Dezy Finance

Security Testing and Assessment

February 24, 2023

Prepared for:
Harald Lang, Miroslav Nedelchev and Biondi Tan
Dezy Finance

PWNED NO MORE LABS

Table of Content

Summary 3

Disclaimer 5

Project Overview 6

Tests 8

Tested Invariants and Properties 8

Code Coverage 9

Key Findings and Recommendations 12

S3: Missing Check on Deposit Token 12

S3: Emergency Withdraw Always Fails 14

S3: Claiming MTA Rewards in Tokens May Fail 16

S4: Missing check on msg.sender in onERC721Received() 18

S7: Fund Loss via S7Strategy::repayLoanAndWithdraw() 19

S7: Incorrect Parameter of _mintProxy() in the S7Strategy::onERC721Received() 21

S7: Accounting Errors in the S7Strategy::withdrawToken() 23

S9: Insufficient Liquidity in Uniswap stETH-ETH V3 Pool 25

Fix Log 27

Appendix 28

Summary

Overview

From January 10, 2023, to February 20, 2023, Dezy Finance engaged Pwned No More Labs to assess the security of its five yield farming strategies in the following GitHub repositories:

- S1: https://github.com/DezyDefi/dezy-strategy-1
  - Commit 3a46838b4e1e352759850a640649acb7c51aaf4f
- S3: https://github.com/DezyDefi/dezy-strategy-3
  - Commit a723ae68c424ff579544a4a3f1133d17ec1e1c2e
- S4: https://github.com/DezyDefi/dezy-strategy-4
  - Commit ec1c7440c5c688fecfb97a198b95955c557d277e
- S7: https://github.com/DezyDefi/dezy-strategy-7
  - Commit 3538c0b5f9210388000a6432cf822e318c6519e6
- S9: https://github.com/DezyDefi/dezy-strategy-9
  - Commit fca5e9ffe403492589c2db9073cc887b9996fb26

A team of two security researchers conducted a security review of the client-provided source code and provided a set of fuzzing tests that checked all the business logic implemented by the client and covered more than 90% of lines of code (from the security point of view, it is not necessary to cover most view and pure functions that cannot be used to break any business logic). We provide the tests, bug findings, and detailed coverage information in subsequent sections of this report.

Project Scope

We reviewed and tested not only the smart contracts that implement the yield farming strategies but also their interactions with external contracts on the live blockchain network.

There are other security-critical components of Dezy Finance, such as off-chain services and the web front end, that is not included in the scope of this smart contract security assessment. We recommend a further review of those components.

Summary of Findings

Severity	# of Findings
High	2
Medium	4
Low	2
Informational	0
Total	8

Throughout the engagement, we identified 8 issues, ranging in severity from low to high:

- Issue related to user fund being frozen in S3
- Issue related to nonfunctional emergency withdraw in S3
- Issue related to failure in claiming reward in S3
- Issue related to potentially losing liquidity position in Uniswap v3 in S4
- Issue related to user fund being directly stolen in S7
- Issue related to potential denial of service in S7
- Issue related to potential failure when withdrawing user fund in S7
- Issue related to user fund loss when investing with low liquidity in S9

We confirm that all the issues mentioned in this report have been acknowledged and fixed by Dezy Finance (see Fix Log).

Disclaimer

This testing report should not be used as investment advice.

Pwned No More Labs uses an automatic testing technique to test smart contracts' security properties and business logic rapidly and continuously. However, we do not provide any guarantees on eliminating all possible security issues. The technique has its limitations: for example, it may not generate a random edge case that violates an invariant during the allotted time. Its use is also limited by time and resource constraints.

