Competitive Security Assessment

M3tering

Dec 6th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

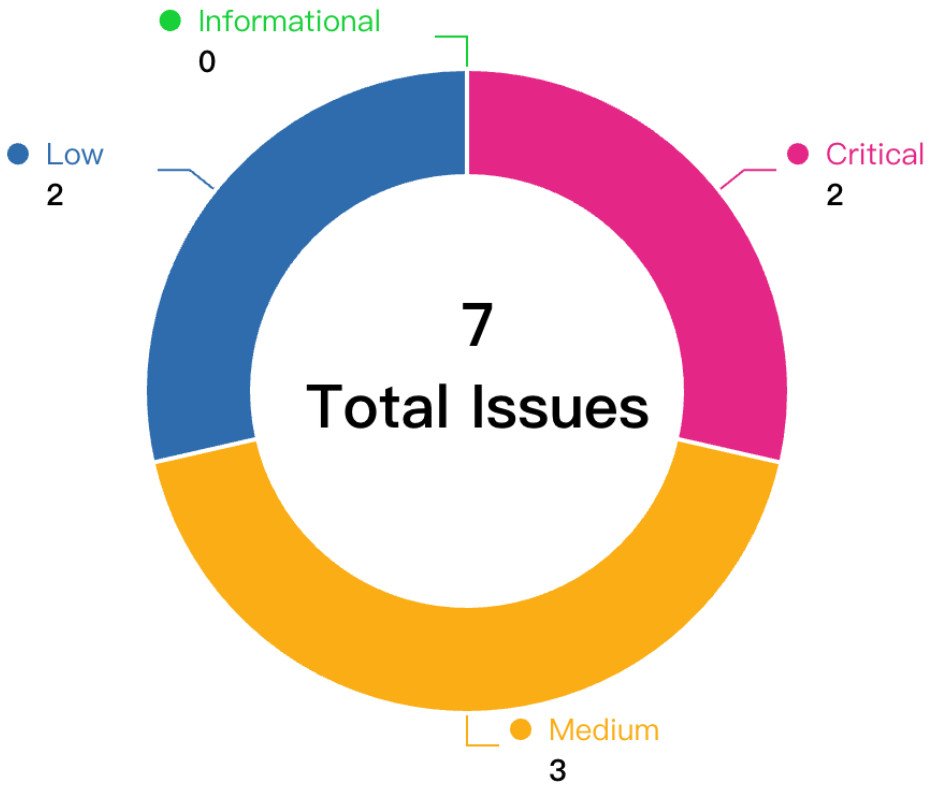
Project Detail

Project Name	M3tering
Platform & Language	Solidity
Codebase	• https://github.com/M3tering/Solaxy • audit commit - 58b5b5ea59bacd181be6f402c6f85f561794df67 • final commit - 58b5b5ea59bacd181be6f402c6f85f561794df67 • https://github.com/M3tering/Protocol/ • audit commit - 04f9040bf27607897e309e1fbec1f678b2d1d3c0 • final commit - 1ff59cbb8878a3c6f2793fce7c9c3ec7daf89615 • https://github.com/M3tering/Protocol-V0 • audit commit - 11e2b6314e1121f3f5350dfc13c6354f2644a46d • final commit - 11e2b6314e1121f3f5350dfc13c6354f2644a46d • https://github.com/M3tering/Protocol-V1 • audit commit - 8fe0ccc666659a84f51024af696e0c8d6dbfa3bb • final commit - b6234f17d1373314151add86d713b133ea7efccc • https://github.com/M3tering/Protocol-V2 • audit commit - f3848c8f09a8808d573f795a865114540282e9f5 • final commit - f3848c8f09a8808d573f795a865114540282e9f5 • https://github.com/M3tering/M3ters • audit commit - 35652a35bb1f92bd6ac2aac5b11913e909eccf07 • final commit - 35652a35bb1f92bd6ac2aac5b11913e909eccf07
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./Solaxy/src/Solaxy.sol	88ce066eb38d35792fde4d74456380760364708108 66b0635f115f47b64e3166
./Protocol-ABC/contracts/Protocol.sol	0ffb05dd6d4e69fe3fd362b09b1f5dea98874907769cf00c768b5c4a7103e6c6
./Protocol-ABC/contracts/interfaces/IProtocol.sol	ddd00f1a32d70bf05f38a4f11cb97e64e3940f6126cf87a0f65358bae04d07b4
./M3ters/contracts/XRC721.sol	1cf1b154125a177d29a9fdc982b4a66db47d9fb4de5b8f1895d89cf7dca93121
./M3ters/contracts/M3ter.sol	e40d2c7c801bcced7decb2eb908295c2a0895ede059d07d794fb3e0ccfdb4ec5
./Protocol-V2/Strategy_V2.sol	3a20364e558ae7c60198c19c4edbc226bc175c2e93bc6b7842003fb354eca238
./Solaxy/src/interfaces/ISolaxy.sol	db0af9de3f529b3ba2097d29952e7e86c011a41c09a5bf803320414e343371fe
./Solaxy/src/XRC20.sol	af89d6c8c31822144d5ac57dbbdb0fa7cda606f682a6f34a6bb4fe0d90cde820
./Protocol-V2/interfaces/IMimo.sol	5f050be3087754614b3332bf5d32ca5061a98730c68b122290387f9fc2fa1c38
./M3ters/contracts/interfaces/IM3ter.sol	095d03faf59e49490d7f782660a9331e307330d3187bae4aeaa8885879a736ed
./Protocol-V1/Strategy_V1.sol	94b5a511818243e8d54a44d5a475fd905878a8575f57cc95a325fe6aad259102
./Protocol-V0/Strategy_V0.sol	2a3856c2b0ab8d6a9769d7c32dfc4fe5fcdf9c8aaad810e241f216e28a66fffa
./Protocol-V2/interfaces/ISolaxy.sol	d1744935481761192aa47133a6d6dcadedd971e0bf260edf6f31ce03c606875a

