SPADE SOLIDITY AUDITS

Padswap

May 15, 2021

For :
Padswap

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	Padswap
Description	DeFi
Platform	Binance Smart Chain
Codebase	Received file
Commit	NA

Audit Summary

Delivery Date	May 15, 2021
Method of Audit	Static Analysis, Manual Review
Timeline	Story Points - 56

Vulnerability Summary

Total Issues	5
Total Critical	0
Total High	0
Total Medium	0
Total Low	1
Total Informational	4

Executive Summary

This report has been prepared for Padswap to assess the security and performance of the project's Smart Contracts. Padswap is a decentralised exchange protocol on the Binance Smart Chain. It forms part of the Toad/Pad ecosystem.

Our detailed audit methodology was as follows:

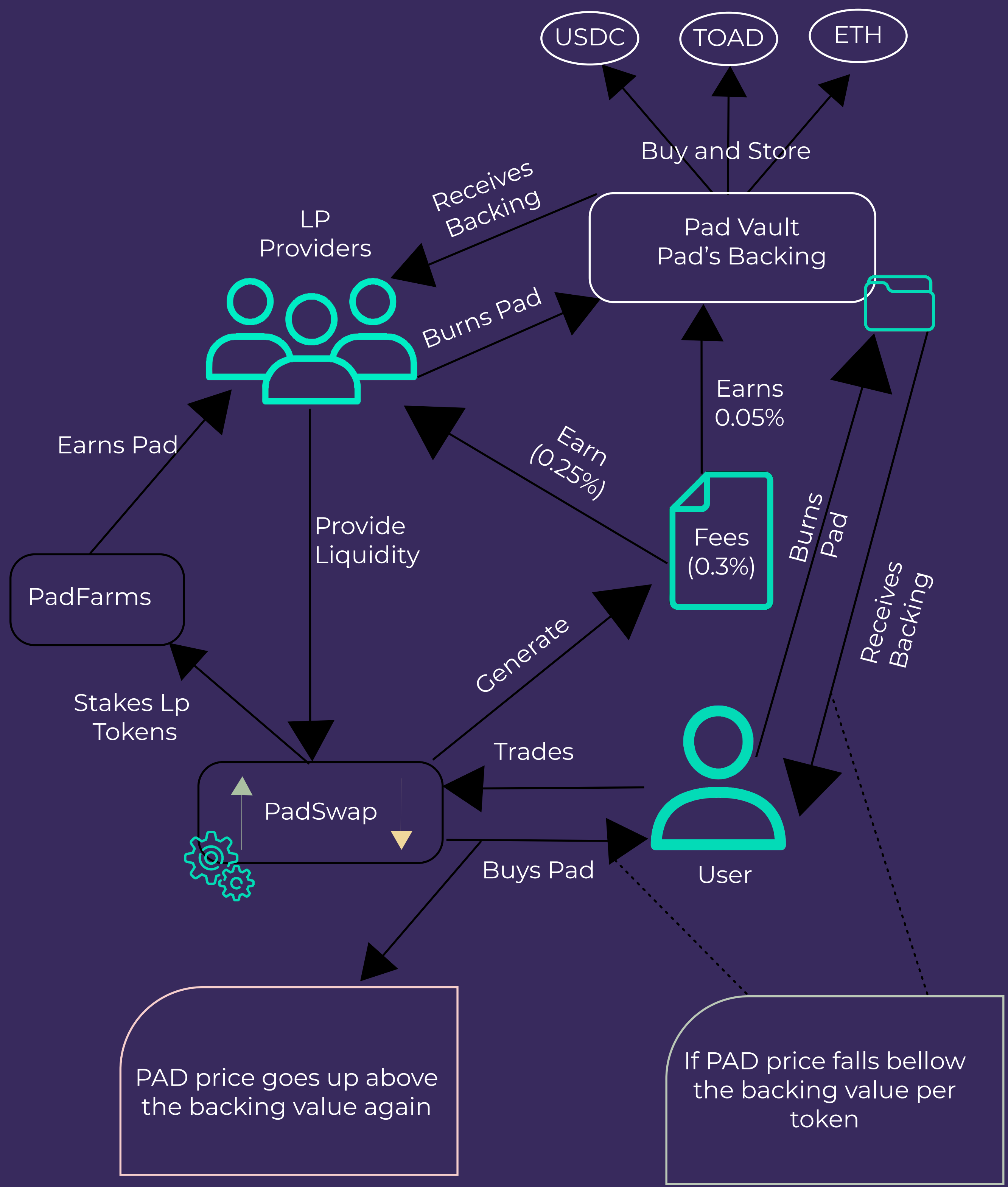
Step 1
A manual line-by-line code review to ensure the logic behind each function is sound and safe from common attack vectors.
Step 2
Simulation of hundreds of thousands of Smart Contract Interactions on a test blockchain using a combination of automated test tools and manual testing to determine if 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 Padswap 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.

