Coinsult

Request your audit at coinsult.net

Advanced Manual

Smart Contract Audit

April 23, 2023

IPPool

Audit requested by

IPP Tech

Audit Summary

Project Name	IPP Tech
Website	https://ipptech.io/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	-
Audit Method	Static Analysis, Manual Review
Start date of audit	April 23, 2023

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

The audit of the Solidity codebase presented significant challenges due to its disorganized nature and lack of accompanying documentation. The convoluted structure of the code made it increasingly difficult to discern the intended functionality of individual components. Furthermore, the absence of any guiding documentation surrounding the functions made it even more arduous to review and assess the overall quality of the code. Despite these obstacles, the auditing process was diligently carried out to ensure the highest standards of security and stability.

Audit Scope

Source Code

Coinsult was comissioned by IPP to perform an audit based on the following code:

https://github.com/ippswap/ippswap-contracts

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used Tools

- ✓ Slither: Solidity static analysis framework
- ✓ Remix: IDE Developer Tool
- ✓ CWE: Common Weakness Enumeration
- ✓ SWC: Smart Contract Weakness Classification and Test Cases
- ✓ DEX: Testnet Blockchains

ipp-ceres-binder.sol

OverPoweredRole- setRewardAmounts() function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

OverPoweredRole- setAddRelationBNBFee() function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

OverPoweredRole- setRewardsToken() function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

Logical - Unchecked return value detected on **addRelation()**

ICeresCore(ceres).addRelation(child, parent);

Logical - Unchecked return value detected on **_afterAddedRelation()**

```
ICeresCore(ceres).distribute(rewardToken_, child, totalAmount_,
0,parentAmount_, grandpaAmount_);
```

Logical - Lack of zero wallet checks detected on **CeresBinderV3 constructor()**

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

Logical - The contract CeresBinderV3 was found to be missing these events on the function **addRelation()** which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract CeresBinderV3 was found to be missing these events on the function **_afterAddedRelation()** which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract CeresBinderV3 was found to be missing these events on the function **setRewardAmounts()** which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract CeresBinderV3 was found to be missing these events on the function **setRewardsToken()** which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract CeresBinderV3 was found to be missing these events on the function **setAddRelationBNBFee()** which would make it difficult or impossible to track these transactions off-chain.

Gas - **External** visibility modifier for functions can save gas costs compared to using the

public modifier

```
function setRewardAmounts(uint _selfAmount,uint _parentAmount,uint
 grandpaAmount) public onlyAdmin
```

Gas - **addRelation()** When inside the **require** statements, non-strict inequalities **(>=, <=)**

are usually costlier than strict equalities **(>, <).**

ipp-ido.sol

OverPoweredRole - The contract uses the "**whenNotPaused**" modifier to prevent certain functions from being executed when the contract is paused. However, if the owner of the contract is the only one who can pause the contract, this can create an overpowered role for the owner. In this case, the contract is tightly coupled to the owner, and only they can manually invoke critical functions.

OverPoweredRole - It is important to ensure that the **onlyAdmin** modifier is properly implemented on **setTimes()** and that only authorized individuals are able to access this function. This can be achieved by properly setting up and managing the admin role within the smart contract system.

OverPoweredRole - Function that updates a contract variable, it is important to ensure that the **onlyAdmin** modifier is properly implemented and that only authorized individuals are able to access this **setPayUsdtAmount()** function.

Additionally, it may be beneficial to include additional input validation checks to ensure that **_payUsdtAmount** is a valid and reasonable value.

OverPoweredRole - Function that updates a contract variable, it is important to ensure that the **onlyAdmin** modifier is properly implemented and that only authorized individuals are able to access this **setMaxPeriodInvest()** function.

Additionally, it may be beneficial to include additional input validation checks to ensure that **_maxPeriodInvest** is a valid and reasonable value

OverPoweredRole - It is important to ensure that the contract owner role is properly implemented and that only authorized individuals are able to access this function. Additionally, it may be beneficial to include additional input validation on **setPeriodDuration()** checks to ensure that duration is a valid and reasonable value.

OverPoweredRole - setIncomeTo() function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

Logical - Weak PRNG detected on **calcPeriodStart()** due to a modulo on **block.timestamp**, **now** or **blockhash**. These can be influenced by miners to some extent so they should be avoided.

Logical - Lack of zero wallet checks detected on **LPPIdo constructor()**

Logical - The contract CeresBinderV3 was found to be missing these events on the function **setPeriodDuration()** which would make it difficult or impossible to track these transactions off-chain.

Logical - Dangerous usage of **block.timestamp**. **block.timestamp** on **invest()** can be manipulated by miners. Avoid relying on block.timestamp.