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
M3T-1	wrong `DAI` and token address	Logical	Critical	Mitigated	zigzag, toffee
M3T-2	Call library contracts error	Logical	Critical	Fixed	zigzag
M3T-3	`Protocol.pay()` can pay to the wrong `revenues` owner	Logical	Critical	Declined	toffee
M3T-4	`Solaxy` very high redeem fee and withdraw fee not mentioned in the whitepaper	Privilege Related	Medium	Acknowledged	toffee

M3T-5	`Protocol::_setStrategyLib()` lacks access control	Logical	Medium	Fixed	ethprinter, zigzag
M3T-6	Strategy_V2 should be a library contracts	Logical	Medium	Fixed	zigzag
M3T-7	`Protocol`::`_setFeeAddress()` Missing Zero Address Check	Logical	Low	Fixed	ethprinter
M3T-8	`_setTariff` function unsafe cast to uint248 from uint256	Integer Overflow and Underflow	Low	Fixed	toffee

M3T-1:wrong `DAI` and token address

Category	Severity	Client Response	Contributor
Logical	Critical	Mitigated	zigzag, toffee

Code Reference

- code/Protocol-V1/Strategy_V1.sol#L13
- code/Protocol-ABC/contracts/Protocol.sol#L17
- code/Solaxy/src/Solaxy.sol#L17
- code/Protocol-V2/Strategy_V2.sol#L17
- code/Protocol-ABC/contracts/Protocol.sol#L20

```
13:ISolaxy SLX = ISolaxy(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b); // TODO: add Solaxy address

17:IERC20 public constant DAI =

17:ERC20 public constant DAI = ERC20(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b);

17:IERC20 SLX = IERC20(0x147CdAe2BF7e809b9789aD0765899c06B361C5cE); // solaxy

20:IERC721(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b); // TODO: M3ter Address
```

Description

zigzag : https://iotexscan.io/search?q=DAI
In this web page, there are two Dai token. In M3tering contracts, the DAI's address is `0x1cbad85aa66ff3c12dc84c5881886eeb29c1bb9b` and the price is $0.1. The most critical thing is that the total supply value of the token is very small, less than 1k.The price of another token is close to ¥1, and the total supply is much larger than `0x1cba` . Obviously, `0x1cba` is more likely to be affected by the market and cause price fluctuations, which is contrary to the original intention of the protocol design.
toffee : the token address is wrong (while some of them have `TODO` comment, still worth point out)

- `0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b` is DAI at https://iotexscan.io/token/io1rjadsk4xdleuztwgf3vgrzrwav5urwumaakt0w#token_transfer
- `0x147CdAe2BF7e809b9789aD0765899c06B361C5cE` is MimoV2Router02NoReferral at https://iotexscan.io/address/io1z37d4c4l06qfh9uf45rktzvuq6ekr3ww6tju03

```
    IERC721 public constant M3ter =
        IERC721(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b); // TODO: M3ter Address //audit:the addr
ess is wrong
======
        ISolaxy SLX = ISolaxy(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b); // TODO: add Solaxy addre
ss //audit:the address is wrong
        if (!IERC20(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b).approve(address(SLX), revenueAmoun
t)) revert Unauthorized();

======

        IERC20 SLX = IERC20(0x147CdAe2BF7e809b9789aD0765899c06B361C5cE); // solaxy //audit:the addre
ss is wrong
        IMimo MIMO = IMimo(0x147CdAe2BF7e809b9789aD0765899c06B361C5cE); // router
        IERC20 DAI = IERC20(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b); // ioDAI
```

commented with `audit:`

Recommendation

zigzag : Refactor the Dai's address to 0x62a9d987cbf4c45a550deed5b57b200d7a319632.
toffee : correct the token and contract address

Client Response

Mitigated. Yeah this is acknowledged. however the supply of DAI can be increased as users bridge in more DAI from Ethereum L1 mainnet to IoTeX mainnet after the protocol is in use.

Also, there is currently a proposal to develop a stablecoin native to the IoteX ecosystem. https://community.iotex.io/t/proposal-for-the-development-of-a-native-iotex-stablecoin/11121 If the proposal is successfully implemented, we might ditch bridged DAI altogether and use the natively supported stablecoin instead.

Acknowledged, once the M3ter and Solaxy contracts are deployed, we would have the correct addresses for the protocol contrsac code and this would be fixed as well. >

the above are just placeholder addresses

M3T-2:Call library contracts error

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	zigzag

Code Reference

- code/Protocol-ABC/contracts/interfaces/IProtocol.sol#L4
- code/Protocol-V0/Strategy_V0.sol#L7
- code/Protocol-ABC/contracts/Protocol.sol#L84

```
4:interface ClaimStrategy {

7:library Strategy_V0 {

84:ClaimStrategy(libAddress).claim(revenueAmount, receiver, outputAmount, deadline);
```

Description

zigzag :

```
    function claim(address libAddress, address receiver, uint256 outputAmount, uint256 deadline) ext
ernal whenNotPaused {
        .......
        ClaimStrategy(libAddress).claim(revenueAmount, receiver, outputAmount, deadline);
        emit Claim(msg.sender, revenueAmount, block.timestamp);
    }
library Strategy_V0 {
    function claim(uint256 revenueAmount, address receiver, uint256 outputAmount, uint256 deadline)
public {
        if (!IERC20(0x1CbAd85Aa66Ff3C12dc84C5881886EEB29C1bb9b).transfer(receiver, revenueAmount)) r
evert TransferError();
    }
}
```

The design is that the user different `libaddress` to perform different ClaimStrategy operations. `ClaimStrategy(libAddress).claim(...)` is a call. Hence, The ERC20's transfer call of `msg.sender` is `Strategy_V0` . Obviously, the Strategy_V0 lack of DAI. The normal design is Protocol delegatedcall Strategy_V0's logic. The most important thing is that the library contracts is no status. See this Link below: https://sepolia.etherscan.io/address/0x1976a59014660b27593dbf3e08cf369b147fdcde#writeContract It means Not allowed to be called directly. So it can cause DOS.

Recommendation

zigzag : 1. delete `mapping(address => bool) public strategyLib;`

```
-     mapping(address => bool) public strategyLib;
+     enum Option{ Zero, One, Two }
  function claim(Option _option, address receiver, uint256 outputAmount, uint256 deadline) external w
henNotPaused {
-         if (strategyLib[libAddress] == false) revert BadStrategy();
          uint256 revenueAmount = revenues[msg.sender];
          if (revenueAmount < 1) revert InputIsZero();
          revenues[msg.sender] = 0;

+         if(option == Zero){
+             Strategy_V0.claim(revenueAmount, receiver, outputAmount, deadline);
+         }else{
+         ......
+         }
          emit Claim(msg.sender, revenueAmount, block.timestamp);
      }
```

2. Using delegatecall

```
      function claim(address libAddress, address receiver, uint256 outputAmount, uint256 deadline) ext
ernal whenNotPaused {
          if (strategyLib[libAddress] == false) revert BadStrategy();
          uint256 revenueAmount = revenues[msg.sender];
          if (revenueAmount < 1) revert InputIsZero();
          revenues[msg.sender] = 0;

+         (bool success, bytes memory data) = libAddress.delegatecall(
+             abi.encodeWithSignature("claim(uint256, address, uint256t, uint256) ", evenueAmount, re
ceiver, outputAmount, deadline)
+        );
+       reruire(success, "Call failed");
          emit Claim(msg.sender, revenueAmount, block.timestamp);
      }
```

`libAddress` need more access control.

Client Response

Fixed.100% agree with this, I only noticed this shortly after the contest was scheduled to begin

I have implemented a fix here

https://github.com/M3tering/Protocol/commit/d40e10c3febd38753d8548f300810f3c95499a7a, and also made follow up

improvements here https://github.com/M3tering/Protocol/commit/25d2b3f44cb3d2207d31acd15fe76ba96b44288a

Summary

I ditched the use of `DELEGATECALL` operation via the `library` and implemented strategy as a smart `contract` instead. I had difficulty using the library to execute state changing function. So instead, strategy can be implemented as a contract.

In the updated implementation approves the strategy contract to only spend `msg.sender` DAI revenue. it also allows the strategy contract to receive encoded bytes as input parameter. I also continue to use a mapping to curate strategies allowed to be executed in the claim function.

Here is a truncated, view of the function code.

```
    function claim(
        address strategyAddress,
        bytes calldata data
    ) external whenNotPaused {
        if (strategy[strategyAddress] == false) revert BadStrategy();
        uint256 revenueAmount = revenues[msg.sender];
        ...
        if (!DAI.approve(strategyAddress, revenueAmount)) revert Unauthorized();
        IStrategy(strategyAddress).claim(revenueAmount, data);
        ...
    }
```

the claim function implements balance checks before and after the strategy contract is executed and reverts if unexpected state change is detected on the protocol DAI balance. I believe these check would be sufficient to prevent reentrancy attacks as any illegal state changes would trigger a revert.

```

    function claim(
        address strategyAddress,
        bytes calldata data
    ) external whenNotPaused {
        ...
        uint256 preBalance = DAI.balanceOf(address(this));
        ...
        uint256 postBalance = DAI.balanceOf(address(this));
        if (postBalance != preBalance - revenueAmount) revert TransferError();
        emit Claim(msg.sender, revenueAmount, block.timestamp);
    }
```

Would really appropriate if this new implementation was reviewed carefully and giving priority to find any more bugs hiding in this implementation pattern.

M3T-3: `Protocol.pay()` can pay to the wrong `revenues` owner

Category	Severity	Client Response	Contributor
Logical	Critical	Declined	toffee

Code Reference

- code/M3ters/contracts/M3ter.sol#L26
- code/Protocol-ABC/contracts/Protocol.sol#L61-L67
- code/Protocol-ABC/contracts/Protocol.sol#L106

```
26:function mint() external onlyRole(REGISTRAR_ROLE) whenNotPaused {

61:function pay(uint256 tokenId, uint256 amount) external whenNotPaused {
62:         if (!DAI.transferFrom(msg.sender, address(this), amount))
63:             revert TransferError();
64:
65:         uint256 fee = (amount * 3) / 1000;
66:         revenues[feeAddress] += fee;
67:         revenues[_ownerOf(tokenId)] += amount - fee;

106:return M3ter.ownerOf(tokenId);
```

Description

toffee : In the `Protocol.pay()` function, the msg.sender send x DAI to `Protocol` contract, and amount less 0.3% of the fee is credited to `tokenId` owner via `revenues[_ownerOf(tokenId)] += amount - fee;`

the call stack trace is as below `Protocol::_ownerOf()` -> `M3ter.ownerOf(tokenId)` -> `ERC721.ownerOf(tokenId)` -> `ERC721._owners[tokenId]`

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol#L172

However the `owner` is not always the user.

Let's take a look on the `M3ter.mint()` function, it is with `onlyRole(REGISTRAR_ROLE)` modifier hence makes it only callable by the contract creator itself. Within the `M3ter` contract there is only `_register` function updates its customized `tokenRegistry` and `keyDirectory` state, **NOT** the common 721 `_owners` state.

I assume the design is to have the REGISTRAR_ROLE to call `_register` to mark the NFT ID to owner address relationship, but the actual owner on chain would still be contract creator `REGISTRAR_ROLE` unless the contract creator transfer it one by one to the actual `owner` address immediately (which would then update the 721 `_owners` state).

Since the `transfer` or `safeTransfer` is not in the `_register` or `mint` function, there can be cases where the

transfer failed after mint completes, and the `tokenRegistry` and `keyDirectory` states will be out of sync of `ERC721._owners` state.

A detailed failing scenario would be if the passed in `publicKey` is a contract and it does not implement `IERC721Receiver.onERC721Received`, the safe transfer will fail. And in that case the revenue will always be added to `REGISTRAR_ROLE` incorrectly.

Recommendation

toffee : in `M3ter.mint` function mint to the user directly with a whitelist for the allowed user list.

Client Response

Declined. The M3ter contract mints NFTs that represent physical smart metering devices IRL. these devices contain cryptographic secure elements used to sign energy consumption data they measure. the purpose of tokenRegistry is to map an NFT id to the cryptographic PublicKey for a device (bytes 32 key for ED25519 digital signature scheme).
Accepted by Secure3.

M3T-4: `Solaxy` very high redeem fee and withdraw fee not mentioned in the whitepaper

Category	Severity	Client Response	Contributor
Privilege Related	Medium	Acknowledged	toffee

Code Reference

- code/Solaxy/src/Solaxy.sol#L250
- code/Solaxy/src/Solaxy.sol#L278

```
250:fee = withdrawnShares.mul(ud60x18(0.359e18)).intoUint256();

278:UD60x18 _fee = ud60x18(shares).mul(ud60x18(0.264e18));
```

Description

toffee : The `Solaxy.computeWithdraw()` charges **35.9%** of the fee when withdraw DAI The `Solaxy.computeRedeem()` charges **26.4%** of the fee when redeem shares

```
        fee = withdrawnShares.mul(ud60x18(0.359e18)).intoUint256();

        UD60x18 _fee = ud60x18(shares).mul(ud60x18(0.264e18));
```

these percentage is high and not mentioned in the public doc https://m3tering.whynotswitch.com/token-economics/mint-and-distribution

Recommendation

toffee : make the doc and code consistent

Client Response

Acknowledged. My bad for failing to document the fee situation actually only a 7% fee is applied during withdrawal or redemption. However this is applied to the DAI reserve and not the SLX. It's a bit tricky.

when a user withdraws or redeems, 73.6% of their SLX is burnt and the length of the linear slope of the bonding curve is reduced to only 26.4% of the original slope. However, the area under the burnt section of the slope is ~93% of the total area under the curve (DAI reserve).

Here is an visualization of this effect.  newplot(16) in the above visualization we assume

- user owns 100_000 SLX (100% of supply)
- slope is 0.00125

Redeeming:

- current price = $100,000 * 0.00125$ = 125
- total area under slope = $\frac{1}{2} * 100,000 * 125$ = 6,250,000
- price after burn = $26,400 * 0.00125$ = 33
- remaining area under fee slope = $\frac{1}{2} * 26,400 * 33$ = 435,600
- fee percentage actually = $435,600 / 6,250,000$ = 0.069696 or ~7%

the `withdraw` function also implements an ~7% fee; but because the DAI to be received is specified as input to the function, the calculation varies a bit. A more simplified form of that calculation is executed within the `computeWithdraw` function to save gas and accurate up 0.002 DAI
fixed the docs with info about the fee, you can read about it here https://m3tering.whynotswitch.com/token-economics/burn-and-exit-fee

M3T-5: `Protocol::_setStrategyLib()` lacks access control

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	ethprinter, zigzag

Code Reference

- code/Protocol-ABC/contracts/Protocol.sol#L57
- code/Protocol-ABC/contracts/Protocol.sol#L57-L59
- code/Solaxy/src/Solaxy.sol#L315

```
57:function _setStrategyLib(address libAddress, bool state) external {

57:function _setStrategyLib(address libAddress, bool state) external {
58:         strategyLib[libAddress] = state;
59:     }

315:if (!DAI.transfer(receiver, assets)) revert TransferError();
```

Description

ethprinter : `Protocol`::`_setStrategyLib()` lacks access control, it allows anyone to set the state of `libAddress`, which could result in the attack of `claim()` function, it first check whether `libAddress` is valid in : `if (strategyLib[libAddress] == false) revert BadStrategy();` and make an external call in : `ClaimStrategy(libAddress).claim(revenueAmount, receiver, outputAmount, deadline);` if an attacker can modify `strategyLib[]` , they can easy bypass the check, which could result in reentrancy attack.

zigzag :

```
    function _setStrategyLib(address libAddress, bool state) external {
        strategyLib[libAddress] = state;
    }
```

Everyone can set the `strategyLib[libAddress]` to true , also can set it false.

```
  function claim(address libAddress, address receiver, uint256 outputAmount, uint256 deadline) extern
al whenNotPaused {
        if (strategyLib[libAddress] == false) revert BadStrategy();
        .....
```

So attacker can monitor the contracts, when someone call _setStrategyLib, it can reset again.Thus causing the claim function dos.

Recommendation

ethprinter : add access control of `Protocol`::`_setStrategyLib()`, only allow `owner` to change the state.

zigzag :

```
function _setStrategyLib(address libAddress, bool state) external onlyRole(SetStrategyLib_ROLE) {
        strategyLib[libAddress] = state;
    }
```

Client Response

Fixed. 100% agree

Implemented a fix for this by creating a spacial role CURATOR_ROLE allowed to curate the mapping of strategies. Without this role, the transaction is expected to always revert. you can find the commit fixing this here. https://github.com/M3tering/Protocol/commit/107335d5611cbc6cc4c6b96d83b9f3ca721a92a7

M3T-6:Strategy_V2 should be a library contracts

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	zigzag

Code Reference

- code/Protocol-V2/Strategy_V2.sol#L8

```
8:contract Strategy_V2 {
```

Description

zigzag : Strategy_V2 should be a library contract. Obviously, Strategy_V2 doesn't have DAI token. `MIMO.swapExactTokensForTokensSupportingFeeOnTransferTokens` will revert.

Recommendation

zigzag :

```
library Strategy_V2 {
```

Client Response

Fixed. As mentioned in https://github.com/Secure3Audit/findings_M3tering/issues/5#issuecomment-1837503748, The library fails to execute and contracts was used instead.

you can see more explanation in https://github.com/Secure3Audit/findings_M3tering/issues/5#issuecomment-1837503748 and the pull request at https://github.com/M3tering/Protocol/commit/d40e10c3febd38753d8548f300810f3c95499a7a

M3T-7: `Protocol` :: `_setFeeAddress()` Missing Zero Address Check

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ethprinter

Code Reference

- code/Protocol-ABC/contracts/Protocol.sol#L45-L49

```
45:function _setFeeAddress(
46:         address otherAddress
47:     ) external onlyRole(DEFAULT_ADMIN_ROLE) {
48:         feeAddress = otherAddress;
49:     }
```

Description

ethprinter : This Function is lack of zero address check in important operation, which may cause some unexpected result.Also If the feeAddress is set to address(0), it may cause damage to the protocol.

Recommendation

ethprinter :

```
function _setFeeAddress(
address otherAddress
) external onlyRole(DEFAULT_ADMIN_ROLE) {
if (feeAddress == address(0)) revert ZeroAddress();
feeAddress = otherAddress;
}
```

Client Response

Fixed.valid concern, has been rectified as suggested

```
function _setFeeAddress(
    address otherAddress
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (otherAddress == address(0)) revert ZeroAddress();
    feeAddress = otherAddress;
}
```

you can find the commit with this fix here.
https://github.com/M3tering/Protocol/commit/2ae2b99c7718ea03ff6a443023657a3de2d466f8

M3T-8: `_setTariff` function unsafe cast to uint248 from uint256

Category	Severity	Client Response	Contributor
Integer Overflow and Underflow	Low	Fixed	toffee

Code Reference

- code/Protocol-ABC/contracts/Protocol.sol#L54

```
54:states[tokenId].tariff = uint248(tariff);
```

Description

toffee : In the `Protocol::_setTariff(uint256 tokenId, uint256 tariff)` function the passed in parameter `tariff` is downcast to `state.tariff` As the `State struct` tries to pact two fields into one slot, `tariff` is defined as `uint248` hence `uint248(tariff)` can potentially overflow

```
    struct State {
        // int:tariff = float:$$ *10^3
        uint248 tariff;
        bool state;
    }
```

Recommendation

toffee : use the OpenZeppelin SafeCast utilities or check value for possible overflow
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeCast.sol#L50

Client Response

Fixed. adding the safe-casting lib would propbaly cost more gas than saved in storage usage of uint248 in the first place. reverted back to using uin256 for the tariff in State struct. you can view the commit here
https://github.com/M3tering/Protocol/commit/1ff59cbb8878a3c6f2793fce7c9c3ec7daf89615

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.