SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Mycelium

Prepared for:	Mycelium
Prepared by:	Sherlock
Lead Security Expert:	WATCHPUG
Dates Audited:	October 5 - October 8, 2022
Prepared on:	October 21, 2022

Introduction

Mycelium specializes in data provision via the Mycelium Node and derivatives exchange infrastructure (formerly known as Tracer DAO).

Scope

Everything in the `mylink-contracts/src` folder.

Tests are run with foundry using `forgetest` or the NPM test script.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Medium	High
4	1

Security Experts

ctf_sec
WATCHPUG
hansfriese
berndartmueller
IllIllI
ak1
bin2chen
dipp
0x52
Ruhum
8olidity
rbserver
CodingNameKiki
Lambda
CRYP70
innertia
__141345__
Sm4rty
defsec
ellahi
rvierdiiev
sorrynotsorry
minhquanym
JohnSmith
caventa
joestakey
cccz

Issue H-1: Attacker can manipulate the pricePerShare to profit from future users' deposits

Source: https://github.com/sherlock-audit/2022-10-mycelium-judging/tree/main/001-H

Found by

Ruhum, ctf_sec, cccz, joestakey, ellahi, __141345__, 8olidity, hansfriese, minhquanym, 0x52, caventa, rvierdiiev, Sm4rty, rbserver, IllIIl, sorrynotsorry, JohnSmith, defsec, WATCHPUG, berndartmueller, ak1

Summary

By manipulating and inflating the pricePerShare to a super high value, the attacker can cause all future depositors to lose a significant portion of their deposits to the attacker due to precision loss.

Vulnerability Detail

A malicious early user can `deposit()` with `1wei` of `LINK` token as the first depositor of the Vault, and get `(1*STARTING_SHARES_PER_LINK)wei` of shares.

Then the attacker can send `STARTING_SHARES_PER_LINK-1` of `LINK` tokens and inflate the price per share from `1/STARTING_SHARES_PER_LINK` to 1.0000 .

Then the attacker call `withdraw()` to withdraw `STARTING_SHARES_PER_LINK-1` shares, and send `1e22` of `LINK` token and inflate the price per share from 1.000 to 1.000e22.

As a result, the future user who deposits `9999e18` will only receive `0` (from `9999e18*1/10000e18`) of shares token.

They will immediately lose all of their deposits.

Impact

Users may suffer a significant portion or even 100% of the funds they deposited to the Vault.

Code Snippet

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L131-142

```
function deposit(uint256 _amount) external {
    require(_amount > 0, "Amount must be greater than 0");
    require(_amount <= availableForDeposit(), "Amount exceeds available
 ↪  capacity");

    uint256 newShares = convertToShares(_amount);
    _mintShares(msg.sender, newShares);

    IERC20(LINK).transferFrom(msg.sender, address(this), _amount);
    _distributeToPlugins();

    emit Deposit(msg.sender, _amount);
}
```

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L614-L620

```
function convertToShares(uint256 _tokens) public view returns (uint256) {
    uint256 tokenSupply = totalSupply(); // saves one SLOAD
    if (tokenSupply == 0) {
        return _tokens * STARTING_SHARES_PER_LINK;
    }
    return _tokens.mulDivDown(totalShares, tokenSupply);
}
```

Tool used

Manual Review

Recommendation

Consider requiring a minimal amount of share tokens to be minted for the first minter, and send part of the initial mints as a permanent reserve to the DAO/treasury/deployer so that the pricePerShare can be more resistant to manipulation.

Issue M-1: When one of the plugins is broken or paused, `deposit()` or `withdraw()` of the whole Vault contract can malfunction

Source: https://github.com/sherlock-audit/2022-10-mycelium-judging/tree/main/006-M

Found by

ctf_sec, IIIIIII, berndartmueller, ak1, WATCHPUG

Summary

One malfunctioning plugin can result in the whole Vault contract malfunctioning.

Vulnerability Detail

A given plugin can temporally or even permanently becomes malfunctioning (cannot deposit/withdraw) for all sorts of reasons.

Eg, Aave V2 Lending Pool can be paused, which will prevent multiple core functions that the Aave v2 plugin depends on from working, including `lendingPool.deposit()` and `lendingPool.withdraw()`.

https://github.com/aave/protocol-v2/blob/master/contracts/protocol/lendingpool/LendingPool.sol#L54

https://github.com/aave/protocol-v2/blob/master/contracts/protocol/lendingpool/LendingPool.sol#L142-L146

```
function withdraw(
  address asset,
  uint256 amount,
  address to
) external override whenNotPaused returns (uint256) {
```

That's because the deposit will always goes to the first plugin, and withdraw from the last plugin first.

Impact

When Aave V2 Lending Pool is paused, users won't be able to deposit or withdraw from the vault.

Neither can the owner remove the plugin nor rebalanced it to other plugins to resume operation.

Because withdrawal from the plugin can not be done, and removing a plugin or re-balancing both rely on this.

Code Snippet

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L456-L473

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L492-L519

Tool used

Manual Review

Recommendation

1. Consider introducing a new method to pause one plugin from the Vault contract level;
2. Aave V2's Lending Pool contract has a view function paused(), consider returning `0` for `availableForDeposit()` and “availableForWithdrawal() when pool paused in AaveV2Plugin:

```
function availableForDeposit() public view override returns (uint256) {
    if (lendingPool.paused()) return 0;
    return type(uint256).max - balance();
}
```

```
function availableForWithdrawal() public view override returns (uint256) {
    if (lendingPool.paused()) return 0;
    return balance();
}
```

Issue M-2: Lack of sanity checks for new plugin address in `addPlugin()`

Source: https://github.com/sherlock-audit/2022-10-mycelium-judging/tree/main/010-M

Found by

Ruhum, ctf_sec, bin2chen, hansfriese, 0x52, dipp, berndartmueller, WATCHPUG

Summary

Without sanity checks for new plugin address in `addPlugin()`, wrong address or duplicated address can be added.