Unlike time-boxed security assessment, Pwned No More Labs advises continuing to create and update security tests throughout the project’s lifetime. In addition, Pwned No More Labs recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

Project Overview

S1

S1 contains 3 sub-strategies for yield farming:

- **S1VesperFinanceDAI** receives your ETH tokens or any whitelisted ERC20 token (WBTC, USDC, USDT, DAI, and XSGD), swaps them into DAI tokens, and invests the DAI tokens into the Vesper Finance DAI pool for generating rewards.
- **S1VesperFinanceETH** receives your ETH tokens or any whitelisted ERC20 token (WBTC, USDC, USDT, DAI, and XSGD) which will be swapped into ETH tokens, and invests the ETH tokens into the Vesper Finance ETH pool for generating rewards.
- **S1mStableUSDC** receives your ETH tokens or any whitelisted ERC20 token (WBTC, USDC, USDT, DAI, and XSGD), swaps them into USDC tokens, and invests the USDC tokens into the mStable vault for generating rewards.

S3

S3 contains 4 sub-strategies for yield farming:

- **S3ETHAavemStableUSDC** receives your ETH tokens and deposits them into Aave Lending Pool v2. The strategy optionally uses a part of the borrowing power to borrow the USDC tokens from Aave. The USDC tokens are then invested into the mStable vault for generating rewards.
- **S3ETHAaveVesperFinanceDAI** receives your ETH tokens and deposits them into Aave Lending Pool v2. The strategy optionally uses a part of the borrowing power to borrow the DAI tokens from Aave. The DAI tokens are then invested into the Vesper Finance DAI pool for generating rewards.
- **S3TokenAavemStableUSDC** receives any whitelisted tokens (WBTC only now) and deposits them into Aave Lending Pool v2. The strategy optionally uses a part of the borrowing power to borrow the USDC tokens from Aave. The USDC tokens are then invested into the mStable vault for generating rewards.
- **S3TokenAaveVesperFinanceDAI** receives any whitelisted tokens (WBTC only now) and deposits them into Aave Lending Pool v2. The strategy optionally uses a part of the borrowing power to borrow the DAI tokens from Aave. The DAI tokens are then invested into the Vesper Finance DAI pool for generating rewards.

S4

Strategy S4 receives a certain number of USDC or ETH tokens from you and then uses the tokens to provide liquidity for Uniswap v2 or v3 pools for generating rewards. The pools covered by our testing include

- Uniswap USDC-XSGD v3 pool
- Uniswap USDC-USDT v3 pool
- Uniswap USDC-USDT v2 pool

S7

Strategy S7 receives a certain number of ERC20 or ETH tokens from you and then deposits the tokens into Aave for taking a loan. The loan is used to provide liquidity for Uniswap v2 or v3 pools for generating rewards. The pools covered by our testing include

- Uniswap USDC-USDT v3 pool
- Uniswap USDC-USDT v2 pool

S9

Strategy S9 receives a certain number of ERC20 or ETH tokens from you, swaps the tokens into the ones wanted by the Ribbon vault and deposits the output tokens into the vault for generating rewards. The vaults covered by our testing include

- Ribbon USDC vault
- Ribbon stETH vault

We note two important points in the protocol implementation:

- For all the strategies, a separate smart contract (called “proxy”) will be created for every user to process user funds and interact with the external contracts. The isolation largely increases the security of user funds. Nevertheless, one of our testing focus is still to check if user fund deposited into any strategy can be frozen or stolen.
- The interfaces of all five strategies are generally the same, which enables a standardized set of tests for all of them.

Tests

Tested Invariants and Properties

We relied on the PNM engine that used a smart fuzzing approach to test the following 10 invariants and properties of the smart contracts of five yielding strategies. In total, we implemented 42 invariant tests and co-implemented 174 property tests with the client.

ID	Invariant/Property Description	Found Bug(s)
Applied to all the five strategies		
01	A strategy can receive user deposits in any supported token and transfer them correctly into corresponding external contracts.	Passed
02	A user can withdraw almost all the deposits in any supported token at any time from a strategy (we use 97%/98% in our testing according to the fee settings used by the client).	S3: Missing Check on Deposit Token S7: Fund Loss via S7Strategy::repayLoanAndWithdraw S9: Insufficient Liquidity in Uniswap stETH-ETH V3 Pool
03	A user can correctly claim the rewards in any supported token from a strategy according to the deposit amount and past time.	S3: Claiming MTA Rewards in Tokens May Fail
04	A user can invoke the emergency withdrawal function to take out the deposits or any other owned tokens from a strategy.	S3: Emergency Withdraw Always Fails
05	A user cannot get more than 5% profit via a strategy in one block.	Passed
Applied to S3 and S7 lending and borrowing tokens on Aave		
06	A user can add collateral to repay its Aave position.	Passed
07	A user can always withdraw the unlocked collateral from Aave.	Passed
Applied to S4 and S7 providing liquidity for Uniswap v3		

08	A user can withdraw owned position NFT from S4/S7.	S4: Missing check on msg.sender in onERC721Received()
09	A user can update Uniswap V3 position after depositing tokens into S4/S7.	Passed
Applied to S9 staking tokens in the Ribbon vault		
10	A user can unstake the tokens locked in a Ribbon vault.	Passed

Code Coverage

To evaluate the effectiveness of our fuzz testing, we list the code coverage of the smart contracts for the five strategies. Only the smart contracts being used and related to the yielding strategies (basically the strategy contracts and proxy contracts) are counted below.

S1

Smart Contract Path	Line Coverage
contracts/S1TrueFi.sol	N/A*
contracts/S1VesperFinanceDAI.sol	100%
contracts/S1VesperFinanceETH.sol	100%
contracts/S1mStableUSDC.sol	100%
contracts/proxies/S1TrueFiProxy.sol	N/A*
contracts/proxies/S1VesperFinanceDAIProxy.sol	100%
contracts/proxies/S1VesperFinanceETHProxy.sol	100%
contracts/proxies/S1mStableUSDCProxy.sol	100%

* The smart contract related to TrueFi is deprecated.

S3

Smart Contract Path	Line Coverage
contracts/S3ETHAaveVesperFinanceDAI.sol	93%

contracts/S3ETHAavemStableUSDC.sol	94%
contracts/S3TokenAaveVesperFinanceDAI.sol	94%
contracts/S3TokenAavemStableUSDC.sol	93%
contracts/proxies/S3ETHAaveVesperFinanceDAIProxy.sol	96%
contracts/proxies/S3ETHAavemStableUSDCProxy.sol	99%
contracts/proxies/S3TokenAaveVesperFinanceDAIProxy.sol	99%
contracts/proxies/S3TokenAavemStableUSDCProxy.sol	99%

S4

Smart Contract Path	Line Coverage
contracts/proxies/S4Proxy.sol	96%
contracts/S4Strategy.sol	92%

S7

Smart Contract Path	Line Coverage
contracts/S7Read.sol	90%
contracts/S7Strategy.sol	93%
contracts/proxies/S7Proxy.sol	91%

S9

Smart Contract Path	Line Coverage
contracts/S9Strategy.sol	96%
contracts/proxies/S9Proxy.sol	100%

contracts/helpers/StethSwapHelper.sol	100%

Key Findings and Recommendations

S3: Missing Check on Deposit Token

Severity: **High**

Difficulty: **High**

Description

The smart contracts `S3TokenAavemStableUSDC` and `S3TokenAaveVesperFinanceDAI` should only allow users to deposit the token specified by the state variable `collateral`. In the developer's settings, the token is WBTC. However, the `depositToken` function does not check whether the argument `_token` equals `collateral` or not.

The `depositToken` function of the `S3TokenAaveVesperFinanceDAI` contract

```
function depositToken(address _token, uint256 _amount, uint8 _borrowPercentage,
bool _borrowAndDeposit) external {
    require(IFees(feesAddress).depositStatus(strategyIndex), "ERR:
DEPOSITS_STOPPED");
    if (_borrowAndDeposit) {

require(IS3Admin(s3Admin).whitelistedAaveBorrowPercAmounts(_borrowPercentage),
"ERROR: INVALID_BORROW_PERC");
    }
    IERC20(_token).transferFrom(msg.sender, address(this), _amount);

     if (depositors[msg.sender] == address(0)) {
```

Impact

Both `S3TokenAavemStableUSDC` and `S3TokenAaveVesperFinanceDAI` can only transfer out the collateral token. If a user mistakenly deposits any ERC20 token that is not the collateral like USDC, the fund will be permanently locked in the Aave pool.

