual are responsible for their own due diligence and continuous security. Spade Solidity’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 Spade Solidity report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to Spade Solidity 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 Spade Solidity 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	Toad Network
Description	Farming Protocol
Platform	Binance Smart Chain
Codebase	Received file
Commit	NA

Audit Summary

Delivery Date	07th June, 2021
Method of Audit	Static Analysis, Manual Review
Timeline	Story Points - 40

Vulnerability Summary

Total Issues	4
Total Critical	0
Total High	0
Total Medium	0
Total Low	2
Total Informational	2

Executive Summary

TOAD network is a decentralised farming protocol on the Binance smart chain. Four farming pools are available providing rewards in either TOAD or TOAD/BNB LP tokens. TOAD will also act as governance token for the TOAD/ PADswap ecosystem.

The protocol is designed to rug pull proof. TOAD's team donated all of their LP tokens to the LP farm for redistribution via farming rewards.

The protocol is designed to be resistant to flash loan attacks with no mint/burn functions derived from straight action.

The TOAD team have employed additional security techniques including master keys, hardware verification, vpns and virtual machines. A voted upon member of the community will act as a a 3rd multisig for the team.

Step 1
A manual line-by-line code review to ensure the logic behind each function is sound and safe from common attack vectors.
Step 2
Simulation of hundreds of thousands of Smart Contract Interactions on a test blockchain using a combination of automated test tools and manual testing to determine if any security vulnerabilities exist.
Step 3
Consultation with the project team on the audit report pre-publication to implement recommendations and resolve any outstanding issues.

Grading

The following grading structure was used to assess the level of vulnerability found within TOAD network Smart Contracts:

Threat Level	Definition
Critical	Severe vulnerabilities which compromise the entire protocol and could result in immediate data manipulation or asset loss.
High	Significant vulnerabilities which compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Medium	Vulnerabilities which if not fixed within in a set timescale could compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Low	Low level vulnerabilities which may or may not have an impact on the optimal performance of the Smart contract.
Informational	Issues related to coding best practice which do not have any impact on the functionality of the Smart Contracts.

TOAD AUDIT REPORT

ID	CONTRACT	SHA-256 CHECKSUM
TFR	TOADFARM.sol	392931f2d3d782aed300277ddb4894ba83ee0f463bd5def5415c96cb3837c087
TLF	TOADLPFARM.sol	309e7843c4f18c4564bfbcd0ea67c6700d8dc35d707c3aae74d46502affa05bc

ID	TITLE	TYPE	SERVERITY
TFR-01	Volatile code	define variable first	Informational
TFR-02	Volatile code	type cast	Low
TLF-01	Volatile code	define variable first	Informational
TLF-02	Volatile code	type cast	Low

TFR-01: define variable first

TYPE	SERVERITY	LOCATION
Volatile Code	Informational	TOADFARM.sol L62

Description:

Define variable first then use logic on modifier

```
modifier hasDripped {

if (dividendPool > 0) {
uint256 secondsPassed = SafeMath.sub(now, lastDripTime);
uint256 dividends = secondsPassed.mul(dividendPool).div(dailyRate);
if (dividends > dividendPool) {
dividends = dividendPool;
}

profitPerShare = SafeMath.add(profitPerShare, (dividends * divMagnitude)
 / tokenSupply);

dividendPool = dividendPool.sub(dividends);

lastDripTime = now;

}

_;

}
profitPerShare = SafeMath.add(profitPerShare, (dividends * divMagnitude)
 / tokenSupply);
```

Use safemath functionsto avaid math error
TOADFARM.sol L62
Suggestion: define variable first then write if clause

Resolution status:

Issue acknowledged by Toad Network Team.

TFR-02: type cast

TYPE	SERVERITY	LOCATION
Volatile code	Low	TOADFARM.sol L291, L300, L175.L183

Description:

Type casting on same line may cause error

```
(uint256) ((int256) (_profitPerShare * tokenBalanceLedger[_customer
Address]) - payoutsTo[_customerAddress]) / divMagnitude;
```

TOADFARM.sol L291, L300

Suggestion:

first type cast from int256 to uint256 then return uint256

Resolution status:

Issue acknowledged by Toad Network Team.

TLF-01: define variable first

TYPE	SERVERITY	LOCATION
Volatile code	Informational	TOADLPFARM.sol L61

Description:

Define variable first then use logic on modifier

```
modifier hasDripped {
if (dividendPool > 0) {
uint256 secondsPassed = SafeMath.sub(now, lastDripTime);
uint256 dividends = secondsPassed.mul(dividendPool).div(dailyRate);
if (dividends > dividendPool) {
dividends = dividendPool;
}
profitPerShare = SafeMath.add(profitPerShare, (dividends * divMagnitude)
/ tokenSupply);
dividendPool = dividendPool.sub(dividends);
lastDripTime = now;
}
_;
}
profitPerShare = SafeMath.add(profitPerShare, (dividends * divMagnitude)
/ tokenSupply);
```

Use safemath functions to avoid math error
TOADLPFARM.sol L61

Suggestion:

define variable first then write if clause

Resolution status:

Issue acknowledged by Toad Network Team.

TLF-02: type cast

TYPE	SERVERITY	LOCATION
Volatile Code	Low	TOADLPFARM.sol L317, L326

Description:

Type casting on same line may cause error

```
(uint256) ((int256) (_profitPerShare * tokenBalanceLedger[_customer
Address]) - payoutsTo[_customerAddress]) / divMagnitude;
```

TOADLPFARM.sol L317, L326

Suggestion:

first type cast from int256 to uint256 then return uint256

Note : type casting on view functions not cause any serious issues, just cause warnings.

Resolution status:

Issue acknowledged by Toad Network Team.

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 unexpectely on certain edge cases that may result in avulnerability.

Data Flow
Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a structassignment 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.