Guild Audits

NOVEMBER 2022

XENDFINANCE SMART CONTRACT AUDIT REPORT

By Guild Audits

TABLE OF CONTENTS

Executive Summary	3
Protocol Overview	4
Project Audit Scope and Findings	6
Mode of Audit and Methodologies	8
Functional Testing	9
Report of Findings	10 - 16
Closing Summary	17
Appendix	17
Disclaimer	17

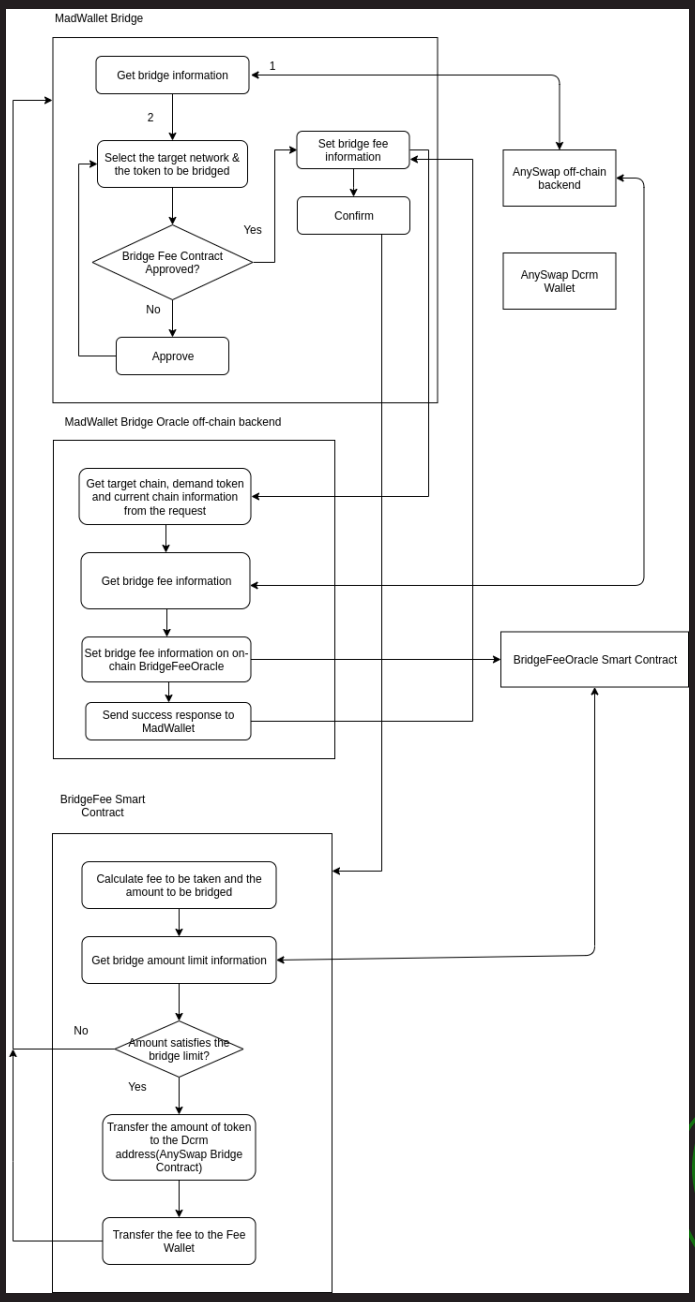
EXECUTIVE SUMMARY

MahdWalletDEXAggregator is an aggregator smart contract that allows for swapping of tokens using different Router. This contract integrates TransparentUpgradeableProxy from openzeppelin, It also inherits Access Control contract from Openzeppelin upgradeable standard; for the purpose of administration management and for the purpose of upgradability of contracts.

