SpookySwap Integration & Fantom Expansion

Smart Contract Audit Report
Prepared for Alpaca Finance

Date Issued: Feb 8, 2022
Project ID: AUDIT2022008
Version: v1.0
Confidentiality Level: Public

inspex
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022008
Version	v1.0
Client	Alpaca Finance
Project	SpookySwap Integration & Fantom Expansion
Auditor(s)	Weerawat Pawanawiwat Puttimet Thammasaeng
Author(s)	Wachirawit Kanpanluk
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Feb 8, 2022	Full report	Wachirawit Kanpanluk

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Alpaca Finance, Inspex team conducted an audit to verify the security posture of the SpookySwap Integration & Fantom Expansion smart contracts between Feb 1, 2022 and Feb 3, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of SpookySwap Integration & Fantom Expansion smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 2 high, 1 medium, 2 low and 2 very low-severity issues. With the project team's prompt response, 2 high, 1 medium, 1 low, and 1 very low-severity issues were resolved or mitigated in the reassessment, while 1 low and 1 very low-severity issues were acknowledged by the team. Therefore, Inspex trusts that SpookySwap Integration & Fantom Expansion smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Alpaca Finance is the largest lending protocol allowing leveraged yield farming on Binance Smart Chain. It helps lenders to earn safe and stable yields, and offers borrowers undercollateralized loans for leveraged yield farming positions, vastly multiplying their farming principals and resulting profits.

SpookySwap Integration & Fantom Expansion is a new feature for Alpaca Finance, expanding the protocol to the Fantom chain. This allows the users to open leveraged yield farming positions with Alpaca Finance to farm on SpookySwap for a higher amount of rewards.

Scope Information:

Project Name	SpookySwap Integration & Fantom Expansion
Website	https://www.alpacafinance.org/
Smart Contract Type	Ethereum Smart Contract
Chain	Fantom
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Feb 1, 2022 - Feb 3, 2022
Reassessment Date	Feb 8, 2022

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 5f1ea3c0e8b65bee715f3c192340416e829b7f2c)

Contract	Location (URL)
MiniFL	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/8.11/MiniFL/MiniFL.sol
Rewarder1	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/8.11/MiniFL/rewarders/Rewarder1.sol
SpookyWorker03	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/6/protocol/workers/spookyswap/SpookyWorker03.sol
SpookySwapStrategyAddBaseTokenOnly	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyAddBaseTokenOnly.sol
SpookySwapStrategyAddTwoSidesOptimal	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyAddTwoSidesOptimal.sol
SpookySwapStrategyLiquidate	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyLiquidate.sol
SpookySwapStrategyPartialCloseLiquidate	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyPartialCloseLiquidate.sol
SpookySwapStrategyPartialCloseMinimizeTrading	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyPartialCloseMinimizeTrading.sol
SpookySwapStrategyWithdrawMinimizeTrading	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/5f1ea3c0e8/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyWithdrawMinimizeTrading.sol

Reassessment: (Commit: 4553a34a6dcfcfbf7aebc693bb5c5c6074c73129)

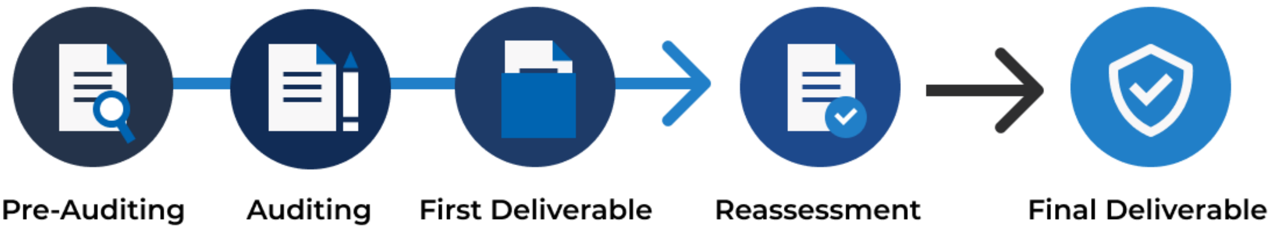
Contract	Location (URL)
MiniFL	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/8.11/MiniFL/MiniFL.sol
Rewarder1	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/8.11/MiniFL/rewarders/Rewarder1.sol
SpookyWorker03	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/6/protocol/workers/spookyswap/SpookyWorker03.sol
SpookySwapStrategyAddBaseTokenOnly	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyAddBaseTokenOnly.sol
SpookySwapStrategyAddTwoSidesOptimal	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyAddTwoSidesOptimal.sol
SpookySwapStrategyLiquidate	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyLiquidate.sol
SpookySwapStrategyPartialCloseLiquidate	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyPartialCloseLiquidate.sol
SpookySwapStrategyPartialCloseMinimizeTrading	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyPartialCloseMinimizeTrading.sol
SpookySwapStrategyWithdrawMinimizeTrading	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/4553a34a6d/contracts/6/protocol/strategies/spookyswap/SpookySwapStrategyWithdrawMinimizeTrading.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Centralized Control of State Variable
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication

Improper Kill-Switch Mechanism
Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

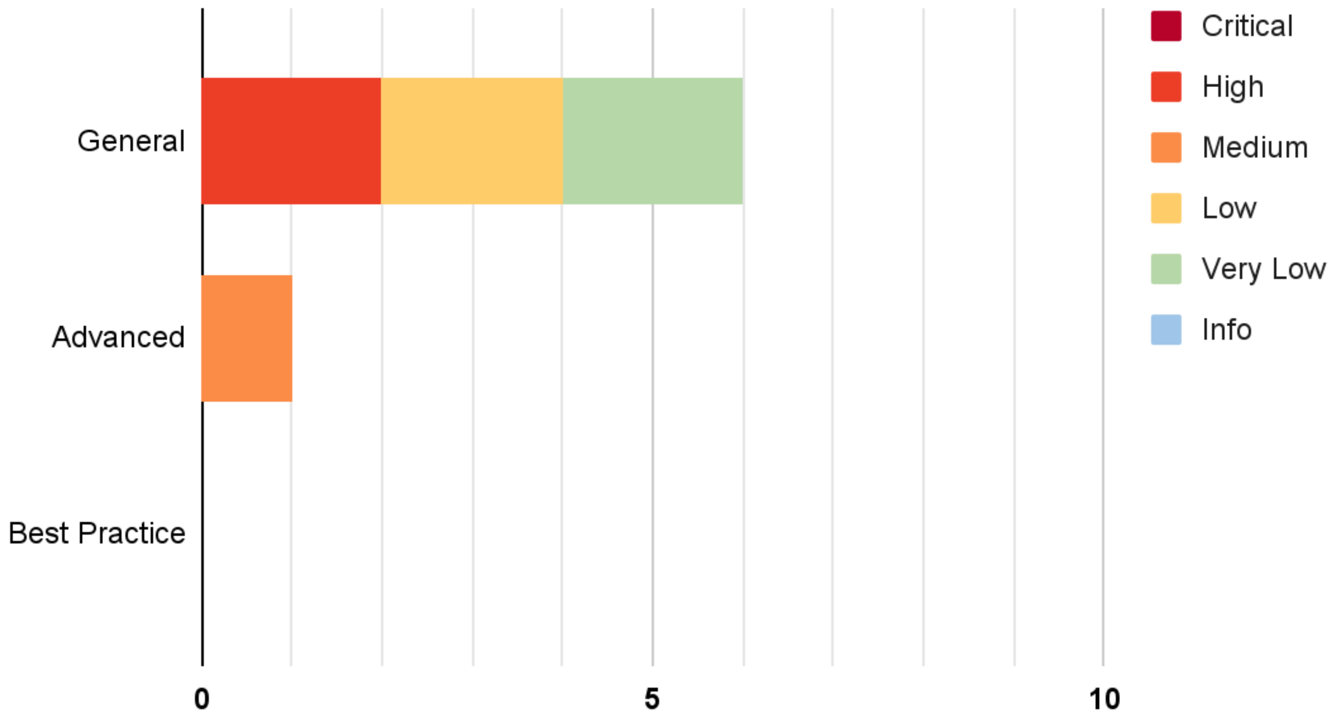
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Likelihood Impact	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

4. Summary of Findings

From the assessments, Inspex has found 7 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Centralized Control of State Variables	General	High	Resolved *
IDX-002	Use of Upgradable Contract Design	General	High	Resolved *
IDX-003	Improper Reward Calculation on _withUpdate Parameter	Advanced	Medium	Resolved *
IDX-004	Design Flaw in massUpdatePool() Function	General	Low	Resolved *
IDX-005	Transaction Ordering Dependence for _reinvest() Function	General	Low	Acknowledged
IDX-006	Use of Outdated Solidity Compiler Version	General	Very Low	Acknowledged
IDX-007	Insufficient Logging for Privileged Functions	General	Very Low	Resolved

* The mitigations or clarifications by Alpaca Finance can be found in Chapter 5.

