



SOLIDProof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

USDFI / STABLE / StableMinter

AUDIT

SECURITY ASSESSMENT

28. November, 2023

FOR



SolidProof_io



@solidproof_io

Introduction	4
Disclaimer	4
Project Overview	5
Summary	5
Social Medias	5
Audit Summary	6
File Overview	7
Imported packages	8
Components	9
Exposed Functions	9
Capabilities	10
Inheritance Graph	11
Audit Information	12
Vulnerability & Risk Level	12
Auditing Strategy and Techniques Applied	13
Methodology	13
Overall Security	14
Upgradeability	14
Ownership	15
Ownership Privileges	16
Minting tokens	16
Burning tokens	20
Blacklist addresses	21
Fees and Tax	22
Lock User Funds	23
Centralization Privileges	24
Audit Results	26
Critical issues	26
High issues	26
Medium issues	26



Low issues

26

Informational issues

26



Introduction

[SolidProof.io](#) is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.

Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

[SolidProof.io](#) reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	USDFi
Website	https://usdfi.com/
About the project	USDFi is crypto's first and only Universal Banking Protocol and DeFi's multichain native banking layer.
Chain	BSC
Language	Solidity
Codebase Link	
USDFI	Provided as files
STABLE	Provided as files
StableMinter	Provided as files
Commit	2ecce2186ce23916d84ef5d769c40c62383025c6
Unit Tests	Not Provided
Comment	The USDFI and the Stable contracts are the same.

Social Medias

Telegram	https://t.me/USDFI
Twitter	https://twitter.com/stable_usdfi
Facebook	N/A
Instagram	https://www.instagram.com/stable.usdfi/
Github	https://github.com/USDFI
Reddit	N/A
Medium	https://medium.com/@usdfi
Discord	https://discord.gg/MjvpF8UwB4
Youtube	https://www.youtube.com/@usdfi
TikTok	N/A
LinkedIn	N/A



Audit Summary

Version	Delivery Date	Changelog
v1.0	22. November 2023	<ul style="list-style-type: none"> • Layout Project • Automated- /Manual-Security Testing • Summary
	28. November 2023	<ul style="list-style-type: none"> • Reaudit

Note - This Audit report consists of a security analysis of the **USDFI / STABLE / StableMinter** smart contract. This analysis did not include functional testing (or unit testing) of the contract's logic. Stable is the same contract as USDFI.



File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts/StableMinter.sol	2f859bbd3f926c55d46333a81839cb236f8b6fc6
contracts/USDFI.sol	64202ecc08b2c9bcb9e1ae60d62afc8f8ad135d
contracts/STABLE.sol	18ea5f7dcdd20ed3b37464300fafb33f96ce2563

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.



Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@openzeppelin/contracts/access/AccessControl.sol	2
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/security/ReentrancyGuard.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	2
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	2
@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol	2
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way.

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

 Contracts	 Libraries	 Interfaces	 Abstract
3	0	1	0


Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable
12	0

External	Internal	Private	Pure	View
3	15	0	0	2

StateVariables

Total	 Public
16	16



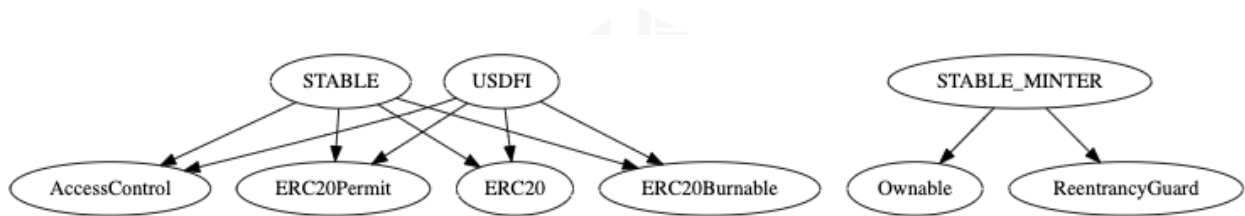
Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
=0.8.19 ^0.8.23	-----	----	----	-----



Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 - 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 - 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 - 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 - 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - a. Reviewing the specifications, sources, and instructions provided to SolidProof to ensure we understand the size, scope, and functionality of the smart contract.
 - b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
 - c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.
2. Testing and automated analysis that includes the following:
 - a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
 - b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.
3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.
4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security Upgradeability

Contract is not an upgradeable



Deployer cannot update the contract with new functionalities

Description

The contract is not an upgradeable contract. The deployer is not able to change or add any functionalities to the contract after deploying.

