Lending

Smart Contract Audit Report
Prepared for Welnance

Date Issued:	Dec 17, 2021
Project ID:	AUDIT2021041
Version:	v1.0
Confidentiality Level:	Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021041
Version	v1.0
Client	Welnance
Project	Lending
Auditor(s)	Suvicha Buakhom Peeraphut Punsuwan
Author	Peeraphut Punsuwan
Reviewer	Weerawat Pawanawiwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Dec 17, 2021	Full report	Peeraphut Punsuwan

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 Welnance, Inspex team conducted an audit to verify the security posture of the Lending smart contracts between Nov 9, 2021 and Nov 18, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Lending 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 3 critical, 2 high, 2 very low, and 2 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or mitigated in the reassessment. Therefore, Inspex trusts that Lending smart contracts have 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

Welnance Finance is a platform that provides Decentralized Exchange, Swap, Staking, and Yield Farming Pools, along with other upcoming features such as, Lottery Lucky Draw, reward, and most importantly, Official Welnance Token.

Lending is a feature of Welnance on the Binance Smart Chain that establishes money markets, which are pools of assets with algorithmically derived interest rates, based on the supply and demand for the asset. Asset suppliers and borrowers can engage directly with the protocol, earning and paying a variable interest rate without having to negotiate terms like maturity, interest rate, or collateral with a peer or counterparty.

Scope Information:

Project Name	Lending
Website	https://welnance.finance/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Nov 9, 2021 - Nov 18, 2021
Reassessment Date	Dec 8, 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: 2cd2024351891b47fcd32d1515f79b19093bafb9)

Contract	Location (URL)
GovernorAlpha	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Governance/GovernorAlpha.sol
WelLens	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Lens/WelLens.sol
Comptroller	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Comptroller.sol
DAIInterestRateModelV2	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/DAIInterestRateModelV2.sol
Exponential	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Exponential.sol
ExponentialNoError	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/ExponentialNoError.sol
JumpRateModel	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/JumpRateModel.sol
Maximillion	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Maximillion.sol
Migrations	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Migrations.sol
PriceOracleProxy	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/PriceOracleProxy.sol
Reservoir	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Reservoir.sol
SimplePriceOracle	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/SimplePriceOracle.sol
Timelock	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Timelock.sol
Unitroller	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/Unitroller.sol

WBep20	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WBep20.sol
WBep20Delegate	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WBep20Delegate.sol
WBep20Delegator	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WBep20Delegator.sol
WBep20Immutable	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WBep20Immutable.sol
WelChainlinkOracle	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WelChainlinkOracle.sol
WelLikeDelegate	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WelLikeDelegate.sol
WelPriceOracle	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WelPriceOracle.sol
WhitePaperInterestRateModel	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WhitePaperInterestRateModel.sol
WLBNB	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WLBNB.sol
WLDaiDelegate	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WLDaiDelegate.sol
WLToken	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WLToken.sol
WLTreasury	https://github.com/Welnance/smart-contract/blob/2cd2024351/contracts/WLTreasury.sol

Reassessment: (Commit: edef88b4d3881f6eb676cf8189af05632d945422)

Contract	Location (URL)
GovernorAlpha	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Governance/GovernorAlpha.sol
WelLens	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Lens/WelLens.sol
Comptroller	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/ComptrollerV2.sol
DAIInterestRateModelV2	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/DAIInterestRateModelV2.sol
Exponential	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Exponential.sol
ExponentialNoError	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/ExponentialNoError.sol
JumpRateModel	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/JumpRateModel.sol
Maximillion	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Maximillion.sol
Migrations	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Migrations.sol
PriceOracleProxy	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/PriceOracleProxy.sol
Reservoir	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Reservoir.sol
Timelock	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Timelock.sol
Unitroller	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/Unitroller.sol
WBep20	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WBep20.sol
WBep20Delegate	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WBep20Delegate.sol
WBep20Delegator	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contract