5. Detailed Findings Information

5.1. Centralized Control of State Variables

ID	IDX-001
Target	SpookySwapStrategyAddBaseTokenOnly SpookySwapStrategyAddTwoSidesOptimal SpookySwapStrategyLiquidate SpookySwapStrategyPartialCloseLiquidate SpookySwapStrategyPartialCloseMinimizeTrading SpookySwapStrategyWithdrawMinimizeTrading SpookyWorker03 Rewarder1 MiniFL
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood:** Medium There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner.
Status	Resolved * Alpaca Finance team has confirmed that the upgradable contracts will be upgraded through the `Timelock` contract. This means any action that would occur to the upgradeable contracts will be able to be monitored by the community conveniently. However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the effect of the `Timelock` contract before using them.

5.1.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

For example, in the `MiniFL` contract. The wallet address with `onlyOwner` role can set the `maxAlpacaPerSecond` state via the `setMaxAlpacaPerSecond()` function for changing the limitation of reward per second any time as shown below:

MiniFL.sol

```
function setMaxAlpacaPerSecond(uint256 _maxAlpacaPerSecond) external onlyOwner
{
  if (_maxAlpacaPerSecond <= alpacaPerSecond) revert MiniFL_InvalidArguments();
  maxAlpacaPerSecond = _maxAlpacaPerSecond;
  emit LogSetMaxAlpacaPerSecond(_maxAlpacaPerSecond);
}
```

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

Target	Contract	Function	Modifier
SpookySwapStrategyAddBaseTokenOnly.sol (L:104)	SpookySwapStrategyAddBaseTokenOnly	setWorkersOk()	onlyOwner
SpookySwapStrategyAddTwoSidesOptimal.sol (L:161)	SpookySwapStrategyAddTwoSidesOptimal	setWorkersOk()	onlyOwner
SpookySwapStrategyLiquidate.sol (L:86)	SpookySwapStrategyLiquidate	setWorkersOk()	onlyOwner
SpookySwapStrategyPartialCloseLiquidate.sol (L:104)	SpookySwapStrategyPartialCloseLiquidate	setWorkersOk()	onlyOwner
SpookySwapStrategyPartialCloseMinimizeTrading.sol (L:132)	SpookySwapStrategyPartialCloseMinimizeTrading	setWorkersOk()	onlyOwner
SpookySwapStrategyWithdrawMinimizeTrading.sol (L:111)	SpookySwapStrategyWithdrawMinimizeTrading	setWorkersOk()	onlyOwner
SpookyWorker03.sol (L:440)	SpookyWorker03	setReinvestConfig()	onlyOwner
SpookyWorker03.sol (L:458)	SpookyWorker03	setMaxReinvestBountyBps()	onlyOwner
SpookyWorker03.sol (L:470)	SpookyWorker03	setStrategyOk()	onlyOwner
SpookyWorker03.sol (L:481)	SpookyWorker03	setReinvestorOk()	onlyOwner

SpookyWorker03.sol (L:491)	SpookyWorker03	setRewardPath()	onlyOwner
SpookyWorker03.sol (L:503)	SpookyWorker03	setCriticalStrategies()	onlyOwner
SpookyWorker03.sol (L:513)	SpookyWorker03	setTreasuryConfig()	onlyOwner
SpookyWorker03.sol (L:527)	SpookyWorker03	setBeneficialVaultConfig()	onlyOwner
Rewarder1.sol (L:169)	Rewarder1	setRewardPerSecond()	onlyOwner
Rewarder1.sol (L:191)	Rewarder1	addPool()	onlyOwner
Rewarder1.sol (L:217)	Rewarder1	setPool()	onlyOwner
Rewarder1.sol (L:298)	Rewarder1	setName()	onlyOwner
Rewarder1.sol (L:305)	Rewarder1	setMaxRewardPerSecond()	onlyOwner
MiniFL.sol (L:97)	MiniFL	addPool()	onlyOwner
MiniFL.sol (L:140)	MiniFL	setPool()	onlyOwner
MiniFL.sol (L:162)	MiniFL	setAlpacaPerSecond()	onlyOwner
MiniFL.sol (L:343)	MiniFL	approveStakeDebtToken()	onlyOwner
MiniFL.sol (L:361)	MiniFL	setMaxAlpacaPerSecond()	onlyOwner

5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract.

Inspex suggests removing the affected functions. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a timelock mechanism to delay the changes for a reasonable amount of time, e.g. 24 hours

5.2. Use of Upgradable Contract Design

ID	IDX-002
Target	SpookySwapStrategyAddBaseTokenOnly SpookySwapStrategyAddTwoSidesOptimal SpookySwapStrategyLiquidate SpookySwapStrategyPartialCloseLiquidate SpookySwapStrategyPartialCloseMinimizeTrading SpookySwapStrategyWithdrawMinimizeTrading SpookyWorker03 Rewarder1 MiniFL
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The logic of affected contracts can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the users' funds anytime they want. **Likelihood: Medium** This action can be performed by the proxy owner without any restriction.
Status	Resolved * Alpaca Finance team has confirmed that the contracts will be under the `Timelock` contract as same as other contracts on Alpaca Finance. This means all critical state variables will be able to be monitored with delay though the `Timelock` contract. However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the effect of the `Timelock` contract before using them.