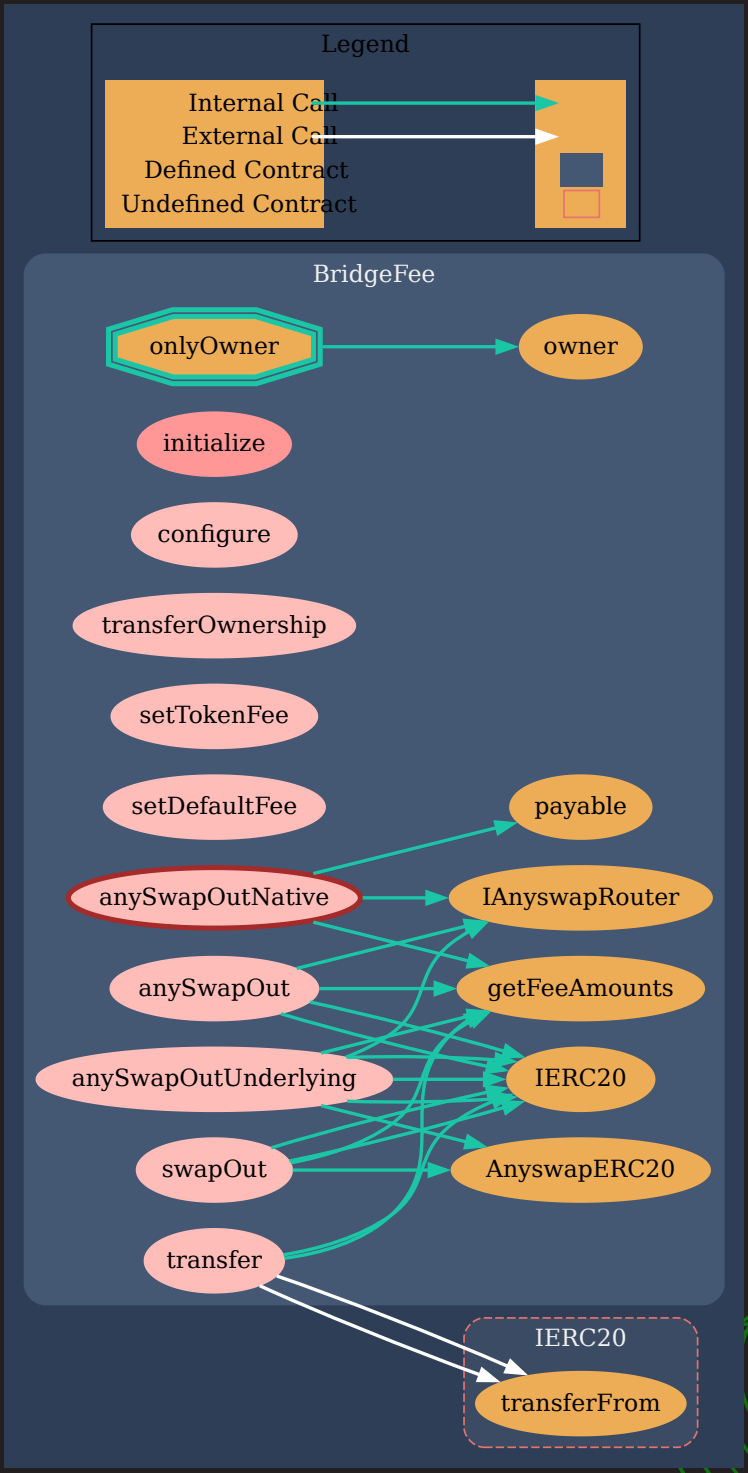
MahdWalletBridgeSmartContract is designed to allow users to interact with the AnySwap router for a swap. This contract features an inbuilt ownership management and also integrates the Openzeppelin TransparentUpgradeableProxy format.

PROTOCOL OVERVIEW

MadWallet Bridge



Bridge Fee


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BridgeFee
onlyOwner
owner
initialize
configure
transferOwnership
setTokenFee
setDefaultFee
payable
anySwapOutNative
IAnyswapRouter
anySwapOut
getFeeAmounts
anySwapOutUnderlying
IERC20
swapOut
AnyswapERC20
transfer
IERC20
transferFrom

PROJECT AUDIT SCOPE AND FINDINGS

The motive of this audit is to review the codebase of **Xend-Finance** MahdWalletDEXAggregator contracts and MahdWalletBridgeSmartContract for the purpose of achieving secured, corrected and quality contracts.

Number of Contracts in Scope:

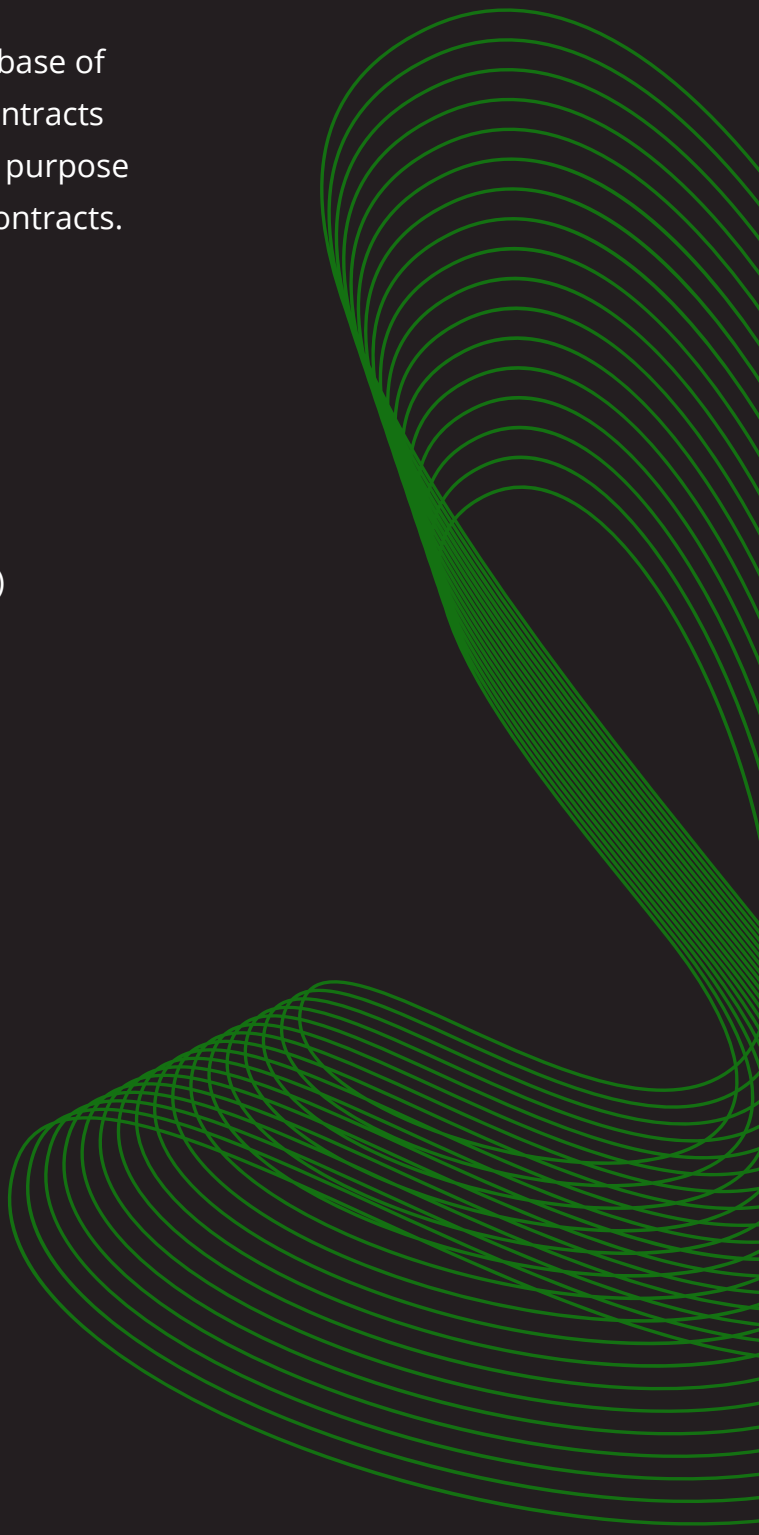
- MainSwapRouter Contract (191 SLOC)
- PolygonSwapRouer Contract (191 SLOC)
- SwapRouer Contract (191 SLOC)
- SwapRouerProxy Contract (12 SLOC)
- SwapRouerProxyAdmin Contract (8 SLOC)
- BridgeFee Contract (271 SLOC)
- BridgeProxy Contract (8 SLOC)

Link to Project codebase:
Null.

