CONTRACT
WOLF

SUMMARY

LANGUAGE	**CHAIN**	**METHODOLOGY**
Solidity	Ethereum	Manual & Automatic Analysis
FILES	**DELIVERY**	**TYPE**
Single	04/11/2023	Standard Audit

4	0	0	0	1	3
Total Findings	Critical	Major	Medium	Minor	Informational

🟥 0 Critical	0 Pending		An exposure that can affect the contract functions in several events that can risk and disrupt the contract
🟧 0 Major	0 Pending		An exposure that can affect the outcome when using the contract that can serve as an opening in manipulating the contract in an unwanted manner
🟧 0 Medium	0 Pending		An opening that could affect the outcome in executing the contract in a specific situation
⬜ 1 Minor	1 Pending		An opening but doesn't have an impact on the functionality of the contract
🟦 3 Informational	3 Pending		An opening that consists information but will not risk or affect the contract

STATUS   ✔️**AUDIT PASSED**

TABLE OF CONTENTS | BUSD

Summary

Project Information

Findings

DISCLAIMER | BUSD

ContractWolf audits and reports should not be considered as a form of project's "Advertisement" and does not cover any interaction and assessment from "Project Contract" to "External Contracts" such as PancakeSwap, UniSwap, SushiSwap or similar.

ContractWolf does not provide any warranty on its released report and should not be used as a decision to invest into audited projects.

ContractWolf provides a transparent report to all its "Clients" and to its "Clients Participants" and will not claim any guarantee of bug-free code within its **SMART CONTRACT**.

ContractWolf's presence is to analyze, audit and assess the Client's Smart Contract to find any underlying risk and to eliminate any logic and flow errors within its code.

Each company or project should be liable to its security flaws and functionalities.

SCOPE OF WORK | BUSD

BUSD team has agreed and provided us with the files that need to be tested (*Github, BSCscan, Etherscan, Local files etc*). The scope of audit is the main contract.

The goal of this engagement is to identify if there is a possibility of security flaws in the implementation of smart contract and its systems.

ContractWolf will be focusing on contract issues and functionalities along with the project claims from smart contract to their website, whitepaper, repository which has been provided by **BUSD**.
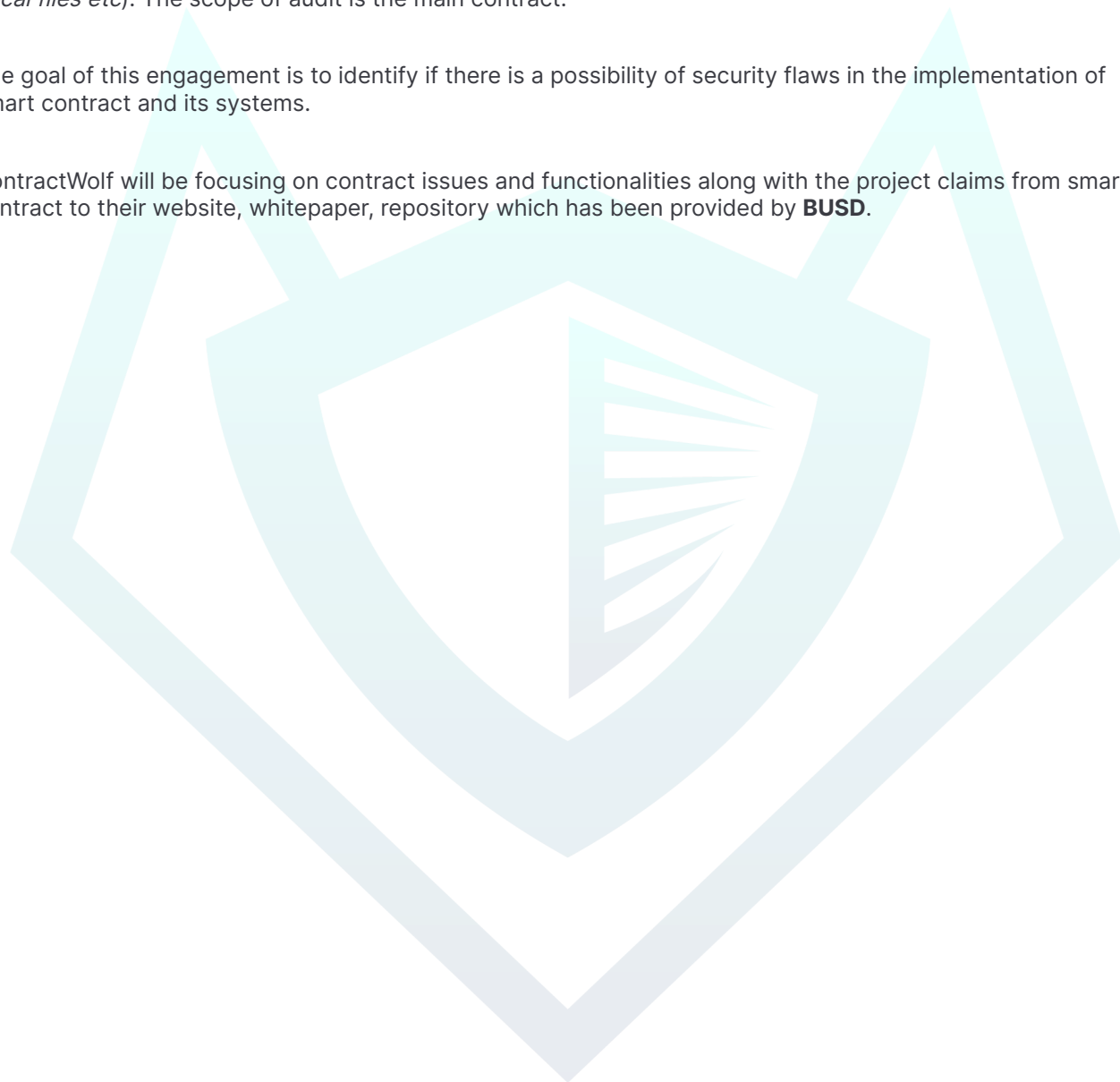
AUDITING APPROACH | BUSD