5.2.1. Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As these smart contracts can be deployed through a proxy contract, the logic of them can be modified by the owner anytime, making the smart contracts untrustworthy.

5.2.2. Remediation

Inspex suggests deploying the contracts without the proxy pattern or any solution that can make smart contracts upgradeable.

However, if the upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes e.g., 1 days. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

5.3. Improper Reward Calculation on _withUpdate Parameter

ID	IDX-003
Target	MiniFL Rewarder1
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium When the `addPool()` and `setPool()` functions are called without updating the pools, the reward will be miscalculated, leading to unfair reward distribution. **Likelihood:** Medium The issue occurs whenever the `totalAllocPoint`, the `alpacaPerSecond`, and the `rewardPerSecond` states are modified with the `_withUpdate` parameter set as false.
Status	Resolved * Alpaca Finance team has confirmed that `_withUpdate` parameter will be passed as `true` until the number of pools is too large to be executed in one transaction. When the pool size grows too large, the team will manually call the `updatePools()` function to update the rewards as a substitute for the `massUpdatePool()` function. The users should monitor the transactions and make sure that the `updatePools()` function is called whenever the `_withUpdate` parameter is set to false.

5.3.1. Description

In the `MiniFL` contract, it allows the users to do yield farming by depositing specific tokens to the defined pools. The contract owner (wallet address with `onlyOwner` role) can add a new pool and modify the existing pool through the `addPool()` and the `setPool()` function respectively.

MiniFL.sol

```
function addPool(
  uint256 _allocPoint,
  IERC20Upgradeable _stakingToken,
  IRewarder _rewarder,
  bool _isDebtTokenPool,
  bool _withUpdate
) external onlyOwner {
  if (address(_stakingToken) == address(ALPACA)) revert
MiniFL_InvalidArguments();
  if (isStakingToken[address(_stakingToken)]) revert MiniFL_DuplicatePool();
```

```
// Sanity check that the staking token is a valid ERC20 token.
_stakingToken.balanceOf(address(this));

if (_withUpdate) massUpdatePools();

totalAllocPoint = totalAllocPoint + _allocPoint;
stakingToken.push(_stakingToken);
rewarder.push(_rewarder);
isStakingToken[address(_stakingToken)] = true;

if (address(_rewarder) != address(0)) {
  // Sanity check that the rewarder is a valid IRewarder.
  _rewarder.name();
}

poolInfo.push(
  PoolInfo({
    allocPoint: _allocPoint.toUint64(),
    lastRewardTime: block.timestamp.toUint64(),
    accAlpacaPerShare: 0,
    isDebtTokenPool: _isDebtTokenPool
  })
);
emit LogAddPool(stakingToken.length - 1, _allocPoint, _stakingToken, _rewarder);
```

MiniFL.sol

```
function setPool(
  uint256 _pid,
  uint256 _allocPoint,
  IRewarder _rewarder,
  bool _overwrite,
  bool _withUpdate
) external onlyOwner {
  if (_withUpdate) massUpdatePools();

  totalAllocPoint = totalAllocPoint - poolInfo[_pid].allocPoint + _allocPoint;
  poolInfo[_pid].allocPoint = _allocPoint.toUint64();
  if (_overwrite) {
    // Sanity check that the rewarder is a valid IRewarder.
    _rewarder.name();
    rewarder[_pid] = _rewarder;
  }
  emit LogSetPool(_pid, _allocPoint, _overwrite ? _rewarder : rewarder[_pid], _overwrite);
}
```

The `addPool()` and the `setPool()` functions accept the `_withUpdate` parameter to determine whether to update the current reward calculation of the pool contract immediately or not through the `massUpdatePools()` function.

The `massUpdatePools()` function will call the `_updatePool()` function to update each pool's `pool.accAlpacaPerShare` state that is used for calculating the user's reward distribution.

MiniFL.sol

```
function _updatePool(uint256 pid) internal returns (PoolInfo memory) {
  PoolInfo memory pool = poolInfo[pid];
  if (block.timestamp > pool.lastRewardTime) {
    uint256 stakedBalance = stakingToken[pid].balanceOf(address(this));
    if (stakedBalance > 0) {
      uint256 timePast = block.timestamp - pool.lastRewardTime;
      uint256 alpacaReward = (timePast * alpacaPerSecond * pool.allocPoint) /
totalAllocPoint;
      pool.accAlpacaPerShare = pool.accAlpacaPerShare +
        ((alpacaReward * ACC_ALPACA_PRECISION) / stakedBalance).toUint128();
    }
    pool.lastRewardTime = block.timestamp.toUint64();
    poolInfo[pid] = pool;
    emit LogUpdatePool(pid, pool.lastRewardTime, stakedBalance,
pool.accAlpacaPerShare);
  }
  return pool;
}
```

As a result, if the contract owner passes `_withUpdate` as `false`, the reward will be calculated incorrectly since the `totalAllocPoint` is updated, but the pending reward of all other pools is not updated immediately with new `totalAllocaPoint` value.