Padswap Project Flowchart:



PAD AUDIT REPORT

ID	CONTRACT	SHA-256 CHECKSUM
PFR	FARMS.sol	c9c4c29aed49c19c3ea940d6512eaf2f6e07a91926ffde562d58bfe82ffcfb09
PMR	MINTER.sol	ab7ededcd339aa7736f5729e65b31dc398423f91b1c9be1f01be3130a73479e8
PPD	PAD.sol	43c53b0e8296d5bf4ff2a42904371244203cbd36479164d94d472aacc75c4032
PPF	PADFLATTEN.sol	cf86bea84fe919ceb7fff71c1aca7b7913914193ea36d52b88384392a2320aa6
PTT	TESTNETTOKENS.sol	e9d991b2098a6837c07ae20d9c8ae0185b576ed0fba0025f152bfa2b34064686
PVT	VAULT.sol	b75824106b70e1cbc332c4664712dc8fe16e9a43e99cf89bf5b2fb15407187b4

ID	TITLE	TYPE	SERVERITY
PVT-01	Volatile code	makes safe math calculation	Low
PFR-01	Volatile code	type cast	Informational
PFR-02	Volatile code	define variable first	Informational
PMR-01	Volatile code	type cast	Informational
PMR-02	Volatile code	define variable first	Informational

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

PVT-01: makes safe math calculation

TYPE	SERVERITY	LOCATION
Volatile Code	Low	Vault.sol L101

Description:

Multiple times of mul, div performs at same line may cause error,

```
uint256 backingAmount =
tokenBalance.mul(magnitude).div(_padSupply).mul(_burnAmount).mul
(leverage).div(magnitude);
```

Vault.sol L101

Suggestion:

Make mul, div function first, and then next line mul functions twice,
It will be better if div functions written at last line.

Resolution status:

Issue resolved by Padswap Team by updating code.

PMR-01: type cast

TYPE	SERVERITY	LOCATION
Volatile code	Informational	Minter.sol L178

Description:
Type casting on same line may cause error

```
(uint256) ((int256) (_profitPerShare * sharesBalanceLedger
[_contractAddress]) - mintedBy[_contractAddress]) / mintMagnitude;
```

Minter.sol L178

Suggestion:
first type cast from int256 to uint256 then return uint256

Resolution status:

Issue acknowledged by Padswap Team.

PMR-02: define variable first

TYPE	SERVERITY	LOCATION
Volatile code	Informational	Minter.sol L52

Description:
Define variable first then use logic on modifier

```
modifier hasUpdated {
    if (remainingSupply > 0 && sharesSupply > 0) {
      uint256 secondsPassed = SafeMath.sub(now, lastMintTime);
      uint256 mintAmount = secondsPassed.mul(remainingSupply).div
      (dailyRate);

      if (mintAmount > remainingSupply) {
        mintAmount = remainingSupply;
      }

      profitPerShare = SafeMath.add(profitPerShare,
      (mintAmount * mintMagnitude) / sharesSupply);
      remainingSupply = remainingSupply.sub(mintAmount);
      lastMintTime = now;
    }
    _;
  }
profitPerShare = SafeMath.add(profitPerShare,
(dividends * divMagnitude) / tokenSupply);
```

Use safemath functions to avoid math error

Minter.sol L52

Suggestion:
define variable first then write if clause

Resolution status:

Issue acknowledged by Padswap Team.

PFR-01: type cast

TYPE	SERVERITY	LOCATION
Volatile Code	Informational	Farms.sol L219

Description:

Type casting on same line may cause error

```
(uint256) ((int256) (_profitPerShare * sharesBalanceLedger[_farmerAddress])
- payoutsTo[_farmerAddress]) / magnitude;
```

Farms.sol L219

Suggestion:
Frst type cast from int256 to uint256 then return uint256

Resolution status:

Issue acknowledged by Padswap Team.

PFR-02: define variable first

TYPE	SERVERITY	LOCATION
Volatile code	Informational	Farms.sol L43

Description:

Define variable first then use logic on modifier

```
modifier hasDripped {
      if (farmPool > 0 && sharesSupply > 0) {
        uint256 secondsPassed = SafeMath.sub(block.timestamp, lastDripTime);
        uint256 rewards = secondsPassed.mul(farmPool).div(dailyRate);

        if (rewards > farmPool) {
          rewards = farmPool;
        }

        profitPerShare = SafeMath.add(profitPerShare, (rewards * magnitude) /
sharesSupply);
        farmPool = farmPool.sub(rewards);
        lastDripTime = block.timestamp;
      }
      _;
    }
profitPerShare = SafeMath.add(profitPerShare, (dividends * divMagnitude) /
tokenSupply);
```

Use safemath functions to avoid math error

Farms.sol L43

Suggestion: define variable first then write if clause

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

Resolution status:

Issue acknowledged by Padswap 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.