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

Informaĭional - Uses literals with too many digits on **maxPeriodInvest**

Informaĭional - Block values as a proxy time detected on **invest()**. Block values can be manipulated by miners to some extent, which can lead to various types of attacks on the smart contract.

```
require(block.timestamp >= startTime,"ido not start");
require(block.timestamp < endTime,"ido ended");

uint periodStart = calcPeriodStart(block.timestamp, periodDuration);
_periodTotalInvestAmount =
```

Gas - When inside the **require** statements, non-strict inequalities **(>=, <=)** are usually costlier than strict equalities **(>, <).**

ipp-token.sol

OverPoweredRole - The rate is limited by a constant **RATE_PRECISION** which is set to **10000**. **setBuyFeeRate() and setSellFeeRate()** functions have a require statement that ensures the input rate does not exceed this value. However, the owner can still set a **very high fee rate** that can potentially harm the token holders.

OverPoweredRole - addOtherSwapPair / removeOtherSwapPair functions allows the contract owner to add a new swap pair to the **isOtherSwapPair** mapping. The function has a **onlyOwner** modifier, which ensures that only the contract owner can call the function. However, we found that this function has an overpowered role, which can potentially be abused by the contract owner to manipulate the mapping and cause unintended consequences.

Logical - Uninitialized state variables. Always initialize your variables: Explicitly

initialize all variables to their intended values in the constructor or elsewhere in the contract. This will make your code more predictable and easier to understand. **buyFeeRate**, **sellFeeRate**.

Logical - Passing multiple variable-length arguments inside **abi.encodePacked()**

may cause issues if the argument's data is controlled or influenced by external users.

This is because **abi.encodePacked()** packs all elements in order regardless of whether they're part of an array. Even if you modify the arrays, so long as the elements are in the same order, it will always return the same hash. Do not allow user controlled data inside **abi.encodePacked()**. Instead of dynamic arrays, use fixed-length arrays. Use **abi.encode()** instead of **abi.encodePacked()**

```
bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR,
structHash));
```

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

Logical - The contract IPP was found to be missing these events on the function **setBuyFeeRate() and setSellFeeRate()** which would make it difficult or impossible to track these transactions off-chain.

Logical - Lack of zero wallet checks detected on **setFeeTo().** Check that the address is not zero.

Informational - Literals with many digits are difficult to read and review. If possible, break down the large number into smaller, more manageable pieces. **16 * 1e6 * 1e18 uint256** ***private constant*** **_totalSupply = 16000000 * 1e18;**

Gas - decreaseAllowance(), increaseAllowance(), approve(), should be external

Gas - When inside the **require** statements, non-strict inequalities **(>=, <=)** are usually costlier than strict equalities **(>, <).**

Sl'akingFactory.sol

OverPoweredRole - The contract **StakingFactory** was found some over powered roles **setRewardOperator(), setRewardSigner()** functions are set by the owner. That function have an overpowered role, which can potentially be abused by the contract owner to manipulate the mapping and cause unintended consequences.

OverPoweredRole - The **setCreateETHFee()** function, which is set by the owner, may pose a security risk if there are no limits on the fee value. If the fee value is arbitrarily set or excessively very high, it could potentially prevent users from creating wallets, or the owner could exploit it to extract more value from users. It is recommended to set appropriate limits on the fee value.

OverPoweredRole - The **takeToken(), takeETH(), takeAllToken(),** and **takeAllETH()** functions have an overpowered role, as they allow the contract **owner** or **CFO** to transfer tokens and Ether from the contract to a specified address without any further restrictions. This could potentially be abused by the contract **owner** or **CFO** to manipulate the token or Ether balance and cause unintended consequences, making it a security risk.

OverPoweredRole & Logical- The contract **CfoTakeableV2** was found to be missing these events on the function **setCfo()** which would make it difficult or impossible to track these transactions off-chain.

OverPoweredRole & Logical - Lack of zero wallet checks detected on

setSwapRouter(). Check that the address is not zero.

Logical - The contract **StakingFactory** was found to be using a mapping (X) containing a struct (Y) on **create()**. This struct also contains another mapping (Z). The vulnerability arises when the an item is deleted from mapping (X). This does not delete data from mapping (Z). The remaining data may compromise the contract. ***delete tempDeployParams;*** Should be implemented instead of deletion to disable the struct containing the mapping.

Logical - Uninitialized state variables. Always initialize your variables: Explicitly

initialize all variables to their intended values in the constructor or elsewhere in the contract. This will make your code more predictable and easier to understand. **createETHFee**

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

Informal'ional - Block values as a proxy time detected on **create()**. Block values can be manipulated by miners to some extent, which can lead to various types of attacks on the smart contract.

StakingRewards.sol

OverPoweredRole - The contract **StakingRewards** was found some over powered roles **setPauseStatus()** functions are set by the owner. That function have an overpowered role, which can potentially be abused by the contract owner to manipulate the mapping and cause unintended consequences.

