www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	USDBK777 Token
Platform:	Polygon
Website:	https://backters.com
Language:	Solidity
Date:	April 8th, 2022

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the backters team to perform the Security audit of the USDBK777 Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 8th, 2022.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

USDBK777 Contract is a smart contract, having functions like operatorBatchTransfer, operatorBatchMint, isOperatorFor, operatorBatchBurn, mint, send, transfer, burn, destroy, addDefaultOperator, etc. The USDBK777 contract inherits the IERC20, ERC777, SafeMath standard smart contracts from the OpenZeppelin library. These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for USDBK777 Token Smart Contract**
Platform	**Polygon / Solidity**
File	USDBK777Token.sol
File MD5 Hash	EADAFF7DA4DCB2C1E6F7CEEBAC46CF3A
Revised Code MD5 Hash	A7EF57C65C556A43D70FFD7083917308
Audit Date	April 8th, 2022
Revision Date	May 27th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Token Name: USDBK • Token Symbol: USDBK • Decimals: 18 • Max Minting Limit: No limits • ERC777 compliance • ERC20 backward compatible	**YES, This is valid.**
Ownership Control: • Owner can mint unlimited tokens • Owner/Authorized person can add a default operator • Owner can destroy the smart contract • Owner can burn someone else's tokens • Owner can freeze/unfreeze the token transfer	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. This token contract does not contain owner control, which does make it fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 2 high, 0 medium and 1 low and some very low level issues. These issues are acknowledged by the dev team.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Not Set
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract file. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in USDBK777 Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the USDBK777 Token.

The USDBK777 Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **well** commented on in the smart contracts. Ethereum's NatSpec commenting style is used which is a good thing.

Documentation

We were given a USDBK777 Token smart contract code in the form of a file.The hash of that code is mentioned above in the table.

As mentioned above, code parts are well commented. And the contract is straightforward so it's easy to understand its programming logic.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	destroy	write	Owner can destroy entire smart contract	No Issue
3	addDefaultOperator	write	Passed	No Issue
4	isOperatorFor	read	Passed	No Issue
5	getTransferEnabled	read	Passed	No Issue
6	setTransferEnabled	write	Passed	No Issue
7	send	write	Passed	No Issue
8	transfer	write	Passed	No Issue
9	burn	write	Passed	No Issue
10	transferFrom	write	Passed	No Issue
11	batchBalanceOf	read	Passed	No Issue
12	operatorBatchTransfer	write	Passed	No Issue
13	operatorBatchMint	write	No max minting limits	Acknowledged by dev team
14	operatorBatchBurn	write	Owner can burn anyone's tokens	Acknowledged by dev team
15	operatorMint	write	No max minting limits	Acknowledged by dev team
16	operatorBurn	write	Owner can burn anyone's tokens	Acknowledged by dev team
17	operatorTransferAnyERC20Token	write	Passed	No Issue
18	name	read	Passed	No Issue
19	symbol	read	Passed	No Issue
20	decimals	write	Passed	No Issue
21	granularity	read	Passed	No Issue
22	totalSupply	read	Passed	No Issue
23	balanceOf	read	Passed	No Issue
24	send	write	Passed	No Issue
25	transfer	write	Passed	No Issue
26	burn	write	Passed	No Issue
27	isOperatorFor	read	Passed	No Issue
28	authorizeOperator	write	Passed	No Issue
29	revokeOperator	write	Passed	No Issue
30	defaultOperators	read	Passed	No Issue
31	operatorSend	write	Passed	No Issue
32	operatorBurn	write	Passed	No Issue
33	allowance	read	Passed	No Issue
34	approve	write	Passed	No Issue
35	transferFrom	write	Passed	No Issue
36	_mint	internal	Passed	No Issue
37	_send	internal	Passed	No Issue

38	_burn	internal	Passed	No Issue
39	_move	write	Passed	No Issue
40	_approve	internal	Passed	No Issue
41	_callTokensToSend	write	Passed	No Issue
42	_callTokensReceived	write	Passed	No Issue
43	_spendAllowance	internal	Passed	No Issue
44	_beforeTokenTransfer	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) Owner (or an authorized operator) can mint unlimited tokens.

```
function operatorMint(
    address account,
    uint256 amount,
    bytes memory data,
    bytes memory operatorData
) public virtual override {
    require(isOperatorFor(_msgSender(), account), "USDBK777: caller is not an operator
    _mint(account, amount, data, operatorData);
```

Minting unlimited tokens creates unhealthy tokenomics. It gives the owner an option to generate more and more tokens, which may inflate the token value.