Comment

N/A



Ownership

The ownership is not renounced

✗ The owner is not renounce

Description

The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including:

- Centralizations
- The owner has significant control over contract's operations

Example

N/A

Comment

N/A

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner can mint new tokens	✗ The owner can mint new tokens
Description	<p>Owners who have the ability to mint new tokens can reward themselves or other stakeholders, who can then sell the newly minted tokens on a cryptocurrency exchange to raise funds. However, there is a risk that the owner may abuse this power, leading to a decrease in trust and credibility in the project or platform. If stakeholders perceive that the owner is using their power to mint new tokens unfairly or without transparency, it can result in decreased demand for the token and a reduction in its value.</p>
Example	<p>If investors drive up the token price, the owner may choose to mint new tokens and sell them on a cryptocurrency exchange to raise funds. If the owner is not transparent and honest about their actions, they may be attempting a rugpull, where they suddenly abandon the project after raising funds, leaving investors with worthless tokens. This can lead to a decrease in the value of existing tokens, potentially rendering them worthless, and causing investors to suffer losses. It is essential for investors to carefully research the project and its developers and exercise caution before investing in any cryptocurrency or DeFi project.</p>

Contract owner can mint new tokens

✗ The owner can mint new tokens

Comment

USDFI/STABLE

Addresses with the "MINTER_ROLE" can mint new tokens. The addresses with "CONTROLLER_ROLE" can add free mint supplies without a limit. Afterwards, the minter role address can mint to any arbitrary address. It is not possible that the minter roled address can mint more than then free mint supply that was added by the controller roled address.

StableMinter

The owner of the StableMinter contract can set receiver addresses and their percentages per address. The percentage cannot be higher than 100%. The percentage must be in sum 100%. Everyone is also able to call the createNewStable function which is minting new tokens (Stable tokens: [Stable token contract](#)) to the set receivers with their percentages when the current timestamp is higher than the last trigger time + 600

File, Line/s:

- USDFI.sol, L36—L77
- STABLE.sol, L29—L70

Codebase:

```

36 function mint(address _to↑, uint256 _amount↑) public onlyRole(MINTER_ROLE) {
37     require(
38         freeMintSupplyTotal ≥ _amount↑,
39         "ERC20: no more supply (total)"
40     );
41     require(
42         freeMintSupplyMinter[msg.sender] ≥ _amount↑,
43         "ERC20: no more supply (minter)"
44     );
45     freeMintSupplyTotal -= _amount↑;
46     freeMintSupplyMinter[msg.sender] -= _amount↑;
47     _mint(_to↑, _amount↑);
48 }
49
50
51
52 ftrace | funcSig
53 function addFreeMintSupply(address _address↑, uint256 _supply↑)
54 public
55 onlyRole(CONTROLLER_ROLE)
56 {
57     freeMintSupplyTotal += _supply↑;
58     freeMintSupplyMinter[_address↑] += _supply↑;
59     emit FreeMintSupplyAdded(_address↑, _supply↑);
60 }
61
62 ftrace | funcSig
63 function subFreeMintSupply(address _address↑, uint256 _supply↑)
64 public
65 onlyRole(CONTROLLER_ROLE)
66 {
67     require(
68         freeMintSupplyTotal ≥ _supply↑,
69         "ERC20: insufficient total supply"
70     );
71     require(
72         freeMintSupplyMinter[_address↑] ≥ _supply↑,
73         "ERC20: insufficient minter supply"
74     );
75     freeMintSupplyTotal -= _supply↑;
76     freeMintSupplyMinter[_address↑] -= _supply↑;
77     emit FreeMintSupplySubtracted(_address↑, _supply↑);
78 }