Failed Invariant

```
// Targeted strategy: S3WBTCAaveVesperFinanceDAI
function invariantDepositWithdrawalFailure() public override {
    address feeToken = address(0);
    vm.prank(user);
    s3.withdraw(100, feeToken);
    require(WBTC.balanceOf(user) >= (USER_DEPOSIT_AMOUNT * 97) / 100);
}
```

Recommendation

Remove the argument `_token`. Use `collateral` directly.

S3: Emergency Withdraw Always Fails

Severity: **Low**

Difficulty: **Low**

Description

The `emergencyWithdraw` function in the smart contracts `S3ETHAavemStableUSDC`, `S3TokenAavemStableUSDC`, `S3TokenAaveVesperFinanceDAI` and `S3TokenAaveVesperFinanceDAI` is not working. Our tests show that calling the emergencyWithdraw function can never withdraw the aWBTC tokens or mUSD saving vault shares owned by the proxy.

The `emergencyWithdraw` function of the `S3TokenAavemStableUSDCProxy` contract

```
    function emergencyWithdraw(address _token, address _depositor)
        external
        onlyDeployer
    {
        IERC20(_token).transfer(
            _depositor,
            IERC20(_token).balanceOf(address(this))
        );
    }
```

We use the `emergencyWithdraw` function in the `S3TokenAavemStableUSDCProxy` contract as an example. The function tries to transfer out all the tokens owned by the proxy, which cannot succeed because of the Aave's debt.

Impact

The `emergencyWithdraw` is not functioning: it cannot help withdraw any user fund.

Failed Invariant

```
// Targeted strategy: S3WBTCAaveVesperFinanceDAI
function invariantEmergencyWithdrawAWBTC() public {
    uint256 depositorBalanceBefore = IERC20(ADDRESS_aWBTC).balanceOf(
        depositor
    );

    if (depositorBalanceBefore > 0) {
        uint256 userBalanceBefore = IERC20(ADDRESS_aWBTC).balanceOf(user);

        vm.startPrank(user);
        s3.emergencyWithdraw(ADDRESS_aWBTC);
        vm.stopPrank();

        uint256 depositorBalanceAfter = IERC20(ADDRESS_aWBTC).balanceOf(
            depositor
        );
        uint256 userBalanceAfter = IERC20(ADDRESS_aWBTC).balanceOf(user);

        require(
            depositorBalanceAfter == 0,
        );
        require(
            userBalanceAfter == userBalanceBefore + depositorBalanceBefore,
        );
    }
}
```

Recommendation

Add an argument to the `emergencyWithdraw` function to specify the number of tokens to be withdrawn.

S3: Claiming MTA Rewards in Tokens May Fail

Severity: **Low**

Difficulty: **Low**

Description

The `claimInToken` function of the sub-strategies `S3ETHAavemStableUSDC` and `S3TokenAavemStableUSDC` is used to claim the rewards originally in MTA tokens and swap them into the desired token:

The `claimInToken` function of the `S3TokenAavemStableUSDC` contract

```
function claimInToken(address _token, uint256 _amountOutMin) external {
    require(depositors[msg.sender] != address(0), "ERR: INVALID_DEPOSITOR");

    uint256 mtaTokens = IS3Proxy(depositors[msg.sender]).claimToDeployer();
    uint256 tokenAmount = IUniswapConnector(uniswapConnector)
        .swapTokenForToken(
            mtaTokenAddress,
            _token,
            mtaTokens,
            _amountOutMin,
            msg.sender
        );
    emit ClaimAdditionalTokens(msg.sender, mtaTokens, tokenAmount, _token);
}
```

If a user calls the `claimInToken` function with `_token` set to WBTC, then the rewarded MTA tokens are swapped into the WBTC tokens which are then transferred to the user. When the number of MTA tokens is too small, the user cannot take out any WBTC token.

Impact

The yielded reward of the sub-strategies `S3ETHAavemStableUSDC` and `S3TokenAavemStableUSDC` can be permanently lost.

Failed Invariant

```
// Targeted strategy: S3WBTCAaveVesperFinanceDAI
function invariantClaimReward() public {
    skip(60 * 60 * 24 * 30);

    uint256 userWBTCBalanceBefore = IERC20(ADDRESS_WBTC).balanceOf(user);
    vm.prank(user);
    s3.claimInToken(ADDRESS_WBTC, 0);

    uint256 userWBTCBalanceAfter = IERC20(ADDRESS_WBTC).balanceOf(user);
    require(userWBTCBalanceBefore > userWBTCBalanceAfter);
}
```

Recommendation

- Check if the returned `tokenAmount` value is 0.
- Prevent the user from invoking `claimInToken` with very few MTA tokens in the front end.

S4: Missing check on msg.sender in onERC721Received()

Severity: **Medium** Difficulty: **Low**

Description

The `S4Strategy` contract implements the callback function `onERC721Received`, which is called when a Uniswap v3 position NFT is transferred to the contract.

The `onERC721Received` function of the `S4Strategy` contract

```
//hook called when nft is transferred to contract
function onERC721Received(
    address operator,
    address from,
    uint256 tokenId,
    bytes calldata data
) external returns (bytes4) {
    require(_depositStatus());
    require(
        INonfungiblePositionManagerStrategy(nfpm).ownerOf(tokenId) ==
            address(this),
        "S4Strategy: Invalid NFT"
    );
    if(depositors[from]==address(0)){
        _mintProxy(depositors[from]);
    }
    // ...
    v3PositionNft[from][token0][token1][poolFee] = tokenId;
```

The callback function is an external one without any access control. An attacker can invoke this function with the \`from\` value being specified to be any wanted address, and the position information of a victim account recorded in the state variable `v3PositionNft` will be modified.

Impact

A user may lose the current Uniswap v3 position without her control.

Recommendation

Check if the `msg.sender` is the NonFungiblePositionManager contract of Uniswap v3.

S7: Fund Loss via S7Strategy::repayLoanAndWithdraw()

Severity: **High**

Difficulty: **Low**

Description

The `repayLoanAndWithdraw` function is public and can be invoked by anyone:

The `repayLoanAndWithdraw` function of the `S7Strategy` contract

```
    function repayLoanAndWithdraw(
        address borrowAsset,
        address proxy,
        uint256 repayAmt,
        uint256 pct
    ) public returns (uint256) {
        IERC20(borrowAsset).approve(lendingPool, repayAmt);
        ILendingPool(lendingPool).repay(borrowAsset, repayAmt, 2, proxy);
        uint256 withdrawAmt = (IERC20(aToken).balanceOf(proxy) * pct) / 100;
        S7Proxy(proxy).aaveWithdraw(supplyAsset, withdrawAmt);
        return withdrawAmt;
    }
```

It can be used to transfer the funds of an arbitrary proxy (specified by `proxy`) out to the `S7Strategy` contract, which can then be further withdrawn by any other account by invoking the `withdrawToken` function.

Impact

An attacker can steal user funds.

Failed Invariant

```
// Targeted strategy: S7UniswapV2USDCUSDT
function invariantV2DepositWithdrawalFailure() public {
    uint256 initialLpBal = IERC20(ADDRESS_USDC_USDT_V2_POOL).balanceOf(
        proxyAddress
    );

    vm.startPrank(user);
    s7.withdrawToken(
        address(0),
        ADDRESS_USDC,
        ADDRESS_USDC_USDT_V2_POOL,
        uint128(initialLpBal),
        1,
        address(0)
    );
    vm.stopPrank();

    require(
        user.balance >= (USER_AMOUNT * 980) / 1000,
        "lost more than 2% ETH"
    );
}
```

Recommendation

Make the `repayLoawnAndWithdraw` function private.