Every line of code along with its functionalities will undergo manual review to check for security issues, quality of logic and contract scope of inheritance. The manual review will be done by our team that will document any issues that they discovered.

METHODOLOGY

The auditing process follows a routine series of steps :

1. Code review that includes the following :
   - Review of the specifications, sources and instructions provided to ContractWolf to make sure we understand the size, scope and functionality of the smart contract.
   - Manual review of code. Our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities, underlying and hidden security flaws.

2. Testing and automated analysis that includes :
   - Testing the smart contract function with common test cases and scenarios to ensure that it returns the expected results.

3. Best practices review. The team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security and control within the smart contract.

4. Recommendations to help the project take steps to eliminate or minimize threats and secure the smart contract.

TOKEN DETAILS | BUSD

This is an audit sample and reference of ContractWolf and should not be used for any project.

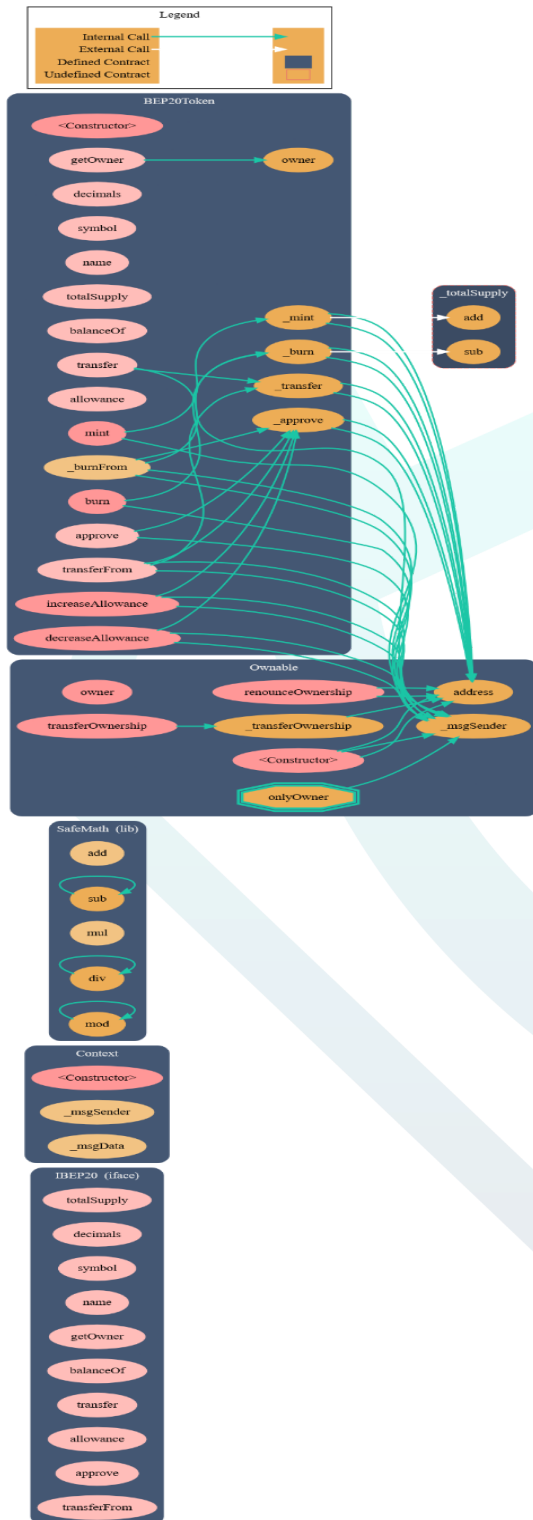| Token Name	Symbol	Decimal	Total Supply	Chain
Binance-USD	BUSD	18	–	Binance Smart Chain

SOURCE

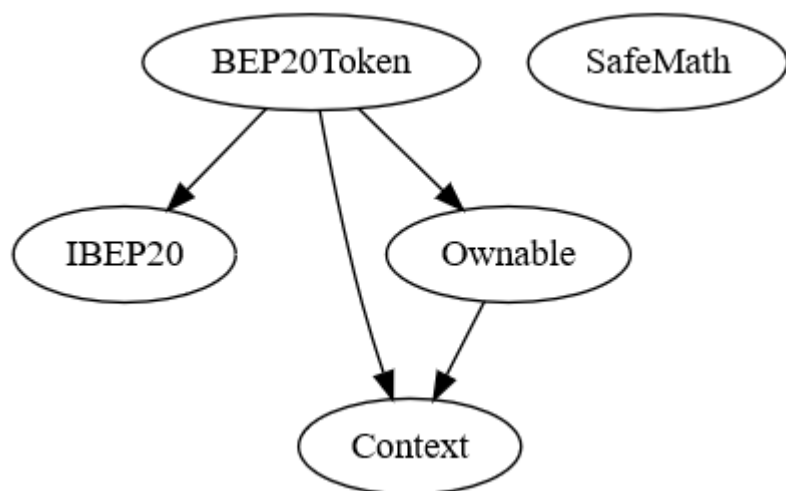Source   *0x55d398326f99059ff775485246999027b3197955*

INHERITANCE GRAPH | BUSD

Inheritance Graph of Smart Contract Functions

CALL GRAPH | BUSD

Inheritance Graph of Smart Contract Functions

FINDINGS | BUSD

4	0	0	0	1	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Matic Through this audit we have uncovered issues ranging from different severity levels Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews we discovered the following findings

ID	Title	File & Line #	Severity	Status
SWC-119	Shadowing State Variables	BUSD.sol, L: 421, 584	Minor	● Pending
SWC-135	Code With No Effects	BUSD.sol, L : 598, 115-119	Informational	● Pending
SWC-103	A Floating Pragma is Set	BUSD.sol, L : 3	Informational	● Pending
CW-012	SafeMath Override	BUSD.sol, L : 134	Informational	● Pending

SWC ATTACKS | BUSD

Smart Contract Weakness Classification and Test Cases

ID	Description	Status
SWC-100	Function Default Visibility	● Passed
SWC-101	Integer Overflow and Underflow	● Passed
SWC-102	Outdated Compiler Version	● Passed
SWC-103	Floating Pragma	● Not Passed
SWC-104	Unchecked Call Return Value	● Passed
SWC-105	Unprotected Ether Withdrawal	● Passed
SWC-106	Unprotected SELF DESTRUCT Instruction	● Passed
SWC-107	Reentrancy	● Passed
SWC-108	State Variable Default Visibility	● Passed
SWC-109	Uninitialized Storage Pointer	● Passed
SWC-110	Assert Violation	● Passed
SWC-111	Use of Deprecated Solidity Functions	● Passed
SWC-112	Delegatecall to Untrusted Callee	● Passed
SWC-113	DoS with Failed Call	● Passed
SWC-114	Transaction Order Dependence	● Passed
SWC-115	Authorization through tx.origin	● Passed
SWC-116	Block values as a proxy for time	● Passed
SWC-117	Signature Malleability	● Passed
SWC-118	Incorrect Constructor Name	● Passed
SWC-119	Shadowing State Variables	● Not Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	● Passed
SWC-121	Missing Protection against Signature Replay Attacks	● Passed
SWC-122	Lack of Proper Signature Verification	● Passed