Duration for Audit:
October 15, 2022 to November 20, 2022

Audit Methodology:

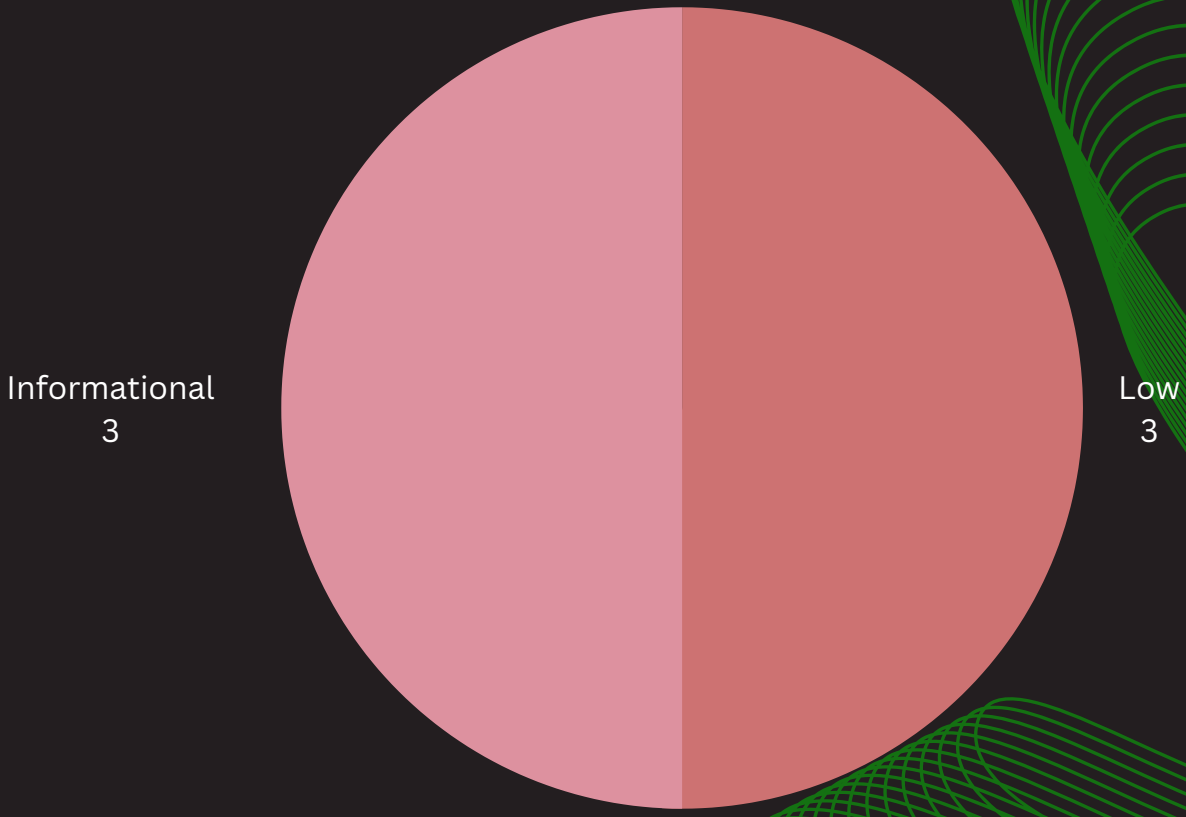
- Automated Testing
- Functional Testing
- Manual Review



Issues found:

- High: 0
- Medium: 0
- Low: 3
- Informational: 3

Chart Illustration



MODE OF AUDIT AND METHODOLOGIES

The mode of audit carried out in this audit process is as follow:

- **Manual Review**: This is the first and principal step carried out to understand the business logic behind a project. At this point, it involves research, discussion and establishment of familiarity with contracts. Manual review is critical to understand the nitty-gritty of the contracts.
- **Automated Testing:** This is the running of tests with audit tools to further cement discoveries from the manual review. After a manual review of a contract, the audit tools which could be Slither, Echidna, or Mythril are run on the contract to find out issues.
- **Functional Testing:** Functional testing involves manually running unit, static, and dynamic testing. This is done to find out possible exploit scenarios that could be used to steal money from the contracts. This helps understand the functionality of the contracts and find out lapses in the reverts check in contract.