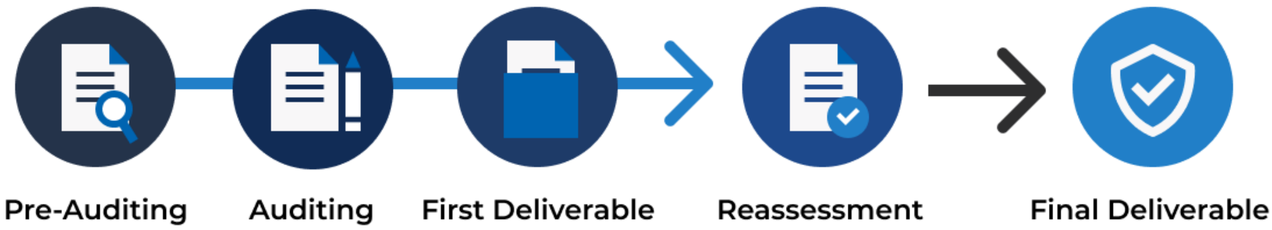
	s/WBep20Delegator.sol
WBep20Immutable	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WBep20Immutable.sol
WelChainlinkOracle	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WelChainlinkOracle.sol
WelLikeDelegate	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WelLikeDelegate.sol
WelPriceOracleV3	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WelPriceOracleV3.sol
WhitePaperInterestRateModel	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WhitePaperInterestRateModel.sol
WLBNB	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WLBNB.sol
WLDaiDelegate	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WLDaiDelegate.sol
WLToken	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WLToken.sol
WLTreasury	https://github.com/Welnance/smart-contract/blob/edef88b4d3/contracts/WLTreasury.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
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

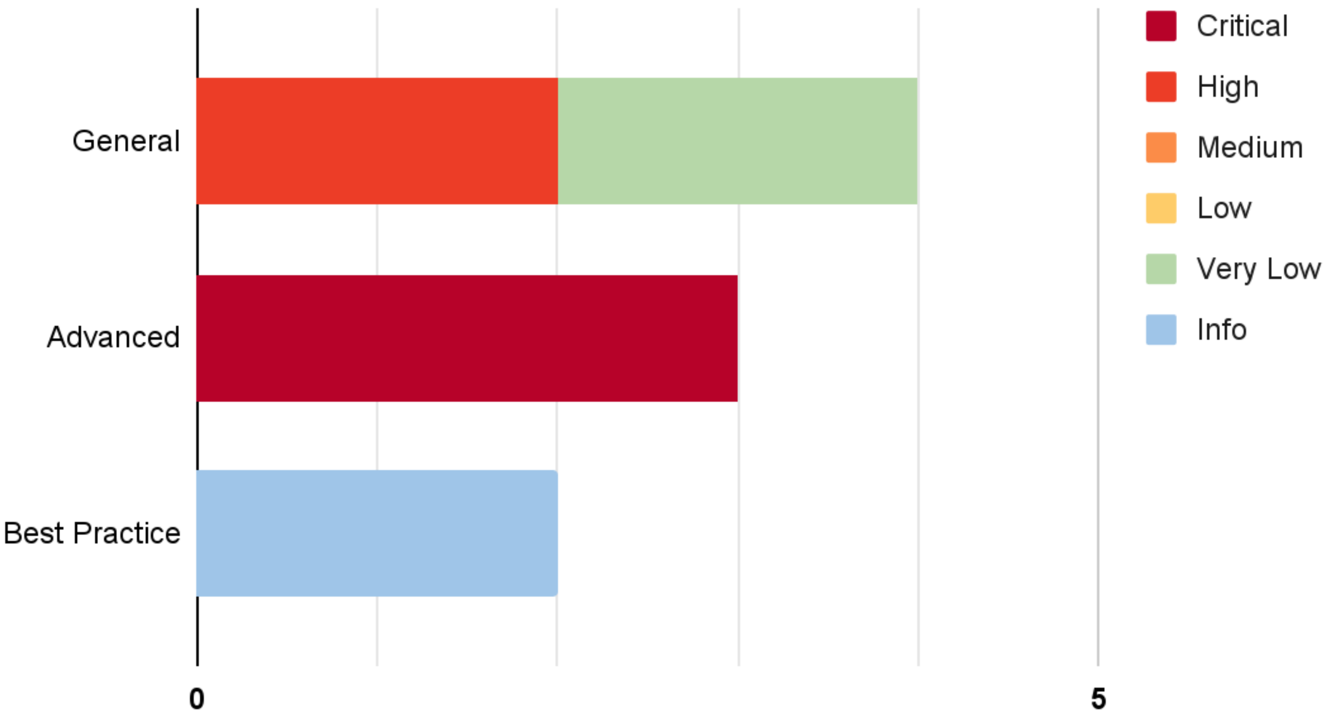
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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 9 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	Arbitrary Token Address Setting	Advanced	Critical	Resolved
IDX-002	Arbitrary Price Setting	Advanced	Critical	Resolved
IDX-003	Use of Unsafe Price Source	Advanced	Critical	Resolved
IDX-004	Use of Upgradable Contract Design	General	High	Resolved *
IDX-005	Centralized Control of State Variable	General	High	Resolved *
IDX-006	Unusable Token via Contract Freezing	General	Very Low	Resolved *
IDX-007	Outdated Compiler Version	General	Very Low	Resolved
IDX-008	Improper Function Visibility	Best Practice	Info	No Security Impact
IDX-009	Inexplicit Solidity Compiler Version	Best Practice	Info	No Security Impact