S7: Incorrect Parameter of _mintProxy() in the S7Strategy::onERC721Received()

Severity: **Medium** Difficulty: **Low**

Description

When receiving a Uniswap V3 position NFT, the `onERC721Received` function tries to create a proxy for the user (specified by the `from` argument) if it is not created yet. However, `msg.sender` is mistakenly used instead of `from`.

The `onERC721Received` function of the `S7Strategy` contract

```
//hook called when nft is transferred to contract
function onERC721Received(
    address operator,
    address from,
    uint256 tokenId,
    bytes calldata data
) external returns (bytes4) {
    // ...
    if (userPoolProxy[from][poolAddress] == address(0)) {
        _mintProxy(msg.sender, poolAddress);
    }
    require(v3PositionNft[from][token0][token1][poolFee] == 0, "E9");
    v3PositionNft[from][token0][token1][poolFee] = tokenId;
    bytes memory tokenData = abi.encode(poolFee, liquidity, token0, token1);
    INonfungiblePositionManagerStrategy(nfpm).safeTransferFrom(
        address(this),
        userPoolProxy[from][poolAddress],
        tokenId,
        tokenData
    );
```

Since the proxy is not correctly initialized, `userPoolProxy[from][poolAddress]` is the zero address which will make the transfer fail later.

Impact

The `onERC721Received` function will always revert when the user proxy is not initialized.

Recommendation

Change the problematic line to `_mintProxy(from, poolAddress)`

S7: Accounting Errors in the S7Strategy::withdrawToken()

Severity: **Medium** Difficulty: **Low**

Description

In the `withdrawToken` function, the variable `result` and `fee` represent the amount of the `supplyAsset` token:

The `withdrawToken` function of the `S7Strategy` contract

```
    //@dev pass address(0) for ETH
    function withdrawToken(
        address tokenOut,
        address borrowAsset,
        address poolAddress,
        uint128 amount,
        uint256 minAmountOut,
        address feeToken
    ) public whitelistedToken(tokenOut) {
        // ...
        result = IERC20(supplyAsset).balanceOf(address(this));
        require(result >= minAmountOut, "E2");
        //Take fee
        uint256 fee = ((
            IFees(feeContract).calcFee(
                strategyId,
                msg.sender,
                feeToken == address(0) ? tokenOut : feeToken
            )
        ) * result) / 1000;
        IERC20(supplyAsset).transfer(
            IFees(feeContract).feeCollector(strategyId),
            fee
        );
        //transfer balance
        _sendToken(tokenOut, msg.sender, result - fee);
        // ...
```

However, when the `_sendToken` function is called, the `tokenOut` tokens are sent out. The function is broken when `tokenOut` is not `supplyAsset`.

Impact

The function will not perform as expected. Especially when `tokenOut` is a no-revert-on-transfer ERC20 token like $ZRX, the function will silently fail.

Failed Invariant

This issue is found by testing the invariant mentioned in the previous issue S7: Fund Loss via S7Strategy::repayLoanAndWithdraw().
Originally, the previous issue can only result in user funds being locked in the strategy contract. With the help of this issue, the locked user funds can be further withdrawn by an attacker.

Recommendation

Use `supplyAsset` instead of `tokenOut` when invoking the `_sendToken` function.

S9: Insufficient Liquidity in Uniswap stETH-ETH V3 Pool

Severity: **Medium** Difficulty: **Low**

Description

The `depositToken` function swaps the deposited tokens into the ones accepted by the Ribbon vault through the Uniswap router if needed:

The `depositToken` function of the `S9Strategy` contract

```
function depositToken(
    address tokenIn,
    address vault,
    uint256 amount,
    uint256 minAmountOut
) public payable whitelistedToken(tokenIn) {
    ...
    amount = ISwapRouter(swapRouter).swapTokenForToken(
        tokenIn,
        vaultAsset,
        amount,
        minAmountOut,
        proxy
    );
```

One supported vault of S9 is the Ribbon STETH vault. When a user deposits ETH tokens, they will be swapped into STETH through the Uniswap stETH-ETH v3 pool. However, the liquidity of this pool may not be sufficient. On January 31, 2023, the TVL of the pool was only $394.98K, and our test showed that depositing 100 ETH resulted in ~70% loss on the Ethereum mainnet.

Impact

Users will suffer huge fund losses if they invest in the sub-strategy S9RibbonSTETH.

Failed Invariant

```
// Targeted strategy: S9RibbonSTETH
```

```
function invariantDepositWithdrawalFailure() public {
    skip(60 * 60 * 24 * 28);
    vm.startPrank(ADDRESS_KEEPER);
    rSTETH.rollToNextRound();
    vm.stopPrank();

    (, uint256 locked, , , , , , , ) = s9.getDepositData(
        user,
        address(rSTETH)
    );
    vm.startPrank(user);
    s9.queueWithdraw(address(rSTETH), locked);
    vm.stopPrank();

    skip(60 * 60 * 24 * 28);
    vm.startPrank(ADDRESS_KEEPER);
    rSTETH.rollToNextRound();
    vm.stopPrank();

    vm.startPrank(user);
    s9.withdrawToken(address(0), address(rSTETH), USER_DEPOSIT_AMOUNT, 0,
address(0));
    vm.stopPrank();

    require(user.balance >= (USER_DEPOSIT_AMOUNT * 50) / 100);
}
```

Recommendation

Switch to the Uniswap stETH-ETH v2 pool. The TVL of the pool is $2.60M on January 31, 2023.

Fix Log

ID	Title	Severity	Status
01	S3: Missing Check on Deposit Token	**High**	**Fixed at commit 9603fd367a990e553e080 25a6073f3e87c65e879**
02	S3: Emergency Withdraw Always Fails	**Low**	**Fixed at commit adfdaa4e91daf569174c8 69d67fbfd6a8785aa25**
03	S3: Claiming MTA Rewards in Tokens May Fail	**Low**	**Mitigated***
04	S4: Missing check on msg.sender in onERC721Received()	**Medium**	**Fixed at commit 169185369670b5db14b7 eb58464f70d0be401d08**
05	S7: Fund Loss via S7Strategy::repayLoanAndWithdraw()	**High**	**Fixed at commit 16afbb7f02969af9240c4 62f5a99c36d359f3576**
06	S7: Incorrect Parameter of _mintProxy() in the S7Strategy::onERC721Received()	**Medium**	**Fixed at commit 16afbb7f02969af9240c4 62f5a99c36d359f3576**
07	S7: Accounting Errors in the S7Strategy::withdrawToken()	**Medium**	**Fixed at commit 16afbb7f02969af9240c4 62f5a99c36d359f3576**
08	S9: Insufficient Liquidity in Uniswap stETH-ETH V3 Pool	**Medium**	**Fixed at commit a87c1b72d9ca61255002 d57241f36e10a7858341**

* This issue is mitigated by the front end that informs users that they need to wait for the MTA rewards growing bigger for withdrawal.

Appendix

Severity Categories

Severity	Description

Undetermined	The extent of the risk can not be determined during this assessment
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Low	The risk is relatively small or is not a risk that the customer indicated as important
Medium	Individual user’s information is at risk, exploitation would be bad for client’s reputation, moderate financial impact, possibly legal implication for client
High	Affects large number of users, very bad for clients reputation or posses great financial or legal risk

Difficulty Levels

Difficulty	Description
Undetermined	The difficulty of the exploit is undetermined during this assessment
Low	Commonly exploited, existing tools can be leveraged or can be easily scripted
Medium	Attacker must write a dedicated exploit, or need in depth knowledge of a complex system
High	The attacker must have privileged insider access or need to know extremely complicated technical details or must discover other issues to exploit this