For example:

Assuming that at `block.timestamp` is 1010000, `alpacaPerSecond` is set to 10 $ALPACA per sec, pool 0 `allocPoint` is set to 300, `totalAllocPoint` is set to 9605, and `pool.lastRewardTime` is set to 1010000.

Timestamp	Action
1010000	All pools' rewards are updated
1020000	A new pool is added using the add() function, causing the `totalAllocPoint` to be changed from 9605 to 10000
1030000	The pools' rewards are updated once again

From current logic, the total rewards allocated to the pool 0 during timestamp 1010000 to timestamp 1030000 is equal to 6,000.00 $ALPACA calculated using the following equation:

```
timePast = 1,030,000 - 1,010,000
alpacaReward = (timePast * alpacaPerSecond * pool.allocPoint) / totalAllocPoint
             = (20,000 * 10 * 300) / 10,000
             = 6,000.00
```

However, the rewards should be calculated by accounting for the original `totalAllocPoint` value during the period when it is not yet updated as follow:

- from timestamp 1,010,000 to timestamp 1,020,000, with a proportion of 300/9,605 = 3,123.37 $ALPACA
- from timestamp 1,020,000 to timestamp 1,030,000, with a proportion of 300/10,000 = 3,000.00 $ALPACA

The correct total $ALPACA rewards is 6,123.37 $ALPACA, which is different from the miscalculated reward by 123.37 $ALPACA.

Please note that this issue affects the following functions in the same way:

Target	Contract	Function	State Modified
Rewarder1.sol (L:169)	Rewarder1	setRewardPerSecond()	rewardPerSecond
Rewarder1.sol (L:191)	Rewarder1	addPool()	totalAllocPoint
Rewarder1.sol (L:217)	Rewarder1	setPool()	totalAllocPoint
MiniFL.sol (L:97)	MiniFL	addPool()	totalAllocPoint
MiniFL.sol (L:140)	MiniFL	setPool()	totalAllocPoint
MiniFL.sol (L:162)	MiniFL	setAlpacaPerSecond()	alpacaPerSecond

5.3.2. Remediation

Inspex suggests removing the `_withUpdate` parameter and always calling the `massUpdatePools()` before updating the `totalAllocPoint`, `alpacaPerSecond`, and `rewardPerSecond` states of `MiniFL` and `Rewarder1` contracts as shown in the following examples:

MiniFL.sol

```
function addPool(
  uint256 _allocPoint,
  IERC20Upgradeable _stakingToken,
  IRewarder _rewarder,
  bool _isDebtTokenPool
) external onlyOwner {
  if (address(_stakingToken) == address(ALPACA)) revert MiniFL_InvalidArguments();
  if (isStakingToken[address(_stakingToken)]) revert MiniFL_DuplicatePool();

  // Sanity check that the staking token is a valid ERC20 token.
  _stakingToken.balanceOf(address(this));

  massUpdatePools();

  totalAllocPoint = totalAllocPoint + _allocPoint;
  stakingToken.push(_stakingToken);
  rewarder.push(_rewarder);
  isStakingToken[address(_stakingToken)] = true;

  if (address(_rewarder) != address(0)) {
    // Sanity check that the rewarder is a valid IRewarder.
    _rewarder.name();
  }

  poolInfo.push(
    PoolInfo({
      allocPoint: _allocPoint.toUint64(),
      lastRewardTime: block.timestamp.toUint64(),
      accAlpacaPerShare: 0,
      isDebtTokenPool: _isDebtTokenPool
    })
  );
  emit LogAddPool(stakingToken.length - 1, _allocPoint, _stakingToken, _rewarder);
```

MiniFL.sol

```
function setPool(
  uint256 _pid,
  uint256 _allocPoint,
  IRewarder _rewarder,
  bool _overwrite
) external onlyOwner {
  // Update when the totalAllocPoint is changed
  if (poolInfo[_pid].allocPoint != _allocPoint.toUint64()) {
    massUpdatePools();
  }

  totalAllocPoint = totalAllocPoint - poolInfo[_pid].allocPoint + _allocPoint;
  poolInfo[_pid].allocPoint = _allocPoint.toUint64();
  if (_overwrite) {
    // Sanity check that the rewarder is a valid IRewarder.
    _rewarder.name();
    rewarder[_pid] = _rewarder;
  }
  emit LogSetPool(_pid, _allocPoint, _overwrite ? _rewarder : rewarder[_pid], _overwrite);
}
```