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

5. Detailed Findings Information

5.1. Arbitrary Token Address Setting

ID	IDX-001
Target	WelPriceOracle
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: Critical** **Impact: High** Anyone can change the token address of $BUSD and $WEL to a malicious token and use the price of the malicious token to gain profit in the platform, e.g., supplying, and borrowing. **Likelihood: High** It is very likely that the attacker will set the new token address to malicious token because the setting function can be called by anyone.
Status	Resolved The Welnance team has resolved this issue as suggested in the `WelPriceOracleV3` contract in commit `5efe49fb5551080dd3160e412248004f6d602fca`.

5.1.1. Description

The `setTokenBUSD()` and the `setTokenWel()` functions that are used to set the token address of $BUSD and $WEL have their function visibilities set to `external`, so anyone can call these functions.

WelPriceOracle.sol

```
function setTokenBUSD (address _tokenAddress) external {
    tokenBUSD = _tokenAddress;
}

function setTokenWel (address _tokenAddress) external {
    tokenWel = _tokenAddress;
}
```

The `tokenBUSD` and `tokenWel` states that are set by calling the `setTokenBUSD()` and the `setTokenWel()` functions are used as a parameter for the `getAmountsOut()` function in the `getUnderlyingPrice()` function in the case that `wlToken` symbol is `wlWEL`.

WelPriceOracle.sol

```
function getUnderlyingPrice(WLToken wlToken) public view returns (uint) {
    if (compareStrings(wlToken.symbol(), "wlWEL")) {
        return getAmountsOut(1000000000000000000,tokenWel, tokenBUSD);
    } else if (compareStrings(wlToken.symbol(), "wlBNB")) {
        IStdReference.ReferenceData memory data = ref.getReferenceData("BNB", "USD");
        return data.rate;
    } else {
        uint256 price;
        BEP20Interface token = BEP20Interface(WBep20(address(wlToken)).underlying());

        if(prices[address(token)] != 0) {
            price = prices[address(token)];
        } else {
            IStdReference.ReferenceData memory data = ref.getReferenceData(token.symbol(), "USD");
            price = data.rate;
        }

        uint decimalDelta = 18-uint(token.decimals());
        return price.mul(10**decimalDelta);
    }
}
```

The `getAmountOut()` function in the `WelPriceOracle` contract is passed to the `getAmountOut()` function of the router contract.

WelPriceOracle.sol

```
function getAmountsOut(uint256 _amountIn, address token0, address token1) public view returns (uint256 result) {
    address[] memory pairToken = new address[](2);
    pairToken[0] = token0;
    pairToken[1] = token1;
    uint256[] memory results = IPancakeRouter(routerAddress).getAmountsOut(_amountIn, pairToken);
    result = results[1];
}
```

That means the `getUnderlyingPrice()` function returns the price of $WEL by asking the price of any token that is set by the `setTokenBUSD()` and the `setTokenWel()` functions.

If the `tokenWel` is set to another token with a very high value, $WEL can be used as collateral and borrow a high value of asset.

5.1.2. Remediation

The address of tokens should not be changed, Inspex suggests removing the `setTokenBUSD()` and the `setTokenWel()` functions.

5.2. Arbitrary Price Setting

ID	IDX-002
Target	SimplePriceOracle
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: Critical** **Impact: High** The price of assets can be manipulated by anyone to gain a huge profit when supplying or borrowing on the platform. **Likelihood: High** The price setting function can be called by anyone.
Status	Resolved Welnance team has resolved this issue as suggested in the `WelPriceOracleV3` contract in commit `edef88b4d3881f6eb676cf8189af05632d945422`.

5.2.1. Description

The `setDirectPrice()` and the `setUnderlyingPrice()` functions that are used to set the token price have their visibility set to the public, so anyone can use these functions to set the token price.

SimplePriceOracle.sol

```
function setUnderlyingPrice(WLToken wlToken, uint underlyingPriceMantissa)
public {
    address asset = address(WBep20(address(wlToken)).underlying());
    emit PricePosted(asset, prices[asset], underlyingPriceMantissa,
underlyingPriceMantissa);
    prices[asset] = underlyingPriceMantissa;
}

function setDirectPrice(address asset, uint price) public {
    emit PricePosted(asset, prices[asset], price, price);
    prices[asset] = price;
}
```

5.2.2. Remediation

The asset price should not be set manually. Inspex suggests using the price data from a trustable price oracle provider.

If the price of the needed assets are not available from other trustable sources, Inspex suggests using the time-weight average price[2].

5.3. Use of Unsafe Price Source

ID	IDX-003
Target	WelPriceOracle
Category	Advanced Smart Contract Vulnerability
CWE	CWE-807: Reliance on Untrusted Inputs in a Security Decision
Risk	**Severity:** Critical **Impact:** High The asset price on the platform can be arbitrarily manipulated by the flashloan attack. The users can use the manipulated price to do arbitrarily action, e.g., borrow more than the value of the collateral assets. **Likelihood:** High It is very likely that the price will be manipulated because the attack is very profitable.
Status	Resolved Welnance team has resolved this issue by implementing the TWAP Price Oracle with 1 hour period in commit `edef88b4d3881f6eb676cf8189af05632d945422` and confirmed that they will update the price in the oracle every hour.

5.3.1. Description

The `getAmountOut()` function gets the real-time price from PancakeSwap's router contract, which can be manipulated easily by flashloan attack.

WelPriceOracle.sol

```
function getAmountsOut(uint256 _amountIn, address token0, address token1)
public view returns (uint256 result) {
    address[] memory pairToken = new address[](2);
    pairToken[0] = token0;
    pairToken[1] = token1;
    uint256[] memory results = IPancakeRouter(routerAddress)
.getAmountsOut(_amountIn, pairToken);
    result = results[1];
}
```

The price depends on the reserves of $BUSD and $WEL in the token pair. The attacker can use the flashswap feature of PancakeSwap to flashloan $BUSD from another pair and swap the flashloaned $BUSD to $WEL to inflate the $WEL price.

If the platform uses the price of the inflated $WEL to calculate the value in the same transaction, the collateral value of $WEL will be massively inflated, allowing the attacker to make profit by supplying $WEL and borrow other assets with more value than the collateral provided.

5.3.2. Remediation

Inspex suggests using the price data from a trustable price oracle provider.

If the price of the needed assets are not available from other trustable sources, Inspex suggests using time-weight average price[2] instead of directly quoting from the reserves.