ID	Description	Status
SWC-123	Requirement Violation	● Passed
SWC-124	Write to Arbitrary Storage Location	● Passed
SWC-125	Incorrect Inheritance Order	● Passed
SWC-126	Insufficient Gas Griefing	● Passed
SWC-127	Arbitrary Jump with Function Type Variable	● Passed
SWC-128	DoS With Block Gas Limit	● Passed
SWC-129	Typographical Error	● Passed
SWC-130	Right-To-Left-Override control character (U+202E)	● Passed
SWC-131	Presence of unused variables	● Passed
SWC-132	Unexpected Ether balance	● Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	● Passed
SWC-134	Message call with hardcoded gas amount	● Passed
SWC-135	Code With No Effects	● Not Passed
SWC-136	Unencrypted Private Data On-Chain	● Passed

CW ASSESSMENT | BUSD

ContractWolf Vulnerability and Security Tests

ID	Name	Description	Status
CW-001	Multiple Version	Presence of multiple compiler version across all contracts	● Passed
CW-002	Incorrect Access Control	Additional checks for critical logic and flow	● Passed
CW-003	Payable Contract	A function to withdraw ether should exist otherwise the ether will be trapped	● Passed
CW-004	Custom Modifier	major recheck for custom modifier logic	● Passed
CW-005	Divide Before Multiply	Performing multiplication before division is generally better to avoid loss of precision	● Passed
CW-006	Multiple Calls	Functions with multiple internal calls	● Passed
CW-007	Deprecated Keywords	Use of deprecated functions/operators such as block.blockhash() for blockhash(), msg.gas for gasleft(), throw for revert(), sha3() for keccak256(), callcode() for delegatecall(), suicide() for selfdestruct(), constant for view or var for actual type name should be avoided to prevent unintended errors with newer compiler versions	● Passed
CW-008	Unused Contract	Presence of an unused, unimported or uncalled contract	● Passed
CW-009	Assembly Usage	Use of EVM assembly is error-prone and should be avoided or double-checked for correctness	● Passed
CW-010	Similar Variable Names	Variables with similar names could be confused for each other and therefore should be avoided	● Passed
CW-011	Commented Code	Removal of commented/unused code lines	● Passed
CW-012	SafeMath Override	SafeMath is no longer needed starting Solidity v0.8+. The compiler now has Built in overflow checking.	● Not Passed

FIXES & RECOMMENDATION

SWC-119 | Shadowing State Variable

Owner from public functions shadows the Owner from Contract Ownable

```
 contract Ownable is Context {
address private _owner;
```

And to

```
function _approve(address owner, address spender, uint256 amount) internal {
```

```
 function allowance(address owner, address spender) external view returns (uint256)
```

Recommendation

Review storage variable layouts for your contract systems carefully and remove any ambiguities. Always check for compiler warnings as they can flag the issue within a single contract.

SWC-135 | Code With No Effects

Function

```
function _burnFrom(address account, uint256 amount) internal {

    _burn(account, amount);

    _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount,
"BEP20: burn amount exceeds allowance"));

  }

}
```

And

```
function _msgData() internal view returns (bytes memory) {

    this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691

    return msg.data;

  }
```

Recommandation

The codes and lines above were never used and should be removed to ensure correct behavior.

SWC-103 | A Floating Pragma is Set

Code

```
// SPDX-License-Identifier: UNLICENSED


pragma solidity ^0.8.17;
```

The compiler version should be a fixed one to avoid undiscovered compiler bugs. Fixed version sample below

```
 // SPDX-License-Identifier: UNLICENSED


pragma solidity 0.8.17;
```

CW-012 | SafeMath Override

```
library SafeMath
```

SafeMath is no longer needed starting Solidity v0.8+. The compiler now has Built in overflow checking.

