CRACKEN

Smart Contract Security Audit

Project: SAS Coin

Aug 29, 2022

Contract Address

0x5DF47C286d5b66826bcF9DFd8234F94850a66A6A

Table of Contents

Disclaimer

The contents of this report reflect only the CRACKEN TECH audit team's understanding of the current progress and status of the security of the code audited, to verify the integrity of the code provided for the scope of this audit. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit are recommended after the issues covered are fixed. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

The review does not address the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from off-chain sources are not extended by this review either.

Audit Review

The source code of the SAS Coin was audited in order to acquire a clear impression of how the project was implemented. The Cracken Tech audit team conducted in-depth research, analysis, and scrutiny, resulting in a series of observations. A detailed list of each issue found, and vulnerabilities in the source code will be included in the audit report. The problems and potential solutions are given in this report, we will identify common sources for such problems and comments for improvement.

The auditing process will follow a routine as special considerations by Cracken:

- Review of the specifications, sources, and instructions provided to Cracken to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Cracken describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code is exercised when we run the test cases.
- Symbolic execution is analyzing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

Project Review

Token Summary

Parameter	Result
Token Name	SAS
Token Symbol	SAS
Token Decimal	9
Total Supply	10,000,000
Platform	BSC
Buy Tax Fee	4%
Sell Tax Fee	4%
Contract Creation Date	Aug 29, 2022
Liquidity Status	Not available when audit
Liquidity Lockup Time	365 Days
Compiler Version	v0.8.16+commit.07a7930e
Optimization	Yes with 200 runs
Contract Address	0x5DF47C286d5b66826bcF9DFd8234F94850a66A6A
Deployer Address	0xA5caF0a379E04Ceb454a2615b05937E85fdA375B
Owner Address	0xA5caF0a379E04Ceb454a2615b05937E85fdA375B

Source Code

CRACKEN was commissioned by SAS Coin to perform an audit based on the following smart contract:

https://bscscan.com/address/0x5DF47C286d5b66826bcF9DFd8234F94850a66A6A

Smart Contract Vulnerability Checks

Vulnerability	Auto-Scan	Manual-Scan	Result
Unencrypted Private Data On-Chain	Complete	Complete	Low / No Risk
Code With No Effects	Complete	Complete	Low / No Risk
Message call with hardcoded gas amount	Complete	Complete	Low / No Risk
Hash Collisions with Multiple Variable Length Arguments	Complete	Complete	Low / No Risk
Unexpected Ether balance	Complete	Complete	Low / No Risk
Presence of unused variables	Complete	Complete	Low / No Risk
Right-To-Left-Override control character (U+202E)	Complete	Complete	Low / No Risk
Typographical Error	Complete	Complete	Low / No Risk
DoS With Block Gas Limit	Complete	Complete	Low / No Risk
Arbitrary Jump with Function Type Variable	Complete	Complete	Low / No Risk
Insufficient Gas Grieving	Complete	Complete	Low / No Risk
Incorrect Inheritance Order	Complete	Complete	Low / No Risk
Write to Arbitrary Storage Location	Complete	Complete	Low / No Risk
Requirement Violation	Complete	Complete	Low / No Risk
Missing Protection against Signature Replay Attacks	Complete	Complete	Low / No Risk
Weak Sources of Randomness from Chain Attributes	Complete	Complete	Low / No Risk
Authorization through tx. origin	Complete	Complete	Low / No Risk
Delegate call to Untrusted Callee	Complete	Complete	Low / No Risk

Vulnerability	Auto-Scan	Manual-Scan	Result
Use of Deprecated Solidity Functions	Complete	Complete	Low / No Risk
Assert Violation	Complete	Complete	Low / No Risk
Reentrancy	Complete	Complete	Low / No Risk
Unprotected SELF-DESTRUCT Instruction	Complete	Complete	Low / No Risk
Unprotected Ether Withdrawal	Complete	Complete	Low / No Risk
Outdated Compiler Version	Complete	Complete	Low / No Risk
Integer Overflow and Underflow	Complete	Complete	Low / No Risk
Function Default Visibility	Complete	Complete	Low / No Risk

Manual Code Review

Classification of Issues

Severity	Description
🔴 High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
🟠 Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
🟡 Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
🔵 Informational	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
🔴 High-Risk	3
🟠 Medium-Risk	0
🟡 Low-Risk	0
🔵 Informational	0
Total	3