Resolution: We suggest setting a maximum limit for the token mint.

Status: This issue is acknowledged by the dev team.

(2) Owner (or an authorized operator) can burn anyone's tokens

```
function operatorBurn(
    address account,
    uint256 amount,
    bytes memory data,
    bytes memory operatorData
) public virtual override(ERC777) {
    require(isOperatorFor(_msgSender(), account), "USDBK777: caller is not an operator
    _burn(account, amount, data, operatorData);
```

If the owner can burn anyone's tokens, then it removes the decentralization. Token holders can FUD that someone (owner) can burn their tokens.

Resolution: We suggest removing this or making token holders burn their own tokens.

Status: This issue is acknowledged by the dev team.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Infinite loop possibility

```
function operatorBatchTransfer(
    address sender,
    address[] memory recipients,
    uint256[] memory amounts,
    bytes memory data,
    bytes memory operatorData
) public virtual override {
    require(isOperatorFor(_msgSender(), sender), "USDBK777: caller is not an operator for
    require(recipients.length == amounts.length, "USDBK777: recipients and amounts length

    for (uint256 i = 0; i < recipients.length; i++) {
        address recipient = recipients[i];
        uint256 amount = amounts[i];

        _send(sender, recipient, amount, data, operatorData, false);
    }
```

There are 3 functions where owners can input unlimited wallets, such as: batchBalanceOf, operatorBatchTransfer, operatorBatchMint, operatorBatchBurn.

If there are lots of wallets used to execute those functions, then it might hit the block's gas limit and may fail.

Resolution: We suggest setting a limit for the number of wallets can be used, or the owner can acknowledge to use limited wallets only.

Status: This issue is acknowledged by the dev team.

Very Low / Informational / Best practices:

(1) No need for empty/default value assignment

```
_transferEnabled = false;
```

All the boolean variables have default as “false”. So, no need to explicitly assign the value. Although this does not raise any security or logical vulnerability, it is a good practice to avoid setting empty/default values explicitly.

Status: This issue is acknowledged by the dev team.

(2) No need to use SafeMath library for solidity version over 0.8.0

```
using SafeMath for uint256;
```

The solidity version over 0.8.0 has an in-built overflow/underflow prevention mechanism. And thus, the explicit use of SathMath library is not necessary. It saves some gas as well.

Status: This issue is acknowledged by the dev team.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We have observed some issues, and they are fixed / acknowledged by the dev team. So, **the smart contract can go for the mainnet deployment**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured"**.

Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

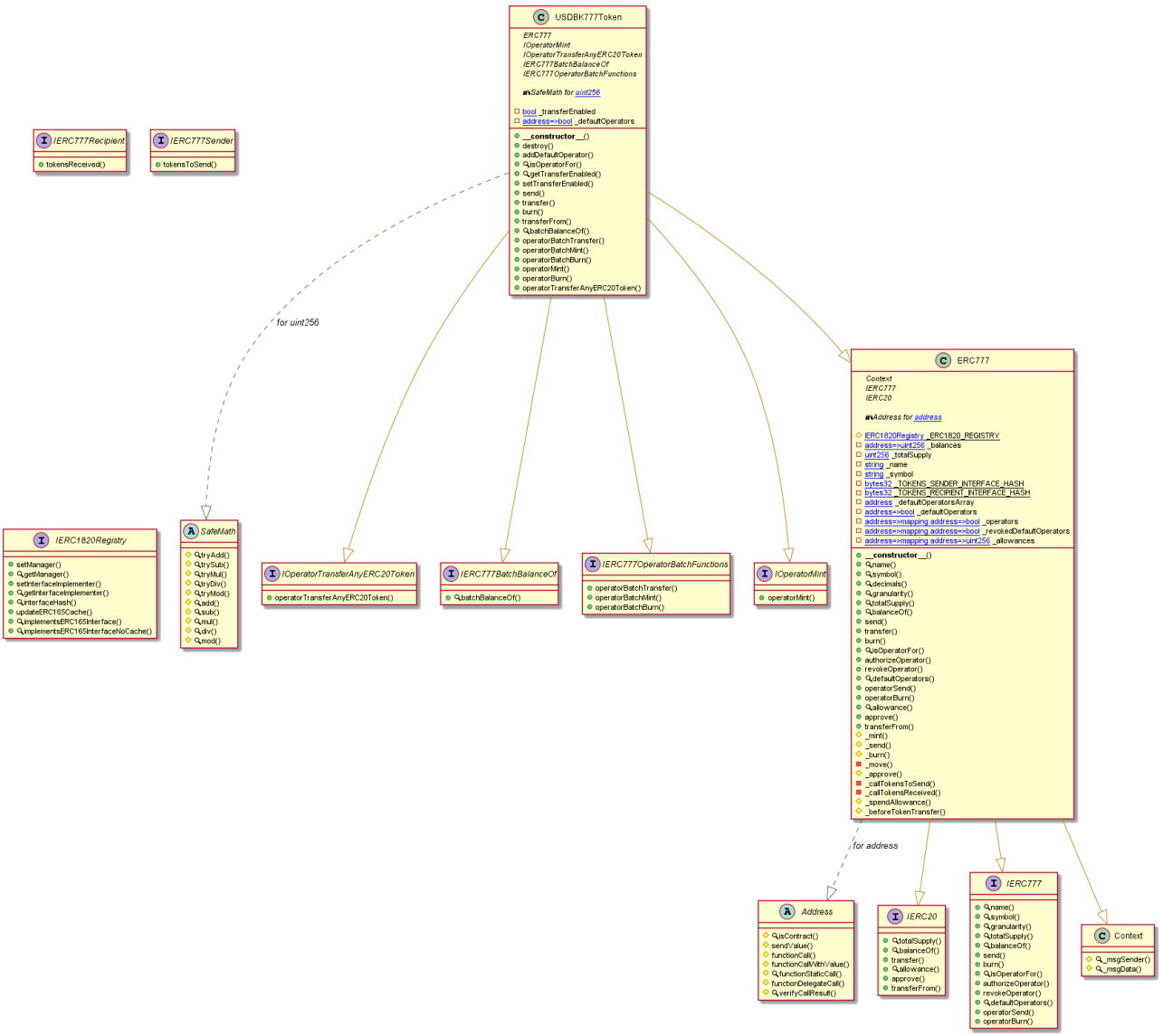
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - USDBK777 Token



Slither Results Log

Slither log >> USDBK777Token.sol

```
INFO:Detectors:
USDBK777Token.constructor(string,string,address[],uint256).name (USDBK777Token.sol#1426) shadows:
        - ERC777.name() (USDBK777Token.sol#945-947) (function)
        - IERC777.name() (USDBK777Token.sol#535) (function)
USDBK777Token.constructor(string,string,address[],uint256).symbol (USDBK777Token.sol#1427) shadows:
        - ERC777.symbol() (USDBK777Token.sol#952-954) (function)
        - IERC777.symbol() (USDBK777Token.sol#541) (function)
USDBK777Token.constructor(string,string,address[],uint256).defaultOperators (USDBK777Token.sol#1428) shadows:
        - ERC777.defaultOperators() (USDBK777Token.sol#1067-1069) (function)
        - IERC777.defaultOperators() (USDBK777Token.sol#642) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in ERC777._burn(address,uint256,bytes,bytes) (USDBK777Token.sol#1253-1277):
        External calls:
        - _callTokensToSend(operator,from,address(0),amount,data,operatorData) (USDBK777Token.sol#1263)
                - IERC777Sender(implementer).tokensToSend(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#
1336)
        State variables written after the call(s):
        - _balances[from] = fromBalance - amount (USDBK777Token.sol#1271)
        - _totalSupply -= amount (USDBK777Token.sol#1273)
Reentrancy in ERC777._send(address,address,uint256,bytes,bytes,bool) (USDBK777Token.sol#1226-1244):
        External calls:
        - _callTokensToSend(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#1239)
                - IERC777Sender(implementer).tokensToSend(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#
1336)
        State variables written after the call(s):
        - _move(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#1241)
                - _balances[from] = fromBalance - amount (USDBK777Token.sol#1292)
                - _balances[to] += amount (USDBK777Token.sol#1294)
Reentrancy in USDBK777Token.constructor(string,string,address[],uint256) (USDBK777Token.sol#1425-1438):
        External calls:
        - ERC777(name,symbol,defaultOperators) (USDBK777Token.sol#1431)
                - _ERC1820_REGISTRY.setInterfaceImplementer(address(this),keccak256(bytes)(ERC777Token),address(this)) (USDB
K777Token.sol#938)
                - _ERC1820_REGISTRY.setInterfaceImplementer(address(this),keccak256(bytes)(ERC20Token),address(this)) (USDBK
777Token.sol#939)
        State variables written after the call(s):
```

```
        - _transferEnabled = false (USDBK777Token.sol#1433)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in ERC777._burn(address,uint256,bytes,bytes) (USDBK777Token.sol#1253-1277):
        External calls:
        - _callTokensToSend(operator,from,address(0),amount,data,operatorData) (USDBK777Token.sol#1263)
                - IERC777Sender(implementer).tokensToSend(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#
1336)
        Event emitted after the call(s):
        - Burned(operator,from,amount,data,operatorData) (USDBK777Token.sol#1275)
        - Transfer(from,address(0),amount) (USDBK777Token.sol#1276)
Reentrancy in ERC777._mint(address,uint256,bytes,bytes,bool) (USDBK777Token.sol#1194-1215):
        External calls:
        - _callTokensReceived(operator,address(0),account,amount,userData,operatorData,requireReceptionAck) (USDBK777Token.s
ol#1211)
                - IERC777Recipient(implementer).tokensReceived(operator,from,to,amount,userData,operatorData) (USDBK777Token
.sol#1362)
        Event emitted after the call(s):
        - Minted(operator,account,amount,userData,operatorData) (USDBK777Token.sol#1213)
        - Transfer(address(0),account,amount) (USDBK777Token.sol#1214)
Reentrancy in ERC777._send(address,address,uint256,bytes,bytes,bool) (USDBK777Token.sol#1226-1244):
        External calls:
        - _callTokensToSend(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#1239)
                - IERC777Sender(implementer).tokensToSend(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#
1336)
        Event emitted after the call(s):
        - Sent(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#1296)
                - _move(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#1241)
        - Transfer(from,to,amount) (USDBK777Token.sol#1297)
                - _move(operator,from,to,amount,userData,operatorData) (USDBK777Token.sol#1241)
Reentrancy in USDBK777Token.constructor(string,string,address[],uint256) (USDBK777Token.sol#1425-1438):
        External calls:
        - _mint(msg.sender,initialSupply,,) (USDBK777Token.sol#1436)
                - IERC777Recipient(implementer).tokensReceived(operator,from,to,amount,userData,operatorData) (USDBK777Token
.sol#1362)
        - ERC777(name,symbol,defaultOperators) (USDBK777Token.sol#1431)
                - _ERC1820_REGISTRY.setInterfaceImplementer(address(this),keccak256(bytes)(ERC777Token),address(this)) (USDB
```

```
K777Token.sol#938)
                - _ERC1820_REGISTRY.setInterfaceImplementer(address(this),keccak256(bytes)(ERC20Token),address(this)) (USDBK
777Token.sol#939)
        Event emitted after the call(s):
