WusdAlpacaStrategy

Smart Contract Audit Report
Prepared for Wault Finance

Date Issued:	Sep 22, 2021
Project ID:	AUDIT2021027
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021027
Version	v1.0
Client	Wault Finance
Project	WusdAlpacaStrategy
Auditor(s)	Pongsakorn Sommalai
Author	Pongsakorn Sommalai
Reviewer	Weerawat Pawanawiwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Sep 22, 2021	Full report	Pongsakorn Sommalai

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 Wault Finance, Inspex team conducted an audit to verify the security posture of the WusdAlpacaStrategy smart contract on Sep 21, 2021. During the audit, Inspex team examined the smart contract and the overall operation within the scope to understand the overview of WusdAlpacaStrategy smart contract. 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 1 low and 1 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved in the reassessment. Therefore, Inspex trusts that WusdAlpacaStrategy smart contract has high-level protections in place to be safe from most attacks.

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

Wault Finance is a decentralized finance hub that connects all of the primary DeFi use-cases within one simple ecosystem. In short, an all-in-one DeFi Platform!

`WusdAlpacaStrategy` smart contract is used to stake the collateral $USDT from `WUSDMaster` contract to the Alpaca USDT vault and to stake the gained $ibUSDT to the Alpaca FairLaunch contract. The reward from both contracts will be sent to the `Treasury` wallet and the principal fund can only be sent back to the `WUSDMaster` contract.

Scope Information:

Project Name	WusdAlpacaStrategy
Website	https://app.wault.finance/bsc/index.html#wusd
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Sep 21, 2021 - Sep 21, 2021
Reassessment Date	Sep 22, 2021

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: 9f9875824c8b2fcbf152475b90a6d4cfb0ceb7b2)

Contract	Location (URL)
WusdAlpacaStrategy	https://github.com/WaultFinance/WUSD/blob/9f9875824c/WusdAlpacaStrategy.sol

Reassessment: (Commit: bec7c99e90e13f29b4f44a9abebbdfcc957403a3)

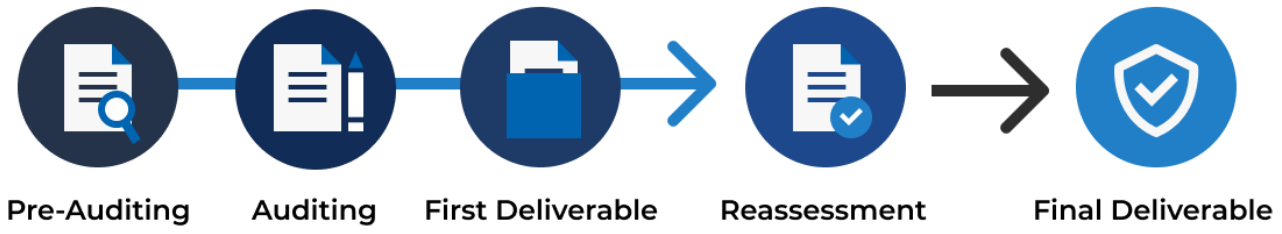
Contract	Location (URL)
WusdAlpacaStrategy	https://github.com/WaultFinance/WUSD/blob/bec7c99e90/WusdAlpacaStrategy.sol

The assessment scope covers only the in-scope smart contract and the smart contract that it is inherited 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
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Use of Upgradable Contract Design
Insufficient Logging for Privileged Functions
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

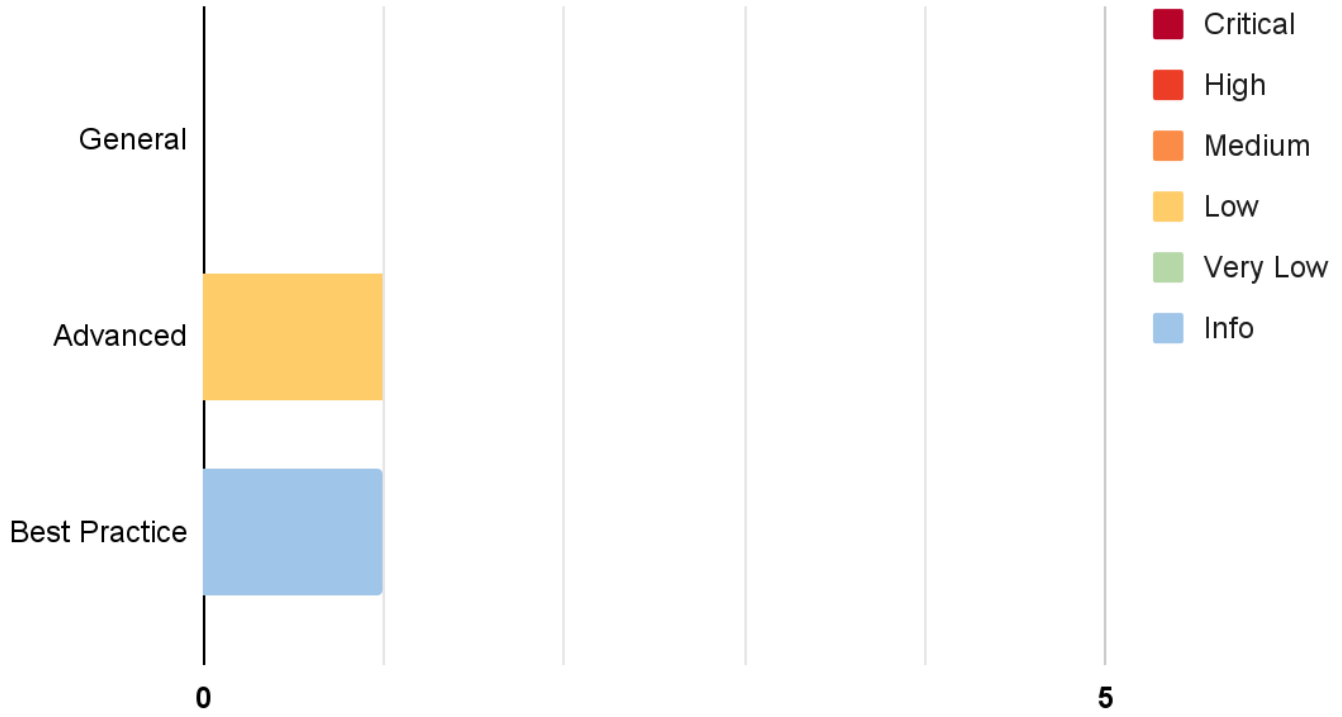
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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**.