AUDIT COMMENTS | BUSD

Smart Contract audit comment for a non-technical perspective

- 🟢 Owner cannot mint after initial deployment
- 🟢 Owner cannot burn tokens
- 🟢 Owner cannot pause the Contract
- 🟢 Owner cannot set max transaction
- 🟢 Contract does not have taxes/fees
- 🟢 Owner cannot block the users
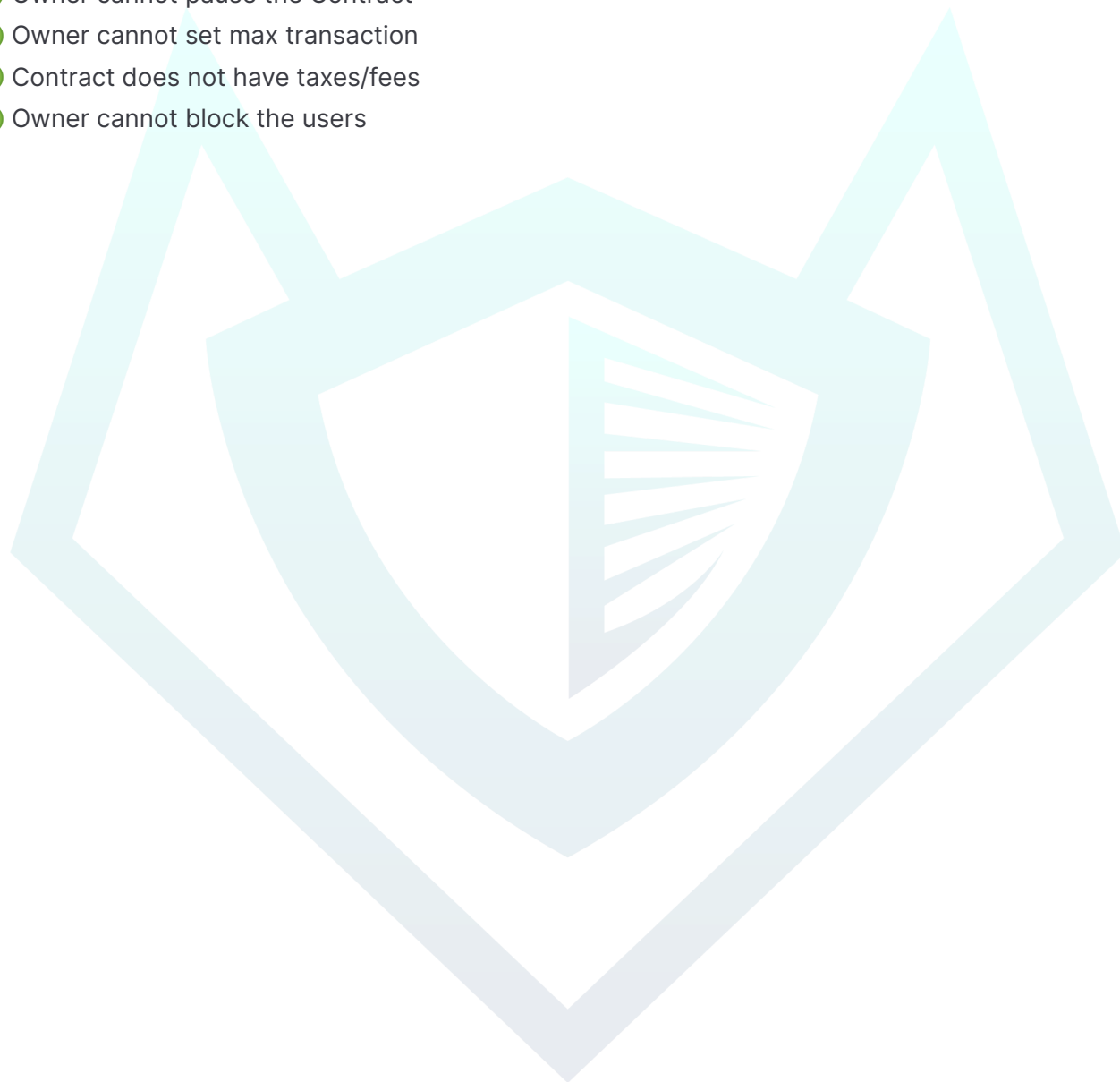
CONTRACTWOLF

Blockchain Security - Smart Contract Audits