5.4. Use of Upgradable Contract Design

ID	IDX-004
Target	Comptroller WBep20Delegate
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The logic of the affected contract can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the user funds anytime they want. **Likelihood:** Medium This action can be performed by the proxy owner without any restriction.
Status	Resolved * The Welnance team has confirmed that they will mitigate this issue by implementing the timelock mechanism for the target contracts of this issue. At the time of reassessment, the timelock mechanism was not implemented yet. Users should check the owner of each contract to make sure that the timelock mechanism is implemented. Furthermore, the users should monitor the timelock for the upgrade of the contracts and act accordingly.

5.4.1. Description

The `Unitroller` contract can be used as the proxy of the `Comptroller` contract.

Unitroller.sol

```
function () external payable {
    // delegate all other functions to current implementation
    (bool success, ) = comptrollerImplementation.delegatecall(msg.data);

    assembly {
          let free_mem_ptr := mload(0x40)
          returndatacopy(free_mem_ptr, 0, returndatasize)

          switch success
          case 0 { revert(free_mem_ptr, returndatasize) }
          default { return(free_mem_ptr, returndatasize) }
    }
}
```

The `WBep20Delegator` contract can be used as the proxy of the `WBep20Delegate` contract.

WBep20Delegator.sol

```
function () external payable {
    require(msg.value == 0,"WBep20Delegator:fallback: cannot send value to fallback");

    // delegate all other functions to current implementation
    (bool success, ) = implementation.delegatecall(msg.data);

    assembly {
        let free_mem_ptr := mload(0x40)
        returndatacopy(free_mem_ptr, 0, returndatasize)

        switch success
        case 0 { revert(free_mem_ptr, returndatasize) }
        default { return(free_mem_ptr, returndatasize) }
    }
}
```

The logic of the contracts can be changed by setting a new implementation contract address in the proxy contract.

Unitroller.sol

```
function _setPendingImplementation(address newPendingImplementation) public returns (uint) {

    if (msg.sender != admin) {
        return fail(Error.UNAUTHORIZED, FailureInfo.SET_PENDING_IMPLEMENTATION_OWNER_CHECK);
    }

    address oldPendingImplementation = pendingComptrollerImplementation;

    pendingComptrollerImplementation = newPendingImplementation;

    emit NewPendingImplementation(oldPendingImplementation, pendingComptrollerImplementation);

    return uint(Error.NO_ERROR);
}
```

WBEP20Delegator.sol

```
function _setImplementation(address implementation_, bool allowResign, bytes memory becomeImplementationData) public {
    require(msg.sender == admin, "WBep20Delegator::_setImplementation: Caller must be admin");

```

```
    if (allowResign) {
delegateToImplementation(abi.encodeWithSignature("_resignImplementation()"));
    }

    address oldImplementation = implementation;
    implementation = implementation_;

delegateToImplementation(abi.encodeWithSignature("_becomeImplementation(bytes)", becomeImplementationData));

    emit NewImplementation(oldImplementation, implementation);
}
```

As the contracts can be upgraded, the owner of the proxy contract can change the logic of the contracts anytime, allowing the owner to modify the functions and include malicious code to the contracts.

5.4.2. Remediation

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

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., 24 hours. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

5.5. Centralized Control of State Variable

ID	IDX-005
Target	WEL Comptroller Migrations PriceOracleProxy Unitroller WBep20Delegate WBep20Delegator WelChainlinkOracle WelPriceOracle WLDaiDelegate WLToken Ownable WLTreasury WelLikeDelegate
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 by the owner. However, only some owner roles can call these functions to change the states.
Status	Resolved * The Welnance team has confirmed that they will mitigate this issue by implementing the timelock mechanism for the target contracts of this issue. At the time of reassessment, the timelock mechanism was not implemented yet. The users should check the owner of each contract to make sure that the timelock mechanism is implemented. Furthermore, the users should monitor the timelock for the execution of the privileged functions and act accordingly.

5.5.1. 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.