● **High-Risk: functions make cause the rug or scam project. Must be fixed.**

Set max buy / sell tax fee

Description:

The owner can change the buy & sell fees up to 100% [HIGH RISK]

```
function doWithB(uint256 a, uint256 b, uint256 c) external onlyOwner() {
          _buyLiquidityFee = a;
          _buyMarketingFee = b;
          _buyTeamFee = c;

          _totalTaxIfBuying = _buyLiquidityFee.add(_buyMarketingFee).add(_buyTeamFee);
     }

     function doWithS(uint256 a, uint256 b, uint256 c) external onlyOwner() {
          _sellLiquidityFee = a;
          _sellMarketingFee = b;
          _sellTeamFee = c;

          _totalTaxIfSelling = _sellLiquidityFee.add(_sellMarketingFee).add(_sellTeamFee);
     }
```

Recommendation:

We recommend adding a requirement to limit the max fee amount.

● **High-Risk: functions make cause the rug or scam project. Must be fixed.**

The trading function is enabled to be paused

Description:

The owner can change Max Transaction Amount without limit

[HIGH RISK]

```
function setMaxOnceEat(uint256 newMaxOnceEat) external onlyOwner() {
            _maxOnceEat = newMaxOnceEat;
    }


    function enableMaxEat(bool newValue) external onlyOwner {
        LookMaxEat = newValue;
    }
```

Recommendation:

We recommend adding a requirement to limit the max transaction amount.

● **High-Risk: functions make cause the rug or scam project. Must be fixed.**

The blacklist function is enabled

Description:

The owner can add blacklist users [HIGH RISK]

```
function manage_CantEat(address[] calldata addresses, bool status) public onlyOwner {
            require(addresses.length < 201);
            for (uint256 i; i < addresses.length; ++i) {
                  whoCantEat[addresses[i]] = status;
            }
      }
```

Recommendation:

We recommend that the owner should disable the blacklist function.

Privileged Functions

onlyOwner

Function Name	Parameters	Visibility
decreaseAllowance	address spender, uint256 subtractedValue	Public
doWithB	uint256 a, uint256 b, uint256 c	External
doWithS	uint256 a, uint256 b, uint256 c	External
enableMaxEat	bool newValue	External
increaseAllowance	address spender, uint256 addedValue	Public
manageExcludeFromCut	address[] calldata addresses, bool status	Public
manage_CantEat	address[] calldata addresses, bool status	Public
multiTransfer_fixed	address[] calldata addresses, uint256 amount	External
no_openDoor	None	Public
setDefi	uint256 value	Public
setDistributionSettings	uint256 newLiquidityShare, uint256 newMarketingShare, uint256 newTeamShare	External
setMarketPairStatus	address account, bool newValue	Public
setMaxOnceEat	uint256 newMaxOnceEat	External
setMaxTotalEat	uint256 newMaxTotalEat	External
setNumTokensBeforeswap	uint256 newValue	External
setSwapAndLiquifyBySmallOnly	bool newValue	Public
setSwapAndLiquifyEnabled	bool _enabled	Public
setWhoCantEat	address recipient, bool status	Public

Function Name	Parameters	Visibility
setdoYouLikeBase	address newAddress	External
setinTheMTFFace	address newAddress	External
setisExcludedFromCut	address account, bool newValue	Public
setisMaxEatExempt	address holder, bool exempt	External
setisOnceEatExempt	address holder, bool exempt	External
to_openDoor	uint256 a	Public
transfer	address recipient, uint256 amount	External
transferFrom	address sender,address recipient,uint256 amount	Public
transferOwnership	address newOwner	Public
waiveOwnership	None	Public

Contract Ownership

The contract ownership of SAS Coin is not currently being renounced. The ownership of the contract grants special powers to the protocol creators, making them the sole addresses that can call sensible ownable functions that may alter the state of the protocol.

The current owner is the address 0xA5caF0a379E04Ceb454a2615b05937E85fdA375B which can be viewed: HERE

The owner wallet has the power to call the functions displayed on the privileged functions list above, if the owner wallet is compromised these privileges could be exploited.

We recommend the team renounce ownership at the right timing if possible, or gradually migrate to a time lock with governing functionalities in respect of transparency and safety considerations.