OverPoweredRole - The contract **StakingRewards** was found some over powered roles **transferCreator()** functions are set by the owner. That function have an overpowered role, which can potentially be abused by the contract owner to manipulate the mapping and cause unintended consequences.

OverPoweredRole - The **takeToken()** functions have an overpowered role, as they allow the contract **creator** to transfer tokens and Ether from the contract to a specified address without any further restrictions. This could potentially be abused by the contract **creator** to manipulate the token or Ether balance and cause unintended consequences, making it a security risk.

OverPoweredRole - The **setRewardRate()** function, which is set by the owner, may pose a security risk if there are no limits on the fee value. If the fee value is arbitrarily set or excessively very high, it could potentially prevent users from creating wallets, or the owner could exploit it to extract more value from users. It is recommended to set appropriate limits on the fee value.

Logical - Sending Ether to arbitrary addresses can be found on **_safeTransferETH()** and dangerous and should only be done with caution. In some cases, the recipient address could be a malicious contract that could exploit vulnerabilities in the sending contract. Ensure that an arbitrary user cannot withdraw unauthorized funds. Since the function **_safeTransferETH** is not marked as payable, the transaction will fail.

Logical - Weak PRNG detected on **_calcPeriodStart()** due to a modulo on **block.timestamp**, **now** or **blockhash**. These can be influenced by miners to some extent so they should be avoided.

Logical - Unchecked return value detected on **_transferFrom()**

IERC20(token).transferFrom(from, address(this), amount);

Logical - The **StakingReward** contract's **_removeLiquidityForRewards()** function was found to be using strict equalities that can be easily manipulated by an attacker.

Logical - The contract **StakingReward** was found to be missing these events on the function **transferCreator()** which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract **StakingReward** was found to be missing these events on the function **setRewardRate()** which would make it difficult or impossible to track these transactions off-chain.

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

Informational - Block values as a proxy time detected on **stake(), notifyRewards()**. Block values can be manipulated by miners to some extent, which can lead to various types of attacks on the smart contract.

Informational - **ecrecover** allows you to convert a valid signature into a different valid signature without requiring knowledge of the private key. It is usually not a problem unless you use signatures to identify items or require them to be uniquely recognizable. The data signer can be recovered using ECDSA.recover, and its address can be compared to verify the signature

Gas - When inside the **require** statements, non-strict inequalities **(>=, <=)** are usually costlier than strict equalities **(>, <).**

SwapFactory.sol

OverPoweredRole - The **setSwapFee()** function, which is set by the owner, may pose a security risk if there are no limits on the fee value. If the fee value is arbitrarily set or excessively very high, it could potentially prevent users from creating wallets, or the owner could exploit it to extract more value from users. It is recommended to set appropriate limits on the fee value.

OverPoweredRole & Logical - Lack of zero wallet checks detected on **setFeeTo().**

Check that the address is not zero.

OverPoweredRole - The **setProtocolFee()** function, which is set by the owner, may pose a security risk if there are no limits on the fee value. If the fee value is arbitrarily set or excessively very high, it could potentially prevent users from creating wallets, or the owner could exploit it to extract more value from users. It is recommended to set appropriate limits on the fee value.

OverPoweredRole - The contract **SwapFactory** was found some over powered roles **setRouter()** functions are set by the owner. That function have an overpowered role, which can potentially be abused by the contract owner to manipulate the mapping and cause unintended consequences.

Logical - The **require(msg.sender == router,"SwapFactory: caller must be router");** statement in the function checks that the caller of the function is the router address, and if not, the function will revert with an error message. **router** variable is not initialized in the **createPair** function. It is a public variable declared in the contract, and its value must be set before the **createPair** function can be called successfully.

Logical - The contract **SwapFactory** was found to be missing these events on the function

setFeeTo() which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract **SwapFactory** was found to be missing these events on the function

setSwapFee() which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract **SwapFactory** was found to be missing these events on the function

setProtocolFee() which would make it difficult or impossible to track these transactions off-chain.

Logical - The contract **SwapFactory** was found to be missing these events on the function **setRouter()** which would make it difficult or impossible to track these transactions off-chain.

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

Informational - Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This bypasses several important safety features and checks of Solidity. This should only be used for tasks that need it and if there is confidence in using it. Multiple vulnerabilities have been detected previously when the assembly is not properly used within the Solidity code; therefore, caution should be exercised while using them.

```
assembly {

     pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
```

Avoid using inline assembly instructions if possible because it might introduce certain issues in the code if not dealt with properly because it bypasses several safety features that are already implemented in Solidity.

Informational - The contract contains an unknown hard-coded address. This address might be used for some malicious activity. Please check the hard-coded address and its usage.

feeTo = address(0xCEF9dfAA5415b46d807712719641F08be0Da9080);

SwapPair.sol

Logical - If an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since **tx.origin** returns the original sender of the transaction which in this case is the authorized account.

```
emit Swaped(tx.origin, amount0In, amount1In, amount0Out,
amount1Out,block.timestamp);
```

You can implement a require of the form **require(tx.origin == msg.sender)**

Logical - Weak PRNG due to a modulo on **block.timestamp**, **now** or **blockhash**. These can be influenced by miners to some extent so they should be avoided.

uint32 blockTimestamp = uint32(block.timestamp % 2 ** 32);

Logical - The **low-level calls** such as the **delegatecall**, **call**, or **callcode**, do not validate prior to the call if the destination account exists or not. They will always return true even if the account is non-existent, therefore, giving invalid output.

```
(bool success, bytes memory data) =
token.call(abi.encodeWithSelector(SELECTOR, to, value));
```

Logical - The **SwapPair** contract's **_safeTransfer()** and **mint()** function was found to be using strict equalities that can be easily manipulated by an attacker.

Logical - Lack of zero wallet checks detected on **initialize().** Check that the address is not zero.

Logical - Outdated versions were detected **pragma solidity ^0.8.0;**

Logical - Floating pragma detected **pragma solidity ^0.8.0;**

SwapRouter.sol

OverPoweredRole - setPairCreator() functions allows the contract owner to add a new swap pair. The function has a **onlyCreator** modifier, which ensures that only the contract owner can call the function. However, we found that this function has an overpowered role, which can potentially be abused by the contract owner to manipulate the mapping and cause unintended consequences.

OverPoweredRole - setSellLpReceiver() functions allows the contract which is used to update the receiver address for selling LP tokens of a specific pair. The function has a **stakingFactory** or **owner** modifier, which ensures that only the contract owner can call the function. However, we found that this function has an overpowered role, which can potentially be abused by the contract owner to manipulate the mapping and cause unintended consequences.

OverPoweredRole - setWhiteList function, which is used to update the white list status of a specific account for a specific pair**.** The function requires that the caller must be either the **stakingFactory**, the **creator** of the pair, or the contract **owner**. If the caller is valid, the function updates the **isWhiteList** mapping for the specified pair and account

OverPoweredRole & Logical - Lack of zero wallet checks detected on

setStakingFactory(). Check that the address is not zero.

Logical - createPair(), addLiquidity(), addLiquidityETH(), removeLiquidity(), removeLiquidityETHSupportingFeeOnTransferTokens(), swapExactTokensForTokensSupportingFeeOnTransferTokens(), swapExactETHForTokensSupportingFeeOnTransferTokens(), swapExactTokensForETHSupportingFeeOnTransferTokens() Functions to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls. This may lead to loss of funds, improper value updates, token loss

Logical - Unchecked return value detected on **removeLiquidity()**

ISwapPair(pair).transferFrom(msg.sender, pair, liquidity);

Logical - Unchecked return value detected on **_addLiquidityForSell()**

_swapSupportingFeeOnTransferTokens(_flipPath(path), address(this));
ISwapPair(pair).mint(sellLpReceiverOf[pair]);

Logical - The **SwapRouter** contract's **_addLiquidityForSell()** function was found to be using strict equalities that can be easily manipulated by an attacker.

Logical - Lack of zero wallet checks detected on **SwapRouter constructor().**

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

SwapQuery.sol

Logical - The signature is calculated by hashing the function name and its parameter types. However, if there are spaces in the function signature calculation, the resulting hash will be different and the function signature will be calculated incorrectly. This can lead to logical errors and unexpected behavior. Ensure that there are no spaces in the function signature calculation. This can be done by carefully checking the function name and its parameter types, and ensuring that they are written without any extra spaces.

```
keccak256('EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)')
```

Logical - pairInfos(), getPairs(), getPairReserves() Calls inside a loop might lead to a denial-of-service attack.

Logical - Outdated versions were detected **pragma solidity 0.8.4;**

SwapERC20.sol

Logical - Dangerous usage of **block.timestamp** on **permit()**. block.timestamp can be manipulated by miners

Logical - Outdated versions were detected **pragma solidity 0.8.0;**

Logical - Floating pragma detected **pragma solidity 0.8.0;**

Notes

Notes by IPP

No notes provided by the team.

Notes by Coinsult

These contracts have been audited based on GitHub code, this code is updateable.

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

The audit of the Solidity codebase presented significant challenges due to its disorganized nature and lack of accompanying documentation. The convoluted structure of the code made it increasingly difficult to discern the intended functionality of individual components. Furthermore, the absence of any guiding documentation surrounding the functions made it even more arduous to review and assess the overall quality of the code. Despite these obstacles, the auditing process was diligently carried out to ensure the highest standards of security and stability.