However, as the contract is not yet deployed, there is potentially no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier / Role
WEL.sol (L:18)	WEL	transferOwnership()	onlyOwner
Comptroller.sol (L:411)	Comptroller	_setPriceOracle()	admin
Comptroller.sol (L:420)	Comptroller	_setCloseFactor()	admin
Comptroller.sol (L:427)	Comptroller	_setCollateralFactor()	admin
Comptroller.sol (L:448)	Comptroller	_setLiquidationIncentive()	admin
Comptroller.sol (L:457)	Comptroller	_supportMarket()	admin
Comptroller.sol (L:476)	Comptroller	_setPauseGuardian()	admin
Comptroller.sol (L:485)	Comptroller	_setMarketBorrowCaps()	admin
Comptroller.sol (L:498)	Comptroller	_setBorrowCapGuardian()	onlyAdmin
Comptroller.sol (L:498)	Comptroller	_setTreasuryData()	admin
Comptroller.sol (L:675)	Comptroller	_setWelSpeed()	admin
Migrations.sol (L:15)	Migrations	_setWelSpeed()	owner
PriceOracleProxy.sol (L:508)	PriceOracleProxy	setSaiPrice()	guardian
Unitroller.sol (L:38)	Unitroller	_setPendingImplementation()	admin
Unitroller.sol (L:85)	Unitroller	_setPendingAdmin()	admin
WBep20Delegate.sol (L:20)	WBep20Delegate	_becomeImplementation()	admin
WBep20Delegate.sol (L:35)	WBep20Delegate	_resignImplementation()	admin
WBep20Delegator.sol (L:60)	WBep20Delegator	_setImplementation()	admin
WelChainlinkOracle.sol (L:64)	WelChainlinkOracle	setUnderlyingPrice()	onlyAdmin
WelChainlinkOracle.sol (L:70)	WelChainlinkOracle	setDirectPrice()	onlyAdmin
WelChainlinkOracle.sol (L:75)	WelChainlinkOracle	setFeed()	onlyAdmin
WelChainlinkOracle.sol (L:93)	WelChainlinkOracle	setAdmin()	onlyAdmin
WelPriceOracle.sol (L:88)	WelPriceOracle	setUnderlyingPrice()	admin

WelPriceOracle.sol (L:95)	WelPriceOracle	setDirectPrice()	admin
WelPriceOracle.sol (L:109)	WelPriceOracle	setAdmin()	admin
WLDaiDelegate.sol (L:30)	WLDaiDelegate	_becomeImplementation()	admin
WLDaiDelegate.sol (L:72)	WLDaiDelegate	_resignImplementation()	admin
WLToken.sol (L:1218)	WLToken	_setPendingAdmin()	admin
WLToken.sol (L:1268)	WLToken	_setComptroller()	admin
WLToken.sol (L:1307)	WLToken	_setReserveFactorFresh()	admin
WLToken.sol (L:1415)	WLToken	_reduceReservesFresh()	admin
WLToken.sol (L:1478)	WLToken	_setInterestRateModelFresh()	admin
Ownable.sol (L:53)	WLTreasury	renounceOwnership()	onlyOwner
Ownable.sol (L:62)	WLTreasury	transferOwnership()	onlyOwner
WLTreasury.sol (L:31)	WLTreasury	withdrawTreasuryBEP20()	onlyOwner
WLTreasury.sol (L:60)	WLTreasury	withdrawTreasuryBNB()	onlyOwner
WelLikeDelegate.sol (L:35)	WelLikeDelegate	_delegateWelLikeTo()	admin

5.5.2. Remediation

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

However, if modifications are needed, Inspex suggests implementing a community-run governance to control the use of these functions, or mitigating the risks of this issue by using a `Timelock` contract to delay the changes for a sufficient amount of time. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

If the timelock is used, for the `Comptroller` contract, the modifier of `validPauseState()` should be changed from `admin` role to another role before applying the timelock. This is because the pause function should be available for immediate use without waiting to handle the emergency cases.

5.6. Unusable Token via Contract Freezing

ID	IDX-006
Target	WEL
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Very Low **Impact:** Low $WEL holders cannot use tokens freely, resulting in loss of opportunities to gain profit from using the token. **Likelihood:** Low It is very unlikely that the contract owner will freeze the tokens in the normal situation because there is no profit for the owner when the token contract is frozen.
Status	Resolved * Welnance team has confirmed they will resolve this issue by transferring the ownership of WEL token contract to the `0xdead` address in the next phase. At the time of reassessment, the ownership of WEL token contract was not transferred to `0xdead` yet. The users should check the owner of the $WEL token contract to make sure that the ownership is transferred.