The methodologies for establishing severity issues:

- High Level Severity Issues
- Medium Level Severity Issues
- Low Level Severity Issues
- Informational Level Severity Issues

FUNCTIONAL TESTING

Some functional testing were carried out to help ascertain code security:

- Should get the usdc address assigned to contract. (318ms)
- Should get the owner of the contract after deployment
- Should revert when configure function is called by non-owner
- Should revert when null address is passed as feeAddress
- Should successfully configure the feeAddress and fee info by owner
- Should revert when owner intends to transfer ownership to null address
- Should successfully transfer ownership from old to new address
- Should successfully set the Fee of a token address
- Should revert if caller passes null address as dcrmAddress
- Should revert if caller passes zero amount
- Should revert when router address is a null address
- Should revert if insufficient native token is sent into the contract
- Should call the swapOut function

REPORT OF FINDINGS

A. MAHDWALLETDEXAGGREGATOR CONTRACT

HIGH SEVERITY ISSUES

- **No issues.**

MEDIUM SEVERITY ISSUES

- **No issues.**

LOW SEVERITY ISSUES

- **NO Two factor verification steps. (Ownable.sol)**
  - TransferOwnership from Ownable.sol does not require a confirmation from the new valid owner to confirm his acceptance of the role, this can lead to permanently giving admin role to a wrong owner and which cannot be revoked again.

Recommendation:
It is recommended to use the two factor verification steps which gives the opportunity to revoke a new owner incase of any mistake of transfer or malicious act noticed from the new owner.

Status: Acnowledged

- **If conditions omit transfer of fee amount in case of a user sending the same amount as the swap amount. (MainnetSwapRouter.sol L175)**
  - A user who sent the same amount has the swapping amount will escape paying the fee attributed to each swap made.

```
if(swappingTokenAmount < amount⬆){
    if(tokenFrom⬆ != address(0)){
        IERC20Upgradeable(tokenFrom⬆).safeTransfer(feeAddress, feeAmount);
    }
    else{
        (success,result) = payable(feeAddress).call{value: feeAmount}("");
        require(success, "Failed to send ETH");
    }
}
```

Recommendation:
The condition swappingTokenAmount < amount should be able to capture if a user is trying to swap the same amount sent swappingTokenAmount =< amount

Status: Acnowledged

INFORMATIONAL SEVERITY ISSUES

- **Gas Optimization (MainnetSwapRouter.sol L93,L96, L112, L116, L129, L134;)**
  - Reading from storage variable for more than once cost gas and will require users to pay high gas cost because of logic implementation in executing the function.

```
if(keccak256(abi.encodePacked((aggregatorId⬆))) == keccak256(abi.encodePacked(("0xFeeDynamic")))){
    swapRouterAddress = zeroExRouter;
    if(tokenFrom⬆ != address(0)){
        if(IERC20Upgradeable(tokenFrom⬆).allowance(address(this), zeroExRouter) == 0){
            IERC20Upgradeable(tokenFrom⬆).approve(zeroExRouter, type(uint256).max);
        }
    }
}
else if (keccak256(abi.encodePacked((aggregatorId⬆))) == keccak256(abi.encodePacked(("oneInchV4FeeDynamic")))){
    swapRouterAddress = oneInchRouter;
    if(tokenFrom⬆ != address(0)){
        if(IERC20Upgradeable(tokenFrom⬆).allowance(address(this), oneInchRouter) == 0){
            IERC20Upgradeable(tokenFrom⬆).approve(oneInchRouter, type(uint256).max);
        }
    }
}
else if (keccak256(abi.encodePacked((aggregatorId⬆))) == keccak256(abi.encodePacked(("airswapV3FeeDynamic")))){
    swapRouterAddress = airswapWrapper;
    if(tokenFrom⬆ != address(0)){
        if(IERC20Upgradeable(tokenFrom⬆).allowance(address(this), airswapWrapper) == 0){
            IERC20Upgradeable(tokenFrom⬆).approve(airswapWrapper, type(uint256).max);
        }
    }
}
else if (keccak256(abi.encodePacked((aggregatorId⬆))) == keccak256(abi.encodePacked(("paraswapV5FeeDynamic")))){
    swapRouterAddress = paraswapRouter;
    if(tokenFrom⬆ != address(0)){
        address proxy = IParaswapRouter(paraswapRouter).getTokenTransferProxy();
        if(IERC20Upgradeable(tokenFrom⬆).allowance(address(this), proxy) == 0){
            IERC20Upgradeable(tokenFrom⬆).approve(proxy, type(uint256).max);
        }
    }
}
```

Recommendation:
It is advisable to cache storage variables in memory and read from memory in all other instances, which is cheaper. It is seen that the Router address was actually cached in swapRouterAddress, a local variable, so advisable to read the value from the cached data and not the storage.