Vulnerability Detail

When adding a new plugin, there are no sanity checks for the new plugin's address.

However, adding a wrong address or duplicated address can cause severe damage to the users, and it may be irreversible:

When the new plugin is not a correct contract, `removePlugin()` will revert at `IPlugin(_plugin).withdraw(_amount);`.

Impact

When a wrong address is added as a plugin, many essential features of the Vault contract will malfunction, including `deposit()` and `withdraw()`, as `totalSupply()` will revert at `IPlugin(plugins[i]).balance()`.

When an existing plugin is wrongfully added as a new plugin, the `totalSupply()` will double count the balance of that plugin, which makes the user who deposits receives fewer shares and the users who withdraw, receives more tokens.

Code Snippet

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L314-L329

Tool used

Manual Review

Recommendation

1. Add a check to ensure `_plugin.Vault==address(this)`, which will first ensure the plugin address is a valid contract with Vault interface, and it's set to the correct vault address.
2. Add a check to ensure `_plugin` is new (not an existing one).

Issue M-3: `_withdrawFromPlugin()` will revert when `_withdrawalValues[i]==0`

Source: https://github.com/sherlock-audit/2022-10-mycelium-judging/tree/main/013-M

Found by

ctf_sec, hansfriese, WATCHPUG

Summary

Vulnerability Detail

When `_withdrawalValues[i]==0` in `rebalancePlugins()`, it means NOT to rebalance this plugin.

However, the current implementation still tries to withdraw 0 from the plugin.

This will revert in AaveV2Plugin as Aave V2's `validateWithdraw()` does not allow `0` withdrawals:

https://github.com/aave/protocol-v2/blob/554a2ed7ca4b3565e2ceaea0c454e5a70b3a2b41/contracts/protocol/libraries/logic/ValidationLogic.sol#L60-L70

```
function validateWithdraw(
  address reserveAddress,
  uint256 amount,
  uint256 userBalance,
  mapping(address => DataTypes.ReserveData) storage reservesData,
  DataTypes.UserConfigurationMap storage userConfig,
  mapping(uint256 => address) storage reserves,
  uint256 reservesCount,
  address oracle
) external view {
  require(amount != 0, Errors.VL_INVALID_AMOUNT);
```

`removePlugin()` will also always `_withdrawFromPlugin()` even if the plugin's balance is 0, as it will also tries to withdraw 0 in that case (balance is 0).

Impact

For AaveV2Plugin (and any future plugins that dont allow withdraw 0):

1. In every rebalance call, it must at least withdraw 1 wei from the plugin for the rebalance to work.

2. The plugin can not be removed or rebalanced when there is no balance in it.

If such a plugin can not deposit for some reason (paused by gov, AaveV2Plugin may face that), this will further cause the whole system unable to be rebalanced until the deposit resumes for that plugin.

Code Snippet

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L367-L373

Tool used

Manual Review

Recommendation

Only call `_withdrawFromPlugin()` when `IPlugin(pluginAddr).balance()>0`:

```
function removePlugin(uint256 _index) external onlyOwner {
    require(_index < pluginCount, "Index out of bounds");
    address pluginAddr = plugins[_index];
    if (IPlugin(pluginAddr).balance() > 0){
        _withdrawFromPlugin(pluginAddr, IPlugin(pluginAddr).balance());
    }
    uint256 pointer = _index;
    while (pointer < pluginCount - 1) {
        plugins[pointer] = plugins[pointer + 1];
        pointer++;
    }
    delete plugins[pluginCount - 1];
    pluginCount--;

    IERC20(LINK).approve(pluginAddr, 0);

    emit PluginRemoved(pluginAddr);
}
```

```
function rebalancePlugins(uint256[] memory _withdrawalValues) external onlyOwner
 ↪  {
    require(_withdrawalValues.length == pluginCount, "Invalid withdrawal
 ↪  values");
    for (uint256 i = 0; i < pluginCount; i++) {
        if (_withdrawalValues[i] > 0)
            _withdrawFromPlugin(plugins[i], _withdrawalValues[i]);
    }
    _distributeToPlugins();
}
```

Issue M-4: Frontrun `deposit()` can cause the depositor to lose all the funds

Source: https://github.com/sherlock-audit/2022-10-mycelium-judging/tree/main/029-M

Found by

ctf_sec, Lambda, CRYP70, 8olidity, bin2chen, hansfriese, innertia, dipp, rbserver, CodingNameKiki, WATCHPUG

Summary

The attacker can frontrun the first depositor's `deposit()` transaction and transfer LINK tokens to the Vault contract directly and cause the depositor (and all the future depositors) to lose all the funds.

Vulnerability Detail

In `onTokenTransfer()`, if there are some existing balance in the Vault contract, say 1 wei of LINK token, `supplyBeforeTransfer` will be `1`, since `totalShares==0`, `newShares` will always be `0`.

The same applies for `deposit()`.

Impact

The depositor who got frontrun by the attacker will lose all their funds. And all the future depositors.

Code Snippet

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L252-L276

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L131-L142

https://github.com/sherlock-audit/2022-10-mycelium/blob/main/mylink-contracts/src/Vault.sol#L614-L620

Tool used

Manual Review

Recommendation

It should check if `totalShares==0` to decide whether this is the first mint.