```

File, Line/s:

- StableMinter.sol, L79—L95

Codebase:


```
79     function setReceiverAndPercent(  
80         address[] memory _receiverAddress↑,  
81         uint16[] memory _percent↑  
82     ) external onlyOwner {  
83         require(  
84             _receiverAddress↑.length == _percent↑.length,  
85             "Arrays must have the same length"  
86         );  
87         uint256 totalPercent = 0;  
88         for (uint256 i = 0; i < _percent↑.length; i++) {  
89             totalPercent = totalPercent + _percent↑[i];  
90         }  
91         require(totalPercent == TOTAL_PERCENT, "must be 100%");  
92         receiver = _receiverAddress↑;  
93         percent = _percent↑;  
94         emit ReceiverAndPercentChanged(_receiverAddress↑, _percent↑);  
95     }
```



Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens

 **The owner cannot burn tokens**

Description

The owner is not able burn tokens without any allowances.

Comment

The owner of the contract cannot burn without any allowance from any address. Addresses can burn their own tokens by calling the “burn” function in the USDFI/ STABLE contracts.



Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses



The owner cannot blacklist addresses

Description	The owner is not able blacklist addresses to lock funds.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%



The owner cannot levy unfair taxes

Description

The owner is not able to set the fees above 25%

Comment

The owner is able to set the minting percentage amount for the receivers in the StableMinter but not for himself. This percentage setting is not the fee, it is the percentage minting amount of the stables token for the receivers.



Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Owner cannot lock the contract

 **The owner cannot lock the contract**

Description

The owner is not able to lock the contract by any functions or updating any variables.

Comment

N/A

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
1. USDFI.sol	<ul style="list-style-type: none"> ❖ MINTER_ROLE <ul style="list-style-type: none"> - Mint new tokens when the caller has free mint supplies ❖ CONTROLLER_ROLE <ul style="list-style-type: none"> - Add/Subtract free mint supplies to an address
2. STABLE.sol	<ul style="list-style-type: none"> ❖ MINTER_ROLE <ul style="list-style-type: none"> - Mint new tokens when the caller has free mint supplies ❖ CONTROLLER_ROLE <ul style="list-style-type: none"> - Add/Subtract free mint supplies to an address
3. StableMinter.sol	<ul style="list-style-type: none"> ❖ onlyOwner <ul style="list-style-type: none"> - Set the receiver and their percentages (must be 100% for all receivers in sum) to mint them new STABLE tokens during the "createNewStable" function which can be called by everyone

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g., Gnosis Safe
- Use of a timelock at least with a latency of e.g., 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement



- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.



Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

No low issues

Informational issues

#1 | Floating Pragma

File	Severity	Location	Status
USDFI.sol	Informational	L9	ACK
STABLE.sol	Informational	L2	ACK

Description - The contracts should be deployed with the same compiler version and flag that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

#2 | Conformity to Solidity naming conventions

File	Severity	Location	Status
StableMinter.sol	Informational	L30	ACK

Description - We recommend you use the following link for the naming conventions: <https://docs.soliditylang.org/en/latest/style-guide.html#naming-conventions>

Public constants should be named "TOTAL_SUPPLY" and private "_TOTAL_SUPPLY".



In your case, you should change the “Token” to “TOKEN”. Don’t forget to change it everywhere else in the code

#3 | Missing Zero Address Validation

File	Severity	Location	Status
USDFI.sol	Informational	L27-L29, L36	ACK
STABLE.sol	Informational	L19-L22, L29	ACK

Description - Make sure to validate that the address passed in the function parameters is “non-zero”.

The minter role address can mint to zero/dead address in the mint function.



Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.





**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY