Security Audit Report

DMM Protocol

SECBIT

Apr 22, 2020

1. Introduction

DMM (DeFi Money Market) Protocol is an Ethereum-based and decentralized protocol that allows any holder of an Ethereum-based digital asset to earn interest. SECBIT Labs conducted an audit from Mar 23th to Apr 22th, 2020, including an analysis of Smart Contracts in 3 areas: code bugs, logic flaws, and risk assessment. The audit results show that the DMM Protocol has no critical security risks, and the SECBIT team has some tips on logical implementation, potential risks, and code revising (see part 4 for details).

Type	Description	Level	Status
Logical Implementation	4.3.1 `increaseTotalSupply()` and `decreaseTotalSupply()` function may cause unexpected change of `interestRate`	Info	Discussed
Code Revising	4.3.2 In the implementation of `DMMToken` contract, `safeTransfer()` is not applied in `transferUnderlyingOut()`	Low	Discussed
Logical Implementation	4.3.3 `getCurrentExchangeRate()` in `DmmTokenLibrary` contract may not support complex calculation of interest	Info	Discussed
Potential Risk	4.4.1 In the implementation of `PreCoordinator` contract, the response from oracles may be reverted in some circumstances	Info	Discussed
Potential Risk	4.4.2 Heavy reliance on external oracles	Info	Discussed
Potential Risk	4.4.3 Instability of Ether deposits and lending	Medium	Discussed

2. Project Information

This part describes the basic information and code structure.

2.1 Basic information

The basic information about the DMM Protocol is shown below:

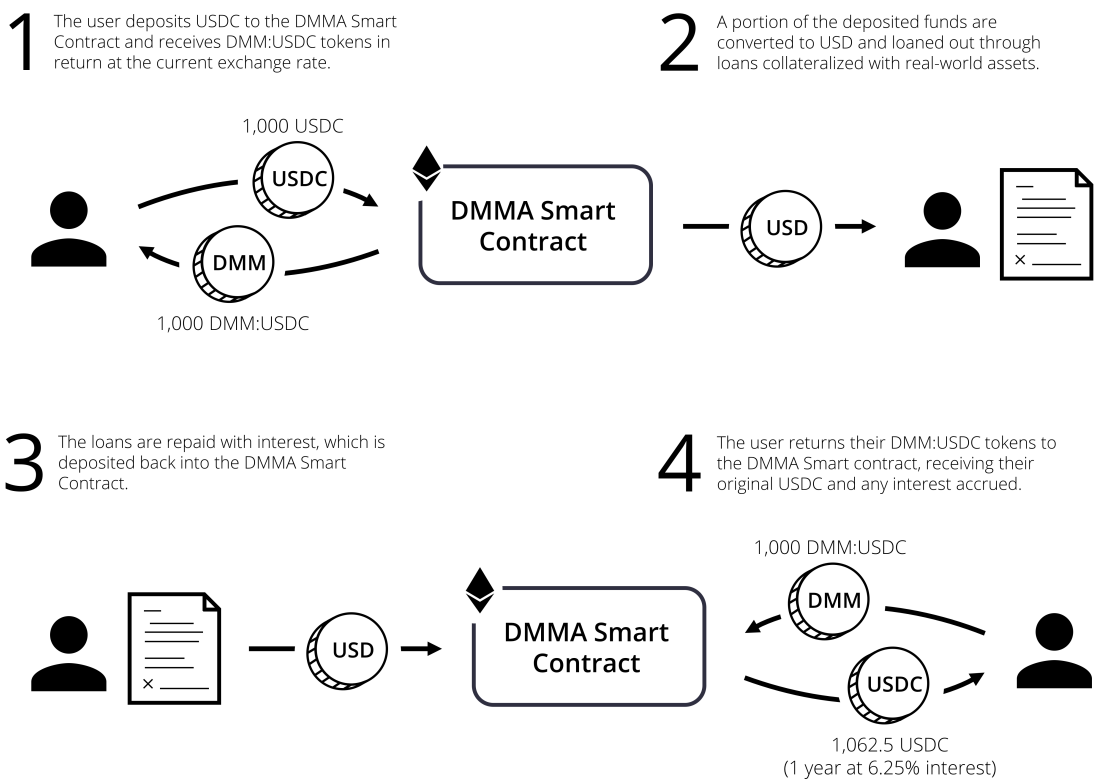
- Project website
    - https://defimoneymarket.com/

- Design details of the protocol
    - https://defimoneymarket.com/DMM-Ecosystem.pdf
- Smart contract code
    - https://github.com/defi-money-market-ecosystem/protocol
    - commit-898e09f

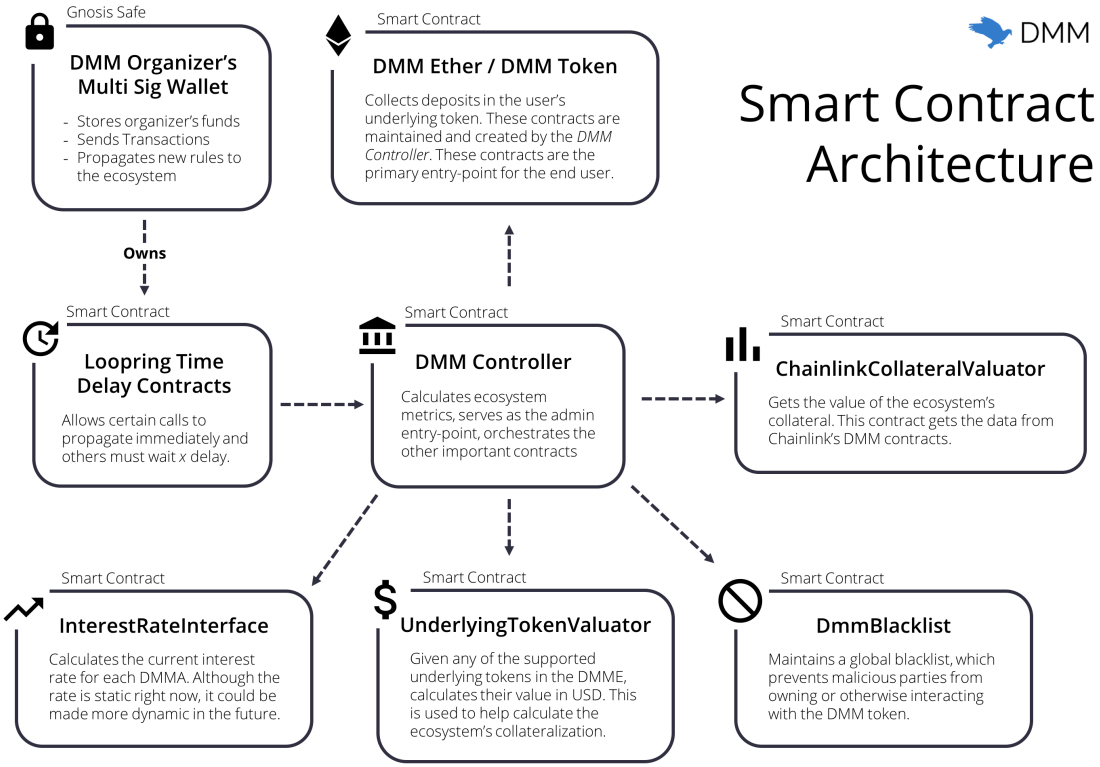
2.2 Project Overview

- Functions and compositions of the DMM Protocol
    - Functions provided in the DMM Protocol
        1. Link Ethereum digital assets and real-world assets
        2. Support different digital assets (e.g. DAI, USDC, etc.) to participate in the DMM Protocol
        3. Enable Ethereum assets to be converted to a specific DMM Token with interest
    - Compositions of the DMM Protocol
        1. DMMA (DeFi Money Market Account)
            - Account specific to each Ethereum digital asset in the DMM Protocol
        2. DMM Token (DeFi Money Market Token)
            - A specific Ethereum asset will be swapped into DMM Token in the DMM protocol
            - The digital asset is pledged for future interest payments. Users could deposit and exit with no restrictions.
        3. DMMW (DeFi Money Market Wrapper)
            - DMMW is an ERC20 smart contract wrapper designed to link any Ethereum token with the DMM Protocol by creating a specific DMM Token, which is backed by real-world assets represented on-chain
    - The mechanism to support the long-term run of the DMM Protocol
        1. Income generated by real-world assets is greater than APY of DMM
        2. Real-world assets backed are greater than those DMM issued
        3. The backing process is transparent via Chainlink

- The flow of funds in the DMM Protocol



- The architecture of the DMM Protocol



2.3 Contract List

The following content shows the main contracts included in DMM project:

Contract	Description
AtmLike.sol	ATM-like deposit and withdraw operations
DelayedTransaction.sol	Contract for time delay transactions
DelayedOwner.sol	Claimable time delay contract which serves as a proxy contract
DmmBlacklistable.sol	Blacklist functions
DmmController.sol	Controller orchestrating the other important contracts
DmmEther.sol	A wrapper around Ether and WETH for minting DMM
DmmEtherFactory.sol	Contract for deploying DmmEther contract
DmmToken.sol	DmmToken main contract
DmmTokenFactory.sol	Contract for deploying Dmm Token contract
InterestRateImplV1.sol	Contract providing current interest rate
OffChainAssetValuatorImplV1.sol	Contract for obtaining value of off-chain assets on Chainlink
OffChainCurrencyValuatorImplV1.sol	Contract for obtaining value of currencies on Chainlink
UnderlyingTokenValuatorImplV1.sol	Contract for obtaining value of tokens including DAI and USDC
UnderlyingTokenValuatorImplV2.sol	Contract for obtaining value of tokens including DAI, WETH and USDC
IDelayedTransaction.sol	DelayedTransaction interface

IDmmController.sol	DmmController interface
IDmmToken.sol	DmmToken interface
IDmmTokenFactory.sol	TokenFactory interface
InterestRateInterface.sol	InterestRate interface
IOffChainAssetValuator.sol	OffChainAssetValuator interface
IOffChainCurrencyValuator.sol	OffChainCurrencyValuator interface
IOwnable.sol	Ownable interface
IPausable.sol	Pausable interface
IUnderlyingTokenValuator.sol	UnderlyingTokenValuator interface
IUsdAggregator.sol	UsdAggregator interface
IWETH.sol	WETH interface
DmmTokenLibrary.sol	Library for DmmToken operations
StringHelpers.sol	Helper for converting address to string
AddressUtil.sol	Library for address-related operations
Blacklistable.sol	The Blacklistable contract
BytesUtil.sol	Library for byte-related operations
Claimable.sol	The Claimable contract
ERC20.sol	ERC Token contract
CommonConstants.sol	Contract storing common constant values

2.4 Deployed Contracts

The following content shows the main contracts of the DMM Protocol deployed on the Ethereum Network where the transactions test correctly for each of the main functions of each contract:

Contract	Address
DmmController	0x4CB120Dd1D33C9A3De8Bc15620C7Cd43418d77E2
DmmBlacklistable	0x516d652e2f12876f5f0244aa661b1c262a2d96b1
DmmEtherFactory	0x1186d7dff910aa6c74bb9af71539c668133034ac
DmmTokenFactory	0x42665308f611b022df2fd48757a457bec12ba668
InterestRateImplV1	0x6f2a3b2efa07d264ea79ce0b96d3173a8feacd35
OffChainAssetValuatorImplV1	0xace9112efe78d9e5018fd12164d30366ca629ab4
OffChainCurrencyValuatorImplV1	0x35cceb6ed6eb90d0c89a8f8b28e00ae23545312b
DelayedOwner	0x9e97ee8631da9e96bc36a6bf39d332c38d9834dd
UnderlyingTokenValuatorImplV2	0x693aa8ead81d2f88a45e870fa7e25f84ca93ca4d
mDAI	0x06301057d77d54b6e14c7faffb11ffc7cab4eaa7
mUSDC	0x3564ad35b9E95340E5Ace2D6251dbfC76098669B
mETH	0xdf9307dff0a1b57660f60f9457d32027a55ca0b2

3. Code Analysis

This part describes details of code assessment, including 2 items: "role classification" and "functional analysis".

3.1 Role Classification

There are several key roles in the protocol, namely Delayed Owner, Protocol Owner, Common Account, and Authorized Nodes.

- Delayed Owner
    - Description Owner of time delay contract
    - Authority
        - Add, execute and cancel delayed certain transactions
        - Set delay parameters
    - Method of Authorization The creator of the contract, or authorized by transferring the ownership of the contract
- Blacklist Owner
    - Description Owner of DmmBlacklistable contract
    - Authority Allows accounts to be blacklisted
    - Method of Authorization The creator of the contract, the ownership cannot be transferred
- Protocol Owner
    - Description The administrator of the DMM Protocol, owner of DMM Controller contract, each DMM Token contract and DMM TokenFactory contract
    - Authority
        - All common accounts' authorities
        - Transfer the ownership of the DMM Protocol
        - Mint and burn tokens from DMM Token contract
        - Create and manage DMM Token/Market
        - Pause DMM Controller Contract
        - Set contract address of the DMM Protocol compositions
    - Method of Authorization The creator of the contract, or authorized by transferring the ownership of the contract
- Common Account
    - Description Any Ethereum account

    - Authority
        - Mint DMM Token with underlying token or ether
        - Redeem underlying token or ether with DMM Token
        - Non-owner operations of DMM Token
    - Method of Authorization Everyone can be a common account
- Authorized Nodes
    - Description The accounts are authorized to answer the requests in chainlink oracles
    - Authority Update the values of requests from OffChainAssetValuator
    - Method of Authorization Authorized by the owner of Oracle contract

3.2 Functional Analysis

We can divide the key functions of the DMM Protocol into several parts:

- Creating and managing DeFi Money Markets
    - The DMM controller creates DeFi Money Markets with `addMarket()` and `addMarketFromExistingDmmToken()`.
    - The DMM controller manages DeFi Money Markets with `enableMarket()`, `disableMarket()`, `increaseTotalSupply()`,`decreaseTotalSupply()`.
- Allowing common accounts to transfer, mint and redeem DMM Tokens with gasless transactions
    - Common accounts transfer DMM Tokens with `transferFromGaslessRequest()`
    - Common accounts mint DMM Tokens with `mintFromGaslessRequest()`
    - Common accounts redeem DMM Tokens with `redeemFromGaslessRequest()`
- Maintaining a global blacklist to prevent malicious parties
    - `DmmBlacklistable` contract provides the blacklist functions
- Updating the value of the ecosystem's collateral with Chainlink's DMM contracts
    - `OffChainAssetValuator` contract updates the value of assets from Chainlink contract. The contract sends requests with `submitGetOffChainAssetsValueRequest()` and receive answers with `fulfillGetOffChainAssetsValueRequest()`.
    - `UnderlyingTokenValuator` contract updates the USD value of Ether from interfaces of `ethUsdAggregator`.

4. Audit Detail

This part describes the process and detailed results of the audit and demonstrates all found problems and potential risks.

4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Labs. We analyzed the project for code bugs, logical implementation, and potential risks. The process consists of four steps:

- Fully analysis of code line by line.
- Evaluation of vulnerabilities and potential risks revealed in the source code.
- Communication on assessment and confirmation.
- Audit report writing.

4.2 Audit Result

After scanning with adelaide, sf-checker, and badmsg.sender (internal version) developed by SECBIT Labs and open source tools including Mythril, Slither, SmartCheck, and Securify, the auditing team performed a manual assessment. The team inspected the contract line by line and the result could be categorized into twenty-one types:

Number	Classification	Result
1	Normal functioning of features defined by the contract	✓
2	No obvious bug (e.g. overflow, underflow)	✓
3	Pass Solidity compiler check with no potential error	✓
4	Pass common tools check with no obvious vulnerability	✓
5	No obvious gas-consuming operation	✓
6	Meet with ERC20	✓
7	No risk in low-level call (call, delegatecall, callcode) and in-line assembly	✓
8	No deprecated or outdated usage	✓

9	Explicit implementation, visibility, variable type and Solidity version number	✓
10	No redundant code	✓
11	No potential risk manipulated by timestamp and network environment	✓
12	Explicit business logic	✓
13	Implementation consistent with annotation and other info	✓
14	No hidden code about any logic that is not mentioned in design	✓
15	No ambiguous logic	✓
16	No risk threatening the developing team	✓
17	No risk threatening exchanges, wallets, and DApps	✓
18	No risk threatening token holders	✓
19	No privilege on managing others' balances	✓
20	No minting method	! *
21	Correct managing hierarchy	✓

* : Two functions of DMM token are related to minting, including `mint()` and `increaseTotalSupply()`. The former is where the user deposits an underlying token in exchange for a DMM token, and the latter controls the total supply of DMM tokens available to the protocol by the administrator. The balance of any particular user will not be affected for unjustified reasons in both cases.

4.3 Issues

4.3.1 `increaseTotalSupply()` and `decreaseTotalSupply()` function may cause unexpected change of `interestRate`

Risk Type	Risk Level	Impact	Status
Logical Implementation	Info	Protocol Stability	Discussed

Description

Protocol Owner can mint and burn DMM Tokens with `increaseTotalSupply()` and `decreaseTotalSupply()` function without the exchange of underlying tokens. The call of functions above changes `totalSupply` and consequently the calculation of `activeSupply()` and `getInterestRate()` function, which may cause unexpected change of `interestRate`. While the investors are sensitive to the interest rate of DMM Tokens, it is necessary to avoid significant changes in the interest rate.

Suggestion

Since the interest rate and its corresponding growth/decay function is not determined yet, the algorithm of growth/decay function is recommended to consider the factor of the use of `increaseTotalSupply()` and `decreaseTotalSupply()` function.

Status

The issue has been discussed and the project team was aware of the interest rate side effect. They would consider this when implementing new interest models.

4.3.2 In the implementation of `DMMToken` contract, `safeTransfer()` is not applied in `transferUnderlyingOut()`

Risk Type	Risk Level	Impact	Status
Code Revising	Low	Standardization	Discussed

Description

The `safeTransfer()` function provides a safe method for transferring tokens and is applied in most of the functions containing token transfers in the DMM Protocol like `adminWithdrawFunds()`. While this method is not applied in `transferUnderlyingOut()`.

```
    function transferUnderlyingOut(address recipient, uint
underlyingAmount) internal {
        address underlyingToken =
controller.getUnderlyingTokenForDmm(address(this));
        IERC20(underlyingToken).transfer(recipient, underlyingAmount);
    }
```

Suggestion

Although underlying tokens are adding to the protocol by admins and most stable coins adopt the correct ERC20 interface, it is still recommended to apply the `safeTranfer()` interface in `transferUnderlyingOut()` to avoid risks and standardize the code.

Status

The issue has been fixed in the repository. For the deployed contract, the project team will be careful by only listing tokens that employ standard interfaces and implementations.

4.3.3 `getCurrentExchangeRate()` in `DmmTokenLibrary` contract may not support complex calculation of interest

Risk Type	Risk Level	Impact	Status
Logical Implementation	Info	Accuracy of calculation	Discussed

Description

The `accrueInterest()` function assumes the accrued interest grows linearly. The formula in `accrueInterest()` function may not capture the accrued interest correctly if the interest changes are complex, or if it goes through multiple adjustments midway. Or when the update interval is long enough, the error of calculated interest could be large and there may even be room for attack. Consider that exchangeRate is a global state and the updates are bound to continue with each user's use, the update interval is small enough that the calculation error is small and therefore less risky for now.

```
    function getCurrentExchangeRate(Storage storage _storage, uint
interestRate) internal view returns (uint) {
        if (_storage.exchangeRateLastUpdatedTimestamp >=
block.timestamp) {
            // The exchange rate has not changed yet
            return _storage.exchangeRate;
        } else {
            uint diffInSeconds =
block.timestamp.sub(_storage.exchangeRateLastUpdatedTimestamp,
"INVALID_BLOCK_TIMESTAMP");
            return accrueInterest(_storage.exchangeRate, interestRate,
diffInSeconds);
        }
    }
```

Suggestion

It is recommended to consider the accuracy of the calculation of interest in conjunction with future interest models.

Status

The issue has been discussed. The development team purposely kept interest rate accrual simple for these assets. AKA linear. Overall, the objective is to provide the interest rate they advertise with no surprises. In the future, along with what was mentioned in 4.3.1 (interest rates being affected by minting) they will likely use a step function, such that slight deviations in the total supply as well as the active supply don't modify rates.

4.4 Risks

4.4.1 In the implementation of `PreCoordinator` contract, the response from oracles may be reverted in some circumstances

Risk Type	Risk Level	Impact	Status
Potential Risk	Info	Chainlink Oracle	Discussed

Description

In the implementation of `PreCoordinator` contract, the status of a service agreement is indicated with `serviceAgreements[_saId].activeRequests`. When the amount of responses reaches `minResponses`, `serviceAgreements[_saId].activeRequests` is set to 0 and `requesters[cbRequestId]` is set to initial value. The responses after the update of answer still counts, thus when the amount of subsequent responses reaches `minResponses`, `serviceAgreements[saId].activeRequests = serviceAgreements[saId].activeRequests.sub(1);` will be executed again. The overflow occurs and the response will be reverted.

```
  function chainlinkCallback(bytes32 _requestId, int256 _data)
    externalf
    recordChainlinkFulfillment(_requestId)
    returns (bool)
  {
    uint256 minResponses =
serviceAgreements[serviceAgreementRequests[_requestId]].minResponses;
    bytes32 cbRequestId = requests[_requestId];
    bytes32 saId = serviceAgreementRequests[_requestId];
    delete requests[_requestId];
    delete serviceAgreementRequests[_requestId];
    emit ServiceAgreementResponseReceived(saId, cbRequestId,
msg.sender, _data);
    if (requesters[cbRequestId].responses.push(_data) == minResponses)
{
      serviceAgreements[saId].activeRequests =
serviceAgreements[saId].activeRequests.sub(1);
      Requester memory req = requesters[cbRequestId];
      delete requesters[cbRequestId];
      int256 result = Median.calculate(req.responses);
      emit ServiceAgreementAnswerUpdated(saId, cbRequestId, result);
      // solhint-disable-next-line avoid-low-level-calls
```

```
        (bool success, ) =
req.callbackAddress.call(abi.encodeWithSelector(req.callbackFunctionId
, cbRequestId, result));
        return success;
      }
      return true;
    }
```

Status

The `PreCoordinator` contract is developed by the Chainlink team and maintained by Chainlink nodes. The DMM Protocol team does not manage the Chainlink contracts directly. Chainlink serves as third parties to collect and update data for the contract. The minimum response limit is a well-designed mechanism of Chainlink's Oracle. When the number of responses reaches the minimum required, the target callback function is immediately executed to update the data to the target contract. This mechanism also prevents data from being updated twice for the same request. It has been working well in a production environment.

4.4.2 Heavy reliance on external oracles

Risk Type	Risk Level	Impact	Status
Potential Risk	Info	Protocol Stability	Discussed

According to the whitepaper of DMM Protocol, the information of backing assets is updated transparently via Chainlink's Oracle. The timeliness and reliability are decided by the authorized nodes of oracle contracts. When there are serious problems with the oracle system, it could have a big impact on the stability of the protocol.

Suggestion

Implementing a reliable oracle on a blockchain is very difficult, and Chainlink provides a relatively reliable solution for on-chain smart contracts. The Chainlink's solution is also particularly suitable for the DMM protocol, considering the need for on-chain representation to off-chain assets.

It is recommended to give simple explanations of the mechanism of oracles to DMM users and provide more information about oracle status and Chainlink nodes. It is also recommended that the development team prepare a contingency plan in case of an oracle failure in advance.

Status

Discussed. Currently, external oracles are the only method to link real-world assets to an on-chain context. Overall stats and health, which can affect administrative control, is made inherently more decentralized by forcing Chainlink nodes to publish the information about the off-chain assets to this protocol. The usage of Chainlink and DMM Explorer (https://explorer.defimoneymarket.com/) is accurately discussed in the white paper, which would give users a nice overview & in-depth view of the ecosystem's health. The project team also has a long-standing relationship with the Chainlink team and is confident of the continuous independence, redundancy, and security of external oracles.

4.4.3 Instability of Ether deposits and lending

Risk Type	Risk Level	Impact	Status
Potential Risk	Medium	Protocol Stability	Discussed

The user deposits Ethers into the protocol in the hope of earning interest. But the normal business process is to convert the cryptocurrency into fiat money, which is ultimately lent out to other users. But as we all know that the price of Ether is unstable and if the price increases more than the originally set interest rate, it is difficult to pay back with fiat currency to users who deposited Ethers originally.

Suggestion

The development team should be aware of this issue already, so only stable coins like DAI and USDC are supported first. But non-stablecoins related business processes are worth discussing further, especially how to ensure that users' original assets can be returned when asset prices have increased a lot compared to before.

Status

This has been discussed and the project team has addressed it when they initially released ETH deposits. Zachary Rynes from the DMM team wrote about this in this article, which gives reasonable and solid solutions. Overall, there are a couple of ways they're mitigating risk in the short-run and long-run that the article discusses or glosses over. For right now, they aren't drawing down ETH deposits. The goal is to grow deposits for now. Thus, their only obligation is to purchase the interest payments, which is negligible. Additionally, they'll weigh the different coins differently as the system grows.

4.5 Other findings

4.5.1 Use of ReentrancyGuard

DMM Protocol introduces `ReentrancyGuard` from the OpenZeppelin library in the implementation of `DmmEther`, `DmmToken`, and `ERC20`, which could help prevent reentrant calls to a function. This is quite a good practice to circumvent the risk of a reentrancy attack, especially considering the recent spate of smart contract security incidents where both Uniswap and Lendf.me suffered huge losses as a result of listing the ERC777 tokens.

4.5.2 Time locks for admin functions

DMM Protocol adopts `DelayedTransaction` for admin functions to add time locks for some important operations. It allows protocol users to react before some operations got executed on-chain, which provides more transparency to community governance. This also increases the security for the stability of the protocol.

5. Conclusion

DMM Protocol is an Ethereum-based and decentralized protocol that allows any holder of an Ethereum-based digital asset to earn interest. After auditing and analyzing the contract code of it, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above. Particularly, SECBIT Labs holds the view that the DMM Protocol project has a high quality of code with a clean structure, good function naming conventions, complete annotations, and well-designed test cases. The entire protocol is designed to be very succinct and efficient. DMM Protocol connects crypto tokens with real-world assets and brings the best of each side to the other one. Crypto users could receive interest generated by real-world assets, which is the first breakthrough in the DeFi world. It also introduces more transparency and security to traditional financial services. On this project, we also saw a balance of centralization and decentralization, a combination of security and convenience. Specifically, it carefully weighed the rights of admins and even introduced delayed transactions for admin operations, which should become a good practice for all DeFi applications. It also offers a more convenient way for normal users to execute gasless transactions, which greatly reduces the difficulty for users to use cryptocurrencies.

Disclaimer

SECBIT smart contract audit service assesses the contract's correctness, security, and performability in code quality, logic design, and potential risks. The report is provided "as is", without any warranties about the code practicability, business model, management system's applicability, and anything related to the contract adaptation. This audit report is not to be taken as an endorsement of the platform, team, company, or investment.

APPENDIX

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the contract's integrity and allow attackers to steal ethers and tokens, or lock ethers inside the contract.
Medium	Damage contract's security under given conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to contract.
Info	Relevant to practice or rationality could possibly bring risks.

SECBIT Labs is devoted to constructing a common-consensus, reliable and ordered blockchain economic entity.

http://www.secbit.io

audit@secbit.io

@secbit_io