5.6.1. Description

The $WEL token is implemented with a freezing mechanism to prevent all tokens from being used when the contract owner decides to freeze it.

The contract owner can call the `freeze()` function to set `isLocked` state to 1 to freeze the contract.

WEL.sol

```
function freeze() public onlyOwner {
    isLocked = 1;

    emit Freezed();
}
```

The `isLocked` state is used by the `validLock` modifier.

WEL.sol

```
modifier validLock {
    require(isLocked == 0, "Token is locked");
    _;
```

```
34 }
```

The functions in the following functions are using the `validLock` modifier.

- approve()
- transfer()
- transferFrom()
- delegate()
- delegateBySig()

When the contract is frozen, all of them can not be used.

5.6.2. Remediation

Inspex suggests removing the freezing mechanism from the contract to ensure that the tokens can be transferred freely.

If the contract can not be redeployed, Inspex suggests transferring the ownership of the contract to `0xdead` address to disable the use of the owner functions.

However, if the freezing mechanism is needed to prevent any unexpected issue on the platform, it is recommended to implement this feature to other contract's logic instead, since the users' tokens should be freely transferable at any time.

5.7. Outdated Compiler Version

ID	IDX-007
Target	WelChainlinkOracle
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	Resolved The Welnance team has resolved this issue as suggested in the `WelPriceOracleV3` contract in commit `edef88b4d3881f6eb676cf8189af05632d945422`.

5.7.1. Description

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

WelChainlinkOracle.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.5.16;
```

5.7.2. Remediation

Inspex suggests upgrading the Solidity compiler to the latest stable version[4]. During the audit activity, the latest stable version of Solidity compiler in major 0.5 is v0.5.17.

5.8. Improper Function Visibility

ID	IDX-008
Target	GovernorAlpha WEL Comptroller PriceOracleProxy SimplePriceOracle Unitroller
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	**No Security Impact** The Welnance team has acknowledged this issue.

5.8.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

For example, the following source code shows that the `_setPriceOracle()` function of the `Comptroller` contract is set to public and it is never called from any internal function.

Comptroller.sol

```
function _setPriceOracle(PriceOracle newOracle) public returns (uint) {
    if (msg.sender != admin) {
        return fail(Error.UNAUTHORIZED,
FailureInfo.SET_PRICE_ORACLE_OWNER_CHECK);
    }
    PriceOracle oldOracle = oracle;
    oracle = newOracle;
    emit NewPriceOracle(oldOracle, newOracle);
    return uint(Error.NO_ERROR);
}
```

The following table contains all functions that have `public` visibility and are never called from any internal function.

File	Contract	Function
GovernorAlpha.sol (L:136)	GovernorAlpha	propose()
GovernorAlpha.sol (L:176)	GovernorAlpha	queue()
GovernorAlpha.sol (L:192)	GovernorAlpha	execute()
GovernorAlpha.sol (L:202)	GovernorAlpha	cancel()
GovernorAlpha.sol (L:248)	GovernorAlpha	castVote()
GovernorAlpha.sol (L:252)	GovernorAlpha	castVoteBySig()
GovernorAlpha.sol (L:281)	GovernorAlpha	__acceptAdmin()
GovernorAlpha.sol (L:286)	GovernorAlpha	__abdicate()
GovernorAlpha.sol (L:291)	GovernorAlpha	__queueSetTimelockPendingAdmin()
GovernorAlpha.sol (L:296)	GovernorAlpha	__executeSetTimelockPendingAdmin()
WEL.sol (L:18)	WEL	transferOwnership()
WEL.sol (L:36)	WEL	freeze()
WEL.sol (L:42)	WEL	unfreeze()
WEL.sol (L:192)	WEL	delegate()
WEL.sol (L:205)	WEL	delegateBySig()
Comptroller.sol (L:411)	Comptroller	_setPriceOracle()
Comptroller.sol (L:476)	Comptroller	_setPauseGuardian()
Comptroller.sol (L:508)	Comptroller	_setProtocolPaused()
Comptroller.sol (L:529)	Comptroller	_become()
Comptroller.sol (L:632)	Comptroller	claimWel()
Comptroller.sol (L:635)	Comptroller	claimWel()
Comptroller.sol (L:640)	Comptroller	claimWel()
Comptroller.sol (L:675)	Comptroller	_setWelSpeed()
PriceOracleProxy.sol (L:15)	PriceOracleProxy	setSaiPrice()
SimplePriceOracle.sol (L:18)	SimplePriceOracle	setUnderlyingPrice()

SimplePriceOracle.sol (L:24)	SimplePriceOracle	setDirectPrice()
Unitroller.sol (L:38)	Unitroller	_setPendingImplementation()
Unitroller.sol (L:58)	Unitroller	_acceptImplementation()
Unitroller.sol (L:85)	Unitroller	_setPendingAdmin()
Unitroller.sol (L:108)	Unitroller	_acceptAdmin()

5.8.2. Remediation

Inspex suggests changing all functions' visibility to `external` if they are not called from any `internal` function as shown in the following example:

Comptroller.sol

```
function _setPriceOracle(PriceOracle newOracle) external returns (uint) {
    if (msg.sender != admin) {
        return fail(Error.UNAUTHORIZED,
FailureInfo.SET_PRICE_ORACLE_OWNER_CHECK);
    }
    PriceOracle oldOracle = oracle;
    oracle = newOracle;
    emit NewPriceOracle(oldOracle, newOracle);
    return uint(Error.NO_ERROR);
}
```

5.9. Inexplicit Solidity Compiler Version

ID	IDX-009
Target	Comptroller ComptrollerV1Storage DAIInterestRateModelV2 InterestRateModel JumpRateModel Maximillion Migrations PriceOracle PriceOracleProxy Reservoir SimplePriceOracle Timelock Unitroller WBep20 WBep20Delegate WBep20Delegator WBep20Immutable WelLikeDelegate WelPriceOracle WhitePaperInterestRateModel WLBNB WLDaiDelegate WLToken WLTreasury
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact The Welnance team has acknowledged this issue.

5.9.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues, for example:

Comptroller.sol

```
pragma solidity ^0.5.16;
//SPDX-License-Identifier: MIT
```

The following table contains all targets which the inexplicit compiler version is declared.

Contract	Version
Comptroller	^0.5.16
ComptrollerV1Storage	^0.5.16
DAIInterestRateModelV2	^0.5.16
InterestRateModel	^0.5.16
JumpRateModel	^0.5.16
Maximillion	^0.5.16
Migrations	^0.5.16
PriceOracle	^0.5.16
PriceOracleProxy	^0.5.16
Reservoir	^0.5.16
SimplePriceOracle	^0.5.16
Timelock	^0.5.16
Unitroller	^0.5.16
WBep20	^0.5.16
WBep20Delegate	^0.5.16
WBep20Delegator	^0.5.16
WBep20Immutable	^0.5.16
WelLikeDelegate	^0.5.16
WelPriceOracle	^0.5.16
WhitePaperInterestRateModel	^0.5.16
WLBNB	^0.5.16
WLDaiDelegate	^0.5.16

WLToken	^0.5.16
WLTreasury	^0.5.16

5.9.2. Remediation

Inspex suggests fixing the solidity compiler to the latest stable version. At the time of audit, the latest stable version of Solidity compiler in major 0.5 is v0.5.17.

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] "TWAP Oracle" [Online]. Available: https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles. [Accessed: 01-November-2021]

[3] "List of Known Bugs — Solidity 0.5.16 documentation." [Online]. Available: https://docs.soliditylang.org/en/v0.5.16/bugs.html. [Accessed: 1-Oct-2021]

[4] ethereum, "Releases · ethereum/solidity." [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 1-Oct-2021]

inspex

CYBERSECURITY PROFESSIONAL SERVICE