



SOLIDProof

Bring trust into your projects

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

MADE IN GERMANY

FWDX

AUDIT

SECURITY ASSESSMENT

25. August, 2023

FOR



SolidProof_io



@solidproof_io



Introduction	3
Disclaimer	3
Project Overview	4
Summary	4
Social Medias	4
Audit Summary	5
File Overview	6
Imported packages	6
Components	7
Exposed Functions	7
Capabilities	8
Inheritance Graph	9
Audit Information	10
Vulnerability & Risk Level	10
Auditing Strategy and Techniques Applied	11
Methodology	11
Overall Security	12
Upgradeability	12
Ownership	13
Ownership Privileges	14
Minting tokens	14
Burning tokens	15
Blacklist addresses	16
Fees and Tax	17
Lock User Funds	18
Centralization Privileges	19
Audit Results	20

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	FWDX
Website	https://fwdx.xyz/
About the project	Decentralized P2P Swaps Experience FWDX's revolutionary P2P swap platform. Trade across multiple chains with firm quotes and zero slippage.
Chain	TBA
Language	Solidity
Codebase	TBA
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	N/A
Twitter	https://twitter.com/FWDX_Official
Facebook	N/A
Instagram	N/A
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	https://discord.gg/aKA2PXJD33
YouTube	N/A
TikTok	N/A
LinkedIn	N/A



Audit Summary

Version	Delivery Date	Change Log
v1.0	25. August 2023	<ul style="list-style-type: none"> · Layout Project · Automated/ Manual-Security Testing · Summary

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project. This analysis did not include functional testing (or unit testing) of the contract's logic. We cannot guarantee 100% logical correctness of the contract as it was not functionally tested by us.





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/FWDXSwapProtocolV1.sol	b3f7a29dee866e2b00fa05258197ba7cc9ae3437

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(local imports).

```
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol
./TransferHelper.sol
```

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. This contract imports some local contracts which are not audited by us, and we will not be responsible for any security cause for those contracts.





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 needed within visibility modifier, such as public, private or internal, which determines the access level of the variable.

Components

 Contracts	 Libraries	 Interfaces	 Abstract
1	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
28	0








External	Internal	Private	Pure	View
13	22	0	0	18

StateVariables

Total	 Public
16	7



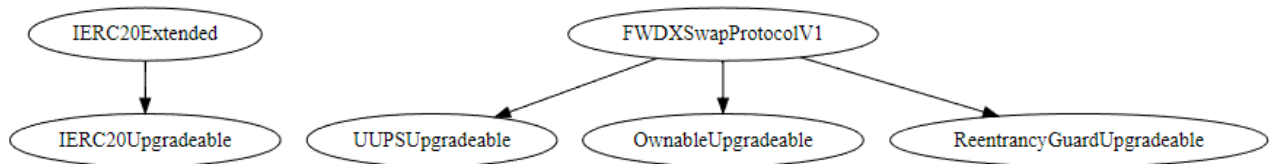
Capabilities

<div>Solidity Versions observed</div>	<div> Experimental Features</div>	<div> Can Receive Funds</div>	<div> Uses Assembly</div>	<div> Has Destroyable Contracts</div>	
<div>0.8.19</div>	<div>-----</div>		<div>Yes</div> <div>(1 asm blocks)</div>	<div>-----</div>	
<div> Transfers ETH</div>	<div> Low-Level Calls</div>	<div> Delegate Call</div>	<div> Uses Hash Functions</div>	<div> ECRecover</div>	<div> New/Create/Create2</div>
			<div>yes</div>		



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 the 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 an upgradable

✗ Deployer can update the contract with new functionalities.

Description	The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.
Example	We assume that you have funds in the contract, and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.
Comment	The contract contains upgradable functionality where the ownable and initializing parameters can be upgraded after the deployment. Also, we recommend disabling the initializers once this function has been called.

File/ Line(s): L14-17, L21-32

Codebase: FWDXSwapProtocolV1.sol

```
contract FWDXSwapProtocolV1 is
    UUPSUpgradeable,
    OwnableUpgradeable,
    ReentrancyGuardUpgradeable
```

```
function initialize(address payable _treasury) public initializer {
    __Ownable_init();
    __UUPSUpgradeable_init();
    require(
        _treasury != address(0),
        "Initiate:: _treasury can not be zero address"
    );
    treasury = _treasury;
    networkId = getChainID();
    marketFee = 25;
    stableCoinMarketFee = 5;
}
```

Ownership

The ownership is not renounced

✗ The ownership is not renounced

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

Note – *The contract cannot be considered as renounced till it is not deployed or having some functionality that can change the state of the contract.*



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 cannot mint new tokens

 **The owner cannot mint new tokens**

Description	The owner is not able to mint new tokens once the contract is deployed.
Comment	N/A



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

N/A



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 can set fees more than 25%.