        - Minted(operator,account,amount,userData,operatorData) (USDBK777Token.sol#1213)
                - _mint(msg.sender,initialSupply,,) (USDBK777Token.sol#1436)
        - Transfer(address(0),account,amount) (USDBK777Token.sol#1214)
                - _mint(msg.sender,initialSupply,,) (USDBK777Token.sol#1436)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (USDBK777Token.sol#197-217) uses assembly
        - INLINE ASM (USDBK777Token.sol#209-212)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (USDBK777Token.sol#81-83) is never used and should be removed
Address.functionCall(address,bytes,string) (USDBK777Token.sol#91-97) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (USDBK777Token.sol#110-116) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (USDBK777Token.sol#124-135) is never used and should be removed
Address.functionDelegateCall(address,bytes) (USDBK777Token.sol#170-172) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (USDBK777Token.sol#180-189) is never used and should be removed
Address.functionStaticCall(address,bytes) (USDBK777Token.sol#143-145) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (USDBK777Token.sol#153-162) is never used and should be removed
Address.sendValue(address,uint256) (USDBK777Token.sol#56-61) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (USDBK777Token.sol#197-217) is never used and should be removed
Context._msgData() (USDBK777Token.sol#887-889) is never used and should be removed
SafeMath.add(uint256,uint256) (USDBK777Token.sol#297-299) is never used and should be removed
```

```
SafeMath.mod(uint256,uint256,string) (USDBK777Token.sol#421-430) is never used and should be removed
SafeMath.mul(uint256,uint256) (USDBK777Token.sol#325-327) is never used and should be removed
SafeMath.sub(uint256,uint256) (USDBK777Token.sol#311-313) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (USDBK777Token.sol#372-381) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (USDBK777Token.sol#226-232) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (USDBK777Token.sol#268-273) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (USDBK777Token.sol#280-285) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (USDBK777Token.sol#251-261) is never used and should be removed
SafeMath.trySub(uint256,uint256) (USDBK777Token.sol#239-244) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.8.2<0.9.0 (USDBK777Token.sol#2) is too complex
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (USDBK777Token.sol#56-61):
        - (success) = recipient.call{value: amount}() (USDBK777Token.sol#59)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (USDBK777Token.sol#124-135):
        - (success,returndata) = target.call{value: value}(data) (USDBK777Token.sol#133)
Low level call in Address.functionStaticCall(address,bytes,string) (USDBK777Token.sol#153-162):
        - (success,returndata) = target.staticcall(data) (USDBK777Token.sol#160)
Low level call in Address.functionDelegateCall(address,bytes,string) (USDBK777Token.sol#180-189):
        - (success,returndata) = target.delegatecall(data) (USDBK777Token.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Variable ERC777._defaultOperators (USDBK777Token.sol#912) is too similar to ERC777.constructor(string,string,address[]).defa
ultOperators_ (USDBK777Token.sol#927)
Variable USDBK777Token._defaultOperators (USDBK777Token.sol#1423) is too similar to ERC777.constructor(string,string,address
[]).defaultOperators_ (USDBK777Token.sol#927)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
name() should be declared external:
        - ERC777.name() (USDBK777Token.sol#945-947)
symbol() should be declared external:
        - ERC777.symbol() (USDBK777Token.sol#952-954)
decimals() should be declared external:
        - ERC777.decimals() (USDBK777Token.sol#962-964)
```

```
granularity() should be declared external:
        - ERC777.granularity() (USDBK777Token.sol#971-973)
totalSupply() should be declared external:
        - ERC777.totalSupply() (USDBK777Token.sol#978-980)
authorizeOperator(address) should be declared external:
        - ERC777.authorizeOperator(address) (USDBK777Token.sol#1037-1047)
revokeOperator(address) should be declared external:
        - ERC777.revokeOperator(address) (USDBK777Token.sol#1052-1062)
defaultOperators() should be declared external:
        - ERC777.defaultOperators() (USDBK777Token.sol#1067-1069)
operatorSend(address,address,uint256,bytes,bytes) should be declared external:
        - ERC777.operatorSend(address,address,uint256,bytes,bytes) (USDBK777Token.sol#1076-1085)
operatorBurn(address,uint256,bytes,bytes) should be declared external:
        - ERC777.operatorBurn(address,uint256,bytes,bytes) (USDBK777Token.sol#1092-1100)
        - USDBK777Token.operatorBurn(address,uint256,bytes,bytes) (USDBK777Token.sol#1643-1651)
approve(address,uint256) should be declared external:
        - ERC777.approve(address,uint256) (USDBK777Token.sol#1121-1125)
destroy() should be declared external:
        - USDBK777Token.destroy() (USDBK777Token.sol#1440-1443)
addDefaultOperator(address) should be declared external:
        - USDBK777Token.addDefaultOperator(address) (USDBK777Token.sol#1456-1465)
getTransferEnabled() should be declared external:
        - USDBK777Token.getTransferEnabled() (USDBK777Token.sol#1480-1482)
setTransferEnabled(bool) should be declared external:
        - USDBK777Token.setTransferEnabled(bool) (USDBK777Token.sol#1484-1489)
batchBalanceOf(address[]) should be declared external:
        - USDBK777Token.batchBalanceOf(address[]) (USDBK777Token.sol#1549-1563)
operatorBatchTransfer(address,address[],uint256[],bytes,bytes) should be declared external:
        - USDBK777Token.operatorBatchTransfer(address,address[],uint256[],bytes,bytes) (USDBK777Token.sol#1565-1581)
operatorBatchMint(address[],uint256[],bytes,bytes) should be declared external:
        - USDBK777Token.operatorBatchMint(address[],uint256[],bytes,bytes) (USDBK777Token.sol#1583-1602)
operatorBatchBurn(address[],uint256[],bytes,bytes) should be declared external:
        - USDBK777Token.operatorBatchBurn(address[],uint256[],bytes,bytes) (USDBK777Token.sol#1604-1623)
operatorMint(address,uint256,bytes,bytes) should be declared external:
        - USDBK777Token.operatorMint(address,uint256,bytes,bytes) (USDBK777Token.sol#1628-1636)
operatorTransferAnyERC20Token(address,address,uint256) should be declared external:
        - USDBK777Token.operatorTransferAnyERC20Token(address,address,uint256) (USDBK777Token.sol#1656-1663)
```

```
operatorMint(address,uint256,bytes,bytes) should be declared external:
        - USDBK777Token.operatorMint(address,uint256,bytes,bytes) (USDBK777Token.sol#1628-1636)
operatorTransferAnyERC20Token(address,address,uint256) should be declared external:
        - USDBK777Token.operatorTransferAnyERC20Token(address,address,uint256) (USDBK777Token.sol#1656-1663)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:USDBK777Token.sol analyzed (14 contracts with 75 detectors), 66 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

USDBK777Token.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability.

more

Pos: 124:4:

Selfdestruct:

Use of selfdestruct: Can block calling contracts unexpectedly. Be especially careful if this contract is planned to be used by other contracts (i.e. library contracts, interactions). Selfdestruction of the callee contract can leave callers in an inoperable state.

more

Pos: 1442:8:

Gas & Economy

Gas costs:

Gas requirement of function ERC777.name is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 945:4:

Gas costs:

Gas requirement of function ERC777.send is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1496:4:

Gas costs:

Gas requirement of function USDBK777Token.send is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1496:4:

Gas costs:

Gas requirement of function ERC777.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1523:4:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function USDBK777Token.operatorBatchMint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1583:4:

Gas costs:

Gas requirement of function USDBK777Token.operatorBatchBurn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1604:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 1558:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 1575:8:

Miscellaneous

Constant/View/Pure functions:

Address.functionStaticCall(address,bytes) : Is constant but potentially should not be.
more
Pos: 143:4:

Constant/View/Pure functions:

USDBK777Token.burn(uint256,bytes) : Potentially should be constant/view/pure but is not.
more
Pos: 1523:4:

Constant/View/Pure functions:

USDBK777Token.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not.
more
Pos: 1540:4:

Similar variable names:

USDBK777Token.operatorBatchTransfer(address,address[],uint256[],bytes,bytes) : Variables have very similar names "recipient" and "recipients".
Pos: 1576:32:

Similar variable names:

USDBK777Token.operatorBatchTransfer(address,address[],uint256[],bytes,bytes) : Variables have very similar names "recipient" and "recipients".
Pos: 1579:26:

Similar variable names:

USDBK777Token.operatorBatchTransfer(address,address[],uint256[],bytes,bytes) : Variables have very similar names "amount" and "amounts".
Pos: 1573:37:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1572:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1573:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1610:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1613:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 402:19:

Solhint Linter

USDBK777Token.sol

```
USDBK777Token.sol:227:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:240:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:252:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:269:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:281:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:377:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:400:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:426:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:1270:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:1291:18: Error: Parse error: missing ';' at '{'
USDBK777Token.sol:1384:22: Error: Parse error: missing ';' at '{'
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io