MiniFL.sol

```
function setAlpacaPerSecond(uint256 _alpacaPerSecond) external onlyOwner {
  if (_alpacaPerSecond > maxAlpacaPerSecond) revert MiniFL_InvalidArguments();

  massUpdatePools();
  alpacaPerSecond = _alpacaPerSecond;
  emit LogAlpacaPerSecond(_alpacaPerSecond);
}
```

5.4. Design Flaw in massUpdatePool() Function

ID	IDX-004
Target	MiniFL Rewarder1
Category	General Smart Contract Vulnerability
CWE	CWE-400: Uncontrolled Resource Consumption
Risk	**Severity:** Low **Impact:** Medium The `massUpdatePool()` function can be unusable due to excessive gas usage. **Likelihood:** Low It is very unlikely that the `poolInfo` size will be raised until the `massUpdatePool()` function is unusable.
Status	Resolved * Alpaca Finance team has confirmed that when the pool size grows too large, the team will manually call the `updatePools()` function to update the rewards as a substitute for the `massUpdatePool()` function. The users should monitor the transactions and make sure that the `updatePools()` function is called whenever the `_withUpdate` parameter is set to false.

5.4.1. Description

The `massUpdatePool()` and `_massUpdatePool()` function executes the `_updatePool()` function, which is a state modifying function for all added farms as shown below:

MiniFL.sol

```
function massUpdatePools() public nonReentrant {
  uint256 len = poolLength();
  for (uint256 i = 0; i < len; ++i) {
    _updatePool(i);
  }
}
```

Rewarder1.sol

```
function _massUpdatePools() internal {
  uint256 _len = poolLength();
  for (uint256 i = 0; i < _len; ++i) {
    _updatePool(poolIds[i]);
  }
}
```

With the current design, the added pools cannot be removed. They can only be disabled by setting the `pool.allocPoint` to 0. Even if a pool is disabled, the `_updatePool()` function for this pool is still called. Therefore, if new pools continue to be added to this contract, the `poolInfo.length` will continue to grow and this function will eventually be unusable due to excessive gas usage.

5.4.2. Remediation

Inspex suggests making the contract capable of removing unnecessary or ended pools to reduce the loop rounds in the `massUpdatePool()` function.

5.5. Transaction Ordering Dependence in _reinvest() Function

ID	IDX-005
Target	SpookyWorker03
Category	General Smart Contract Vulnerability
CWE	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
Risk	**Severity:** Low **Impact:** Medium The front-running attack can be performed, resulting in a bad swapping rate for the reinvestment and lower reward for the platform users. **Likelihood:** Low It is easy to perform the attack. However, with a low profit, there is low motivation to attack with this vulnerability.
Status	**Acknowledged** The Alpaca Finance team has acknowledged this issue and states that the reinvestment is done every time someone interacts with the pool. Hence, the yield to be sold and reinvest will be small and will not have much impact on the price.

5.5.1. Description

In the `SpookyWorker03` contract, the `_reinvest()` function is used for collecting and reinvesting the farm reward. This issue can be triggered via the `reinvest()` and the `work()` functions. A part of the reward is charged as a fee and sent to the treasury account or the reinvestor. After that, the reward is swapped to the base token in line 231 with the minimum swapping output amount set to 0, then sent to the strategy to get the LP token at line 234-235.

SpookyWorker03.sol

```
207 function _reinvest(
208   address _treasuryAccount,
209   uint256 _treasuryBountyBps,
210   uint256 _callerBalance,
211   uint256 _reinvestThreshold
212 ) internal {
213   // 1. Withdraw all the rewards. Return if reward <= _reinvestThershold.
214   spookyMasterChef.withdraw(pid, 0);
215   uint256 reward = boo.balanceOf(address(this));
216   if (reward <= _reinvestThreshold) return;
217
218   // 2. Approve tokens
```

```
219   boo.safeApprove(address(router), uint256(-1));
220   address(lpToken).safeApprove(address(spookyMasterChef), uint256(-1));
221
222   // 3. Send the reward bounty to the _treasuryAccount.
223   uint256 bounty = reward.mul(_treasuryBountyBps) / 10000;
224   if (bounty > 0) {
225     uint256 beneficialVaultBounty = bounty.mul(beneficialVaultBountyBps) / 10000;
226     if (beneficialVaultBounty > 0) _rewardToBeneficialVault(beneficialVaultBounty, _callerBalance);
227     boo.safeTransfer(_treasuryAccount, bounty.sub(beneficialVaultBounty));
228   }
229
230   // 4. Convert all the remaining rewards to BTOKEN.
231   router.swapExactTokensForTokens(reward.sub(bounty), 0, getReinvestPath(), address(this), now);
232
233   // 5. Use add Token strategy to convert all BaseToken without both caller balance and buyback amount to LP tokens.
234   baseToken.safeTransfer(address(addStrat), actualBaseTokenBalance().sub(_callerBalance));
235   addStrat.execute(address(0), 0, abi.encode(0));
236
237   // 6. Stake LPs for more rewards
238   spookyMasterChef.deposit(pid, lpToken.balanceOf(address(this)));
239
240   // 7. Reset approvals
241   boo.safeApprove(address(router), 0);
242   address(lpToken).safeApprove(address(spookyMasterChef), 0);
243
244   emit Reinvest(_treasuryAccount, reward, bounty);
245 }
```