Liquidity Overview

Liquidity Information

Parameter	Result
Pair Address	0x22a4c7d340ceab298135e72ab8771f9e2b8ff054
SAS Reserves	0.00 SAS
BNB Reserves	0.00 BNB
Liquidity Value	$0.00 USDT
Liquidity Ownership	The token does not have liquidity at the moment of the audit

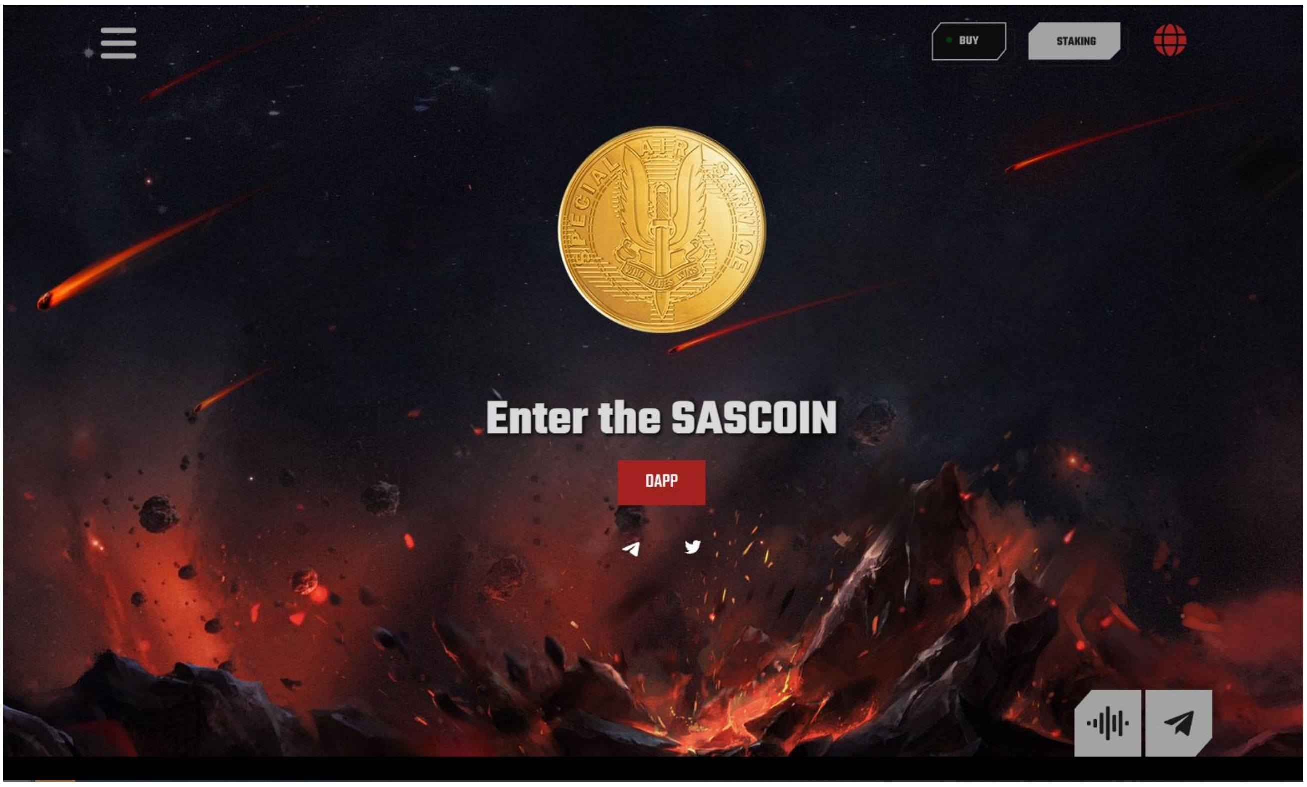
Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x522ce36b924a8c8b56a1b5ba6197c48a65474f70	9,118,800	91.1880%
2	Pinksale: PinkLock V2	881,200	8.8120%

Social Media Check

Social Media Type	Link	Result
Website	http://www.opcdao.org	Checked
Twitter	https://twitter.com/opcdaoofficial/	Checked
Telegram	https://t.me/OPCDaoEnglish/	Checked

Website Review



- Mobile Friendly (A few errors for a wider screen)
- Contains no code errors
- SSL is not secured
- No spelling errors

Audit Conclusion

- The owner cannot pause trading.
- The owner cannot mint new tokens.
- **The owner can add blacklist users.**
- **The owner can set the max transaction amount without limit.**
- **The owner can change the buy/sell fee up to 100%, which includes liquidity fee, marketing fee, and team fee.**

  (All functions cannot be used if the ownership is renounced)