Status: Acnowledged

REPORT OF FINDINGS

B. MAHDWALLETBRIDGESMARTCONTRACT

HIGH SEVERITY ISSUES

- **No issues.**

MEDIUM SEVERITY ISSUES

- **No issues.**

LOW SEVERITY ISSUES

- **Two-Step Mechanism for Ownership Transfer (transferOwner | L56)**
  - Inasmuch as there is a function check to prevent assigning address zero as the new owner, when the function is called with a mistaken address as parameter by the existing owner, the ownership privileges are immediately transferred to this unknown address and the original owner loses the contract and will be unable to retrieve their ownership. Also, the function fails to check that an owner doesn't pass his own address as a parameter, for this will cause a waste of gas.

Recommendation:
Adding an extra function to permit the newly assigned owner to claim ownership will eradicate the issue of automatic assignment of mistaken address. When an address is set as the owner using the transferOwner function, the contract can have additional function, say, updateOwner, that will be called by the assigned person and then, afterwards, the assigned person becomes the new owner and then the former no longer has the admin or ownership privileges.

Status: Acnowledged

- **Unindexed Event (OwnershipTransferred | L25)**
  - Logging events are not sufficient alone for some critical actions in a contract. If events are logged and indexed, the parameters emitted are searchable and filter from a bunch of other emitted information.

```
event OwnershipTransferred(address oldOwner, address newOwner);
```

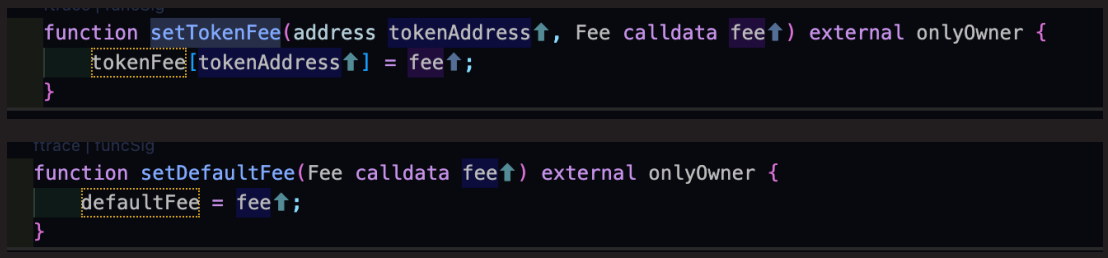
Recommendation:
It is recommended to index affected events to ease the processes of searching for information from emitted events.

Status: Closed

- **Missing Event Emission (L46, L66, L72)**
  - When significant actions happen from function calls, it is expected that we track these actions or changes in order to aid in the filtering or gathering of information. These actions could include state variable changes that are paramount to the contract. Functions that are affected include the following:

```
function configure(
    address _feeAddress,
    Fee calldata _defaultFee
) external override onlyOwner {
    require(_feeAddress != address(0), "invalid fee address");

    defaultFee = _defaultFee;
    feeAddress = _feeAddress;
}
```



```
function setTokenFee(address tokenAddress, Fee calldata fee) external onlyOwner {
    tokenFee[tokenAddress] = fee;
}

function setDefaultFee(Fee calldata fee) external onlyOwner {
    defaultFee = fee;
}
```

- **configure**
- **setTokenFee**
- **setDefaultFee**

Recommendation:
Consider emitting events for all of the affected functions in order to help assemble and filter events in relation to the contract.

Status: Closed

CLOSING SUMMARY

There were discoveries of some low and informational issues after the audit. The audit team thereafter suggested some remediation to help remedy the issues found in the contract.

APPENDIX

- *Audit: The review and testing of contracts to find bugs.*
- *Events: In smart contracts, several pieces of information are important not to be missed. For this reason, an event is a solidity feature that helps us track some parameters when they are emitted.*
- *Issues: These are possible bugs that could lead exploits or help redefine the contracts better.*
- *Slither: A tool used to automatically find bugs in a contract.*
- *Severity: This explains the status of a bug.*

DISCLAIMER

While the audit report is aimed at achieving a quality codebase with assured security and correctness, it should not be interpreted as a guide or or recommendation for people to invest in **Xend-finance** contracts.

With smart contract audit being a multifaceted process, we admonish the **Xend-finance** team to carry out further audit from other audit firms or provide a bug bounty program to ensure that more critical audit is done to the contract.

GUILD AUDITS

Guild Audits is geared towards providing blockchain and smart contract security in the fuming web3 world. The firm is passionate about remedying the constant hacks and exploits that deters the web3 motive.