Impact \ Likelihood	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 2 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	Missing Emergency Handling	Advanced	**Low**	Resolved
IDX-002	Redundant Calling of onlyOwner Modifier	Best Practice	**Info**	Resolved

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

5. Detailed Findings Information

5.1 Missing Emergency Handling

ID	IDX-001
Target	WusdAlpacaStrategy
Category	Advanced Smart Contract Vulnerability
CWE	CWE-544: Missing Standardized Error Handling Mechanism
Risk	**Severity:** Low **Impact:** Medium When the emergency case occurs, the $USDT will be stuck in the Alpaca FairLaunch contract. **Likelihood:** Low It is very unlikely that the normal `withdraw()` or `withdrawAll` functions of Alpaca FairLaunch contract will be broken.
Status	Resolved This issue has already been resolved by the Wault team in commit bec7c99e90e13f29b4f44a9abebbdfcc957403a3.

5.1.1. Description

When the `invest()` function of `WusdAlpacaStrategy` contract is called. The $USDT in the contract will be deposited to the Alpaca USDT vault and the gained $ibUSDT will be deposited to the Alpaca FairLaunch contract in line 543 as shown below:

WusdAlpacaStrategy.sol

```
533 function invest(uint256 usdtAmount) public onlyOwner {
534     require(usdt.balanceOf(address(this)) >= usdtAmount, 'not enough USDT in
    strategy');
535
536     usdt.approve(address(ibUsdt), usdtAmount);
537     ibUsdt.deposit(usdtAmount);
538     usdtInvestedAmount += usdtAmount;
539     uint256 ibUsdtAmountReceived = ibUsdt.balanceOf(address(this));
540     ibUsdtTotalAmount += ibUsdtAmountReceived;
541
542     ibUsdt.approve(address(stakeContract), ibUsdtAmountReceived);
543     stakeContract.deposit(address(this), stakePid, ibUsdtAmountReceived);
544
545     emit UsdtInvested(usdtAmount);
546 }
```

In the Alpaca FairLaunch contract, there is the `emergencyWithdraw()` function that can be used to withdraw all staked assets in the case that an emergency situation occurs as follows:

URL:
https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/token/FairLaunch.sol#L319-L328

```
319 function emergencyWithdraw(uint256 _pid) external nonReentrant {
320     PoolInfo storage pool = poolInfo[_pid];
321     UserInfo storage user = userInfo[_pid][msg.sender];
322     require(user.fundedBy == msg.sender, "only funder");
323     IERC20(pool.stakeToken).safeTransfer(address(msg.sender), user.amount);
324     emit EmergencyWithdraw(msg.sender, _pid, user.amount);
325     user.amount = 0;
326     user.rewardDebt = 0;
327     user.fundedBy = address(0);
328 }
```

However, in the `WusdAlpacaStrategy` contract, there is no function that can execute the `emergencyWithdraw()` function of Alpaca Fairlaunch contract.

Therefore, when the emergency case occurs, the $USDT will be stuck in the Alpaca FairLaunch contract.