From the source code above, the last parameter passed to the `addStrat.execute()` function is also set to 0. That parameter is used to check the minimum LP amount to gain from the liquidity provision at line 74, preventing excessive slippage from the token swapping.

SpookySwapStrategyAddBaseTokenOnly.sol

```
30 /// @notice This function is written base on fee=998, feeDenom=1000
31 /// @dev Execute worker strategy. Take BaseToken. Return LP tokens.
32 /// @param data Extra calldata information passed along to this strategy.
33 function execute(
34   address, /* user */
35   uint256, /* debt */
36   bytes calldata data
37 ) external override onlyWhitelistedWorkers nonReentrant {
```

```
  // 1. Find out what farming token we are dealing with and min additional LP tokens.
  uint256 minLPAmount = abi.decode(data, (uint256));
  IWorker03 worker = IWorker03(msg.sender);
  address baseToken = worker.baseToken();
  address farmingToken = worker.farmingToken();
  ISwapPairLike lpToken = worker.lpToken();
  // 2. Approve router to do their stuffs
  baseToken.safeApprove(address(router), uint256(-1));
  farmingToken.safeApprove(address(router), uint256(-1));
  // 3. Compute the optimal amount of baseToken to be converted to farmingToken.
  uint256 balance = baseToken.myBalance();
  (uint256 r0, uint256 r1, ) = lpToken.getReserves();
  uint256 rIn = lpToken.token0() == baseToken ? r0 : r1;
  // find how many baseToken need to be converted to farmingToken
  // Constants come from
  // 2-f = 2-0.002 = 1.998
  // 4(1-f) = 4*998*1000 = 3992000, where f = 0.0020 and 1,000 is a way to avoid floating point
  // 1998^2 = 3992004
  // 998*2 = 1996
  uint256 aIn = AlpacaMath.sqrt(rIn.mul(balance.mul(3992000).add(rIn.mul(3992004)))).sub(rIn.mul(1998)) / 1996;
  // 4. Convert that portion of baseToken to farmingToken.
  address[] memory path = new address[](2);
  path[0] = baseToken;
  path[1] = farmingToken;
  router.swapExactTokensForTokens(aIn, 0, path, address(this), now);
  // 5. Mint more LP tokens and return all LP tokens to the sender.
  (, , uint256 moreLPAmount) = router.addLiquidity(
    baseToken,
    farmingToken,
    baseToken.myBalance(),
    farmingToken.myBalance(),
    0,
    0,
    address(this),
    now
  );
  require(moreLPAmount >= minLPAmount, "insufficient LP tokens received");
  address(lpToken).safeTransfer(msg.sender, lpToken.balanceOf(address(this)));
  // 6. Reset approval for safety reason
  baseToken.safeApprove(address(router), 0);
  farmingToken.safeApprove(address(router), 0);
}
```

Therefore, any amount of LP token is accepted, allowing front-running attack to be performed, resulting in less LP token for the reinvestment and reduced reward for the platform users.

5.5.2. Remediation

Inspex suggests implementing a price oracle and using the price from the oracle to calculate the acceptable slippage. As an example, TWAP oracle can be used to get the price of the token pair from the on-chain data[2].

5.6. Use of Outdated Solidity Compiler Version

ID	IDX-006
Target	SpookySwapStrategyAddBaseTokenOnly SpookySwapStrategyAddTwoSidesOptimal SpookySwapStrategyLiquidate SpookySwapStrategyPartialCloseLiquidate SpookySwapStrategyPartialCloseMinimizeTrading SpookySwapStrategyWithdrawMinimizeTrading SpookyWorker03
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low From the list of known Solidity bugs, direct impact cannot be caused from those bugs themselves. **Likelihood:** Low From the list of known Solidity bugs, it is very unlikely that those bugs would affect these smart contracts.
Status	**Acknowledged** The Alpaca Finance team has acknowledged this issue and decided to keep the version as 0.6.6 as their dependencies and protocol are implemented in 0.6.6.

5.6.1. Description

The Solidity compiler versions specified in the smart contracts were outdated. These versions have publicly known inherent bugs[3] that may potentially be used to cause damage to the smart contracts or the users of the smart contracts.

The outdated Solidity compiler contract are as follows:

Target	Contract	Version
SpookySwapStrategyAddBaseTokenOnly.sol (L: 14)	SpookySwapStrategyAddBaseTokenOnly	0.6.6
SpookySwapStrategyAddTwoSidesOptimal.sol (L: 14)	SpookySwapStrategyAddTwoSidesOptimal	0.6.6
SpookySwapStrategyLiquidate.sol (L: 14)	SpookySwapStrategyLiquidate	0.6.6

SpookySwapStrategyPartialCloseLiquidate.sol (L: 14)	SpookySwapStrategyPartialCloseLiquidate	0.6.6
SpookySwapStrategyPartialCloseMinimizeTrading.sol (L: 14)	SpookySwapStrategyPartialCloseMinimizeTrading	0.6.6
SpookySwapStrategyWithdrawMinimizeTrading.sol (L: 14)	SpookySwapStrategyWithdrawMinimizeTrading	0.6.6
SpookyWorker03.sol (L: 14)	SpookyWorker03	0.6.6

5.6.2. Remediation

Inspex suggests upgrading the Solidity compiler to the latest stable version[4].

During the audit activity, the latest stable versions of Solidity compiler for major version 0.6 is `v0.6.12`.

5.7. Insufficient Logging for Privileged Functions

ID	IDX-007
Target	SpookySwapStrategyAddBaseTokenOnly SpookySwapStrategyAddTwoSidesOptimal SpookySwapStrategyLiquidate SpookySwapStrategyPartialCloseLiquidate SpookySwapStrategyPartialCloseMinimizeTrading SpookySwapStrategyWithdrawMinimizeTrading
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users, reducing the chance of the users to act when irregular actions are done. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved Alpaca Finance team has resolved this issue by adding events to the necessary functions as suggested in commit `4553a34a6dcfcfbf7aebc693bb5c5c6074c73129`.

5.7.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the owner can set the worker roles to any address by executing the `setWorkersOk()` function in the `SpookySwapStrategyLiquidate` contract, and no events are emitted.

SpookySwapStrategyLiquidate.sol

```
function setWorkersOk(address[] calldata workers, bool isOk) external onlyOwner {
  for (uint256 idx = 0; idx < workers.length; idx++) {
    okWorkers[workers[idx]] = isOk;
  }
}
```

The privileged functions that executed without log are as follows:

target	Contract	Function	Modifier
SpookySwapStrategyAddBaseTokenOnly.sol (L:104)	SpookySwapStrategyAddBaseTokenOnly	setWorkersOk()	onlyOwner
SpookySwapStrategyAddTwoSidesOptimal.sol (L:161)	SpookySwapStrategyAddTwoSidesOptimal	setWorkersOk()	onlyOwner
SpookySwapStrategyLiquidate.sol (L:86)	SpookySwapStrategyLiquidate	setWorkersOk()	onlyOwner
SpookySwapStrategyPartialCloseLiquidate.sol (L:104)	SpookySwapStrategyPartialCloseLiquidate	setWorkersOk()	onlyOwner
SpookySwapStrategyPartialCloseMinimizeTrading.sol (L:132)	SpookySwapStrategyPartialCloseMinimizeTrading	setWorkersOk()	onlyOwner
SpookySwapStrategyWithdrawMinimizeTrading.sol (L:111)	SpookySwapStrategyWithdrawMinimizeTrading	setWorkersOk()	onlyOwner

5.7.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

SpookySwapStrategyLiquidate.sol

```
86 event LogSetWorkerOk(address[] indexed workers, bool isOk);
87
88 function setWorkersOk(address[] calldata workers, bool isOk) external onlyOwner
   {
89   for (uint256 idx = 0; idx < workers.length; idx++) {
90     okWorkers[workers[idx]] = isOk;
91   }
92   emit LogSetWorkerOk(workers, isOk);
93 }
```

6. Appendix

6.1. About Inspex

CYBERSECURITY PROFESSIONAL SERVICE

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] “Oracles | Uniswap” [Online]. Available: https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles. [Accessed: 01-Febuary-2022]

[3] “'List of Known Bugs — Solidity 0.8.12 documentation'” [Online]. Available: https://docs.soliditylang.org/en/latest/bugs.html. [Accessed: 01-Febuary-2022]

[4] “Releases · ethereum/solidity” [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 01-Febuary-2022]

inspex

CYBERSECURITY PROFESSIONAL SERVICE