✗ The owner can set fees more than 25%

Description

For example, a decentralized exchange (DEX) smart contract may charge a fee for each trade executed on the platform. This fee can be set by the owner of the contract and may be a percentage of the trade value or a flat fee 25%. In other cases, the owner of the smart contract may set fees for accessing or using certain features of the contract. For instance, a subscription-based service smart contract may charge a monthly or yearly fee for access to premium features.

It's important to note that the fees set by the owner of a smart contract may not be the same as the gas fees required to execute the contract on the blockchain. Gas fees are generally set by the network and vary based on factors such as network congestion and the complexity of the transaction. The fees set by the contract owner, on the other hand, are independent of gas fees and are typically charged in addition to gas fees. Overall, fees set by the owner of a smart contract can provide an additional source of revenue for the contract's owner and can help to ensure the sustainability of the contract over time.

Comment

The owner can set the Market fee to 100%, if done so then the user needs to pay double in order to create a market and the users who fill their market will also have to pay 100% of the `_swapInAmount`.

File/ Line(s): L655-662, L665-678

Codebase: FWDXSwapProtocolV1.sol

```
function updateMarketFee(uint256 _newMarketFee) external onlyOwner {
    require(
        _newMarketFee <= 10000,
        "UpdateMarketFee:: Basis points cannot exceed 10000"
    );
    emit PlatformFeeUpdated(_marketFee, _newMarketFee, false);
    _marketFee = _newMarketFee;
}
```



```
ftrace | funcSig
function updateStableCoinMarketFee() {
    uint256 _newStableCoinMarketFee;
} external onlyOwner {
    require(
        _newStableCoinMarketFee <= 10000,
        "UpdateStableCoinMarketFee:: Basis points cannot exceed 10000"
    );
    emit PlatformFeeUpdated(
        stableCoinMarketFee,
        _newStableCoinMarketFee,
        true
    );
    stableCoinMarketFee = _newStableCoinMarketFee;
}
```





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 token 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.

Contract owner cannot lock user funds.



The owner cannot lock user funds.

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
FWDXSwapProtocolV1.sol	<ul style="list-style-type: none"> ➤ The owner can update the treasury wallet. ➤ The owner can set market fees up to 100%. ➤ The owner can set stable coin market fees up to 100%. ➤ The owner can change the stablecoin. ➤ The owner can add multiple stablecoins and change the stablecoin status at once.

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 Result

#1 | The owner can set fees up to 100%.

File	Severity	Location	Status
FWDXSwapProtocolV1.sol	Medium	L655-662, L665-678	ACK

Description – The owner can set the Market fee to 100%, if done so then the user needs to pay double in order to create a market and the users who fill their market will also have to pay 100% of the `_swapInAmount`.

Remediation – Add the functionality in the contract that the amount of fees should not be greater than 25%.

#2 | Missing zero or dead address check.

File	Severity	Location	Status
FWDXSwapProtocolV1.sol	Low	L645-652, L681-691, L694-704	Open

Description – Check that the address cannot be set as a zero or dead address.

#3 | Missing Visibility.

File	Severity	Location	Status
FWDXSwapProtocolV1.sol	Low	L83	Open

Description – Add 'public' or 'private' while initializing a variable or mapping in the contract.

#4 | Redundant check.

File	Severity	Location	Status
FWDXSwapProtocolV1.sol	Low	L268-271	Open

Description – The amount is already checked in the function on line 254. Remove unnecessary codes to avoid the high gas fees.



#5 | Unnecessary code.

File	Severity	Location	Status
FWDXSwapProtocolV1.sol	Low	L707-711	Open

Description – Remove unused functions to avoid the high gas fees.

#6 | NatSpec Documentation missing.

File	Severity	Location	Status
FWDXSwapProtocolV1.sol	Informational	--	Open

Description – If you started to comment on your code, also comment on all other functions, variables, etc.



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