5.1.2. Remediation

Inspex suggests implementing the `emergencyWithdraw()` function in the `WusdAlpacaStrategy` contract in order to execute the `emergencyWithdraw()` function of Alpaca Fairlaunch contract in line 612 as shown in the following example:

WusdAlpacaStrategy.sol

```
610 function emergencyWithdraw() external onlyOwner {
611     uint256 usdtStartAmount = usdt.balanceOf(address(this));
612     stakeContract.emergencyWithdraw(stakePid);
613     ibUsdt.withdraw(ibUsdt.balanceOf(address(this)));
614     ibUsdtTotalAmount = 0;
615     uint256 usdtWithdrawn = usdt.balanceOf(address(this)) - usdtStartAmount;
616     if(usdtWithdrawn > usdtInvestedAmount) {
617         usdt.safeTransfer(treasury(), usdtWithdrawn - usdtInvestedAmount);
618         usdt.safeTransfer(address(wusdMaster), usdtInvestedAmount);
619 
620         emit UsdtWithdrawnToTreasury(usdtWithdrawn - usdtInvestedAmount);
621         emit UsdtWithdrawnToMaster(usdtInvestedAmount);
622     } else {
623         usdt.safeTransfer(address(wusdMaster), usdtWithdrawn);
624 
625         emit UsdtWithdrawnToMaster(usdtWithdrawn);
626     }
627     usdtInvestedAmount = 0;
628 }
```

5.2. Redundant Calling of onlyOwner Modifier

ID	IDX-002
Target	WusdAlpacaStrategy
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved This issue has already been resolved by the Wault team in commit bec7c99e90e13f29b4f44a9abebbdfcc957403a3.

5.2.1. Description

The `onlyOwner` modifier is called twice in the following scenarios:

Scenario 1: investAll() function

When the `investAll()` function is called, the `invest()` function will be called and the `onlyOwner` modifier is set to both `investAll()` and `invest()` functions as shown in the following source code:

WusdAlpacaStrategy.sol

```
529 function investAll() external onlyOwner {
530     invest(usdt.balanceOf(address(this)));
531 }
532
533 function invest(uint256 usdtAmount) public onlyOwner {
534     require(usdt.balanceOf(address(this)) >= usdtAmount, 'not enough USDT in
    strategy');
535
536     usdt.approve(address(ibUsdt), usdtAmount);
537     ibUsdt.deposit(usdtAmount);
538     usdtInvestedAmount += usdtAmount;
539     uint256 ibUsdtAmountReceived = ibUsdt.balanceOf(address(this));
540     ibUsdtTotalAmount += ibUsdtAmountReceived;
541
542     ibUsdt.approve(address(stakeContract), ibUsdtAmountReceived);
543     stakeContract.deposit(address(this), stakePid, ibUsdtAmountReceived);
544
545     emit UsdtInvested(usdtAmount);
```

```
546 }
```

Scenario 2: withdrawToMaster() and withdrawRewards() functions

When the `withdrawToMaster()` and `withdrawRewards()` functions are called, the `withdraw()` function will be called and the `onlyOwner` modifier is set to all `withdrawToMaster()`, `withdrawRewards()`, and `withdraw()` functions as shown in the following source code:

WusdAlpacaStrategy.sol

```
548 function withdraw(address to, uint256 ibUsdtAmount) internal onlyOwner returns
    (uint256) {
549     uint256 usdtStartAmount = usdt.balanceOf(address(this));
550
551     stakeContract.withdraw(address(this), stakePid, ibUsdtAmount);
552     alpaca.safeTransfer(treasury(), alpaca.balanceOf(address(this)));
553
554     ibUsdt.withdraw(ibUsdtAmount);
555     uint256 usdtWithdrawn = usdt.balanceOf(address(this)) - usdtStartAmount;
556     usdt.safeTransfer(to, usdtWithdrawn);
557     ibUsdtTotalAmount -= ibUsdtAmount;
558
559     return usdtWithdrawn;
560 }
561
562 function withdrawToMaster(uint256 ibUsdtAmount) external onlyOwner {
563     require(ibUsdtAmount <= ibUsdtTotalAmount, 'not enough ibUSDT');
564     require(calculateUsdtAmount(ibUsdtAmount)  <= usdtInvestedAmount,
    'withdrawing more than invested');
565     uint256 usdtWithdrawn = withdraw(address(wusdMaster), ibUsdtAmount);
566     if(usdtWithdrawn < usdtInvestedAmount)
567         usdtInvestedAmount -= usdtWithdrawn;
568     else
569         usdtInvestedAmount = 0;
570
571     emit UsdtWithdrawnToTreasury(usdtWithdrawn);
572 }
573
574 function withdrawRewards(uint256 ibUsdtAmount) external onlyOwner {
575     require(ibUsdtAmount <= ibUsdtTotalAmount, 'not enough ibUSDT');
576     require(calculateUsdtAmount(ibUsdtTotalAmount - ibUsdtAmount) >=
    usdtInvestedAmount, 'withdrawing more than rewards');
577     uint256 usdtWithdrawn = withdraw(treasury(), ibUsdtAmount);
578
579     emit UsdtWithdrawnToTreasury(usdtWithdrawn);
580 }
```

5.2.2. Remediation

Inspex suggests removing `onlyOwner` modifier from the following functions:

- `withdraw()`
- `investAll()`

For example:

WusdAlpacaStrategy.sol

```
529 function investAll() external {
    ...
548 function withdraw(address to, uint256 ibUsdtAmount) internal returns (uint256)
    {
```

6. Appendix

6.1. About Inspex

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]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE