Arrakis Finance v2 PALM Audit Report

Oct 28, 2022

WATCHPUG

Table of Contents

Summary

This report has been prepared for Arrakis Finance v2 PALM Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**Arrakis Finance v2 PALM Audit Report**
Codebase	**https://github.com/ArrakisFinance/v2-palm**
Commit	**fe25734501e6cc7f429d34bbf8f70d86845c51c9**
Language	**Solidity**

Audit Summary

Delivery Date	**Oct 28, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**13**

[WP-M1] Unsafe `ERC20.approve()`

Medium

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/PALMTerms.sol#L40-L113

```
function openTerm(SetupPayload calldata setup_, uint256 mintAmount_)
    external
    payable
    override
    noLeftOver(setup_.token0, setup_.token1)
    returns (address vault)
{
@@ 47,103 @@

    setup_.token0.approve(vault, setup_.amount0);
    setup_.token1.approve(vault, setup_.amount1);

    vaultV2.setRestrictedMint(address(this));

    vaultV2.mint(mintAmount_, address(this));

    emit SetupVault(setup_.owner, vault);
}
```

`PALMTerms#openTerm()` calls `IERC20.approve()` before the call to `vaultV2.mint()`.

However, there are many Weird ERC20 Tokens that won't work correctly using the standard `IERC20` interface.

For example, the `USDT` token on mainnet does not return a bool on the `approve()` method.

As a result, when calling `IERC20(USDT).approve()`, the transaction will revert with an error: "function returned an unexpected amount of data", as the expected return data is a bool, but it actually does not return any data.

This means that `PALMTerms#openTerm()` doesn't work well with one of the most popular tokens.

`increaseLiquidity()` has the same issue.

Recommendation

Calling `safeApprove(0)` first and then `safeApprove(amount)`.

Status

✓ **Fixed**

[WP-M2] Unsafe `ERC20.transfer()`

Medium

Issue Description

Calling `ERC20.transfer()` directly without using `SafeERC20.transfer()` will result in reverts for some of the most popular tokens, e.g. USDT, BNB, OMG.

See: Weird ERC20 Tokens

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L45-L53

```
    modifier noLeftOver(IERC20 token0_, IERC20 token1_) {
        uint256 token0Balance = token0_.balanceOf(address(this));
        uint256 token1Balance = token1_.balanceOf(address(this));
        _;
        uint256 leftOver0 = token0_.balanceOf(address(this)) - token0Balance;
        uint256 leftOver1 = token1_.balanceOf(address(this)) - token1Balance;
        if (leftOver0 > 0) token0_.transfer(msg.sender, leftOver0);
        if (leftOver1 > 0) token1_.transfer(msg.sender, leftOver1);
    }
```

Recommendation

Consider using OpenZeppelin's `SafeERC20` library.

Status

✓ **Fixed**

[WP-L3] Unbounded loop may cause `_addVault()` to malfunction due to out-of-gas

Low

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L249-L258

```
function _addVault(address creator_, address vault_) internal {
        address[] storage vaultsOfCreator = vaults[creator_];

        for (uint256 i = 0; i < vaultsOfCreator.length; i++) {
            require(vaultsOfCreator[i] != vault_, "PALMTerms: vault exist");
        }

        vaultsOfCreator.push(vault_);
        emit AddVault(creator_, vault_);
    }
```

An attacker can add a lot of vaults under the victim's address (`creator_`).

Once the length of `vaultsOfCreator` is large enough, and the gas cost of function `_addVault()` can exceed the block limit, making it impossible to `addVault()` .

Recommendation

See [Recommendation] of [WP-G12].

Status

✓ **Fixed**

[WP-M4] `PALMManager._preExec()` Lack of internal accounting for accumulated rebalancing fee

Medium

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/PALMManager.sol#L49-L64

```
49      function _preExec(address vault_, uint256 feeAmount_)
50          internal
51          returns (uint256 balance)
52      {
53          VaultInfo memory vaultInfo = vaults[vault_];
54          require(
55              vaultInfo.balance >= feeAmount_,
56              "PALMManager: Not enough balance to pay fee"
57          );
58          balance = vaultInfo.balance - feeAmount_;
59
60          // update lastRebalance time
61          // solhint-disable-next-line not-rely-on-time
62          vaults[vault_].lastRebalance = block.timestamp;
63          vaults[vault_].balance = balance;
64      }
```

In the current implementation, a `feeAmount_` parameter will be passed by Operators when calling `rebalance()`, the `feeAmount_` will be charged from the vault's balance.

However, there is no internal accounting to record the total accumulated fee amount for the whole system or one particular operator.

There is no method to withdraw the fees either.

Recommendation

Consider adding a storage variable called `operatorBalances` to record the operators' fee earnings:

```
function _preExec(address vault_, uint256 feeAmount_)
    internal
    returns (uint256 balance)
{
    VaultInfo memory vaultInfo = vaults[vault_];
    require(
        vaultInfo.balance >= feeAmount_,
        "PALMManager: Not enough balance to pay fee"
    );
    balance = vaultInfo.balance - feeAmount_;

    operatorBalances[msg.sender] += feeAmount_;

    // update lastRebalance time
    // solhint-disable-next-line not-rely-on-time
    vaults[vault_].lastRebalance = block.timestamp;
    vaults[vault_].balance = balance;
}
```

And also add a new method for the operators to withdraw their earnings.

Status

✓ Fixed

[WP-S5] An alternative implementation of `renewTerm()`

Issue Description

The current implementation of `renewTerm()` will pull all the funds back and then reinit the vault, which is not necessary.

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/PALMTerms.sol#L180-L220

```
function renewTerm(IArrakisV2 vault_)
    external
    override
    noLeftOver(vault_.token0(), vault_.token1())
{
    IPALMManager manager_ = IPALMManager(manager);
    require( // solhint-disable-next-line not-rely-on-time
        manager_.getVaultInfo(address(vault_)).termEnd < block.timestamp,
        "PALMTerms: term not ended."
    );
    IPALMManager(manager).renewTerm(address(vault_));

    (uint256 amount0, uint256 amount1, uint256 balance) = _burn(
        vault_,
        address(this),
        resolver
    );

    uint256 emolumentAmt0 = _getEmolument(amount0, emolument);
    uint256 emolumentAmt1 = _getEmolument(amount1, emolument);
    vault_.token0().approve(address(vault_), amount0 - emolumentAmt0);
    vault_.token1().approve(address(vault_), amount1 - emolumentAmt1);
    if (emolumentAmt0 > 0)
        vault_.token0().safeTransfer(termTreasury, emolumentAmt0);
    if (emolumentAmt1 > 0)
        vault_.token1().safeTransfer(termTreasury, emolumentAmt1);
    {
        Inits memory inits;
        (inits.init0, inits.init1) = _getInits(
            balance,
            amount0 - emolumentAmt0,
            amount1 - emolumentAmt1
```

```
        );

        vault_.setInits(inits.init0, inits.init1);
    }

    vault_.mint(balance, address(this));

    emit RenewTerm(address(vault_), emolumentAmt0, emolumentAmt1);
}
```

Recommendation

```
function renewTerm(IArrakisV2 vault_)
    external
    override
    noLeftOver(vault_.token0(), vault_.token1())
{
    IPALMManager manager_ = IPALMManager(manager);
    require( // solhint-disable-next-line not-rely-on-time
        manager_.getVaultInfo(address(vault_)).termEnd < block.timestamp,
        "PALMTerms: term not ended."
    );
    IPALMManager(manager).renewTerm(address(vault_));

    uint256 balance = IERC20(address(vault_)).balanceOf(this);
    uint256 emolumentShares = _getEmolument(balance, emolument);

    BurnLiquidity[] memory burnPayload = resolver.standardBurnParams(
        emolumentShares,
        vault_
    );
    (uint256 emolumentAmt0, uint256 emolumentAmt1) = vault_.burn(burnPayload,
emolumentShares, termTreasury);

    emit RenewTerm(address(vault_), emolumentAmt0, emolumentAmt1);
}
```

Status

✓ Fixed

[WP-S6] A simpler and condition controlled implementation of `decreaseLiquidity()`

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/PALMTerms.sol#L205-L287

```
function decreaseLiquidity(
    DecreaseBalance calldata decreaseBalance_,
    uint256 mintAmount_
)
    external
    override
    noLeftOver(
        decreaseBalance_.vault.token0(),
        decreaseBalance_.vault.token1()
    )
{
    _requireMintNotZero(mintAmount_);
    _requireIsOwner(vaults[msg.sender], address(decreaseBalance_.vault));

    address me = address(this);

    (uint256 amount0, uint256 amount1, ) = _burn(
        decreaseBalance_.vault,
        me,
        resolver
    );
    require(
        decreaseBalance_.amount0 < amount0,
        "PALMTerms: send back amount0 > amount0"
    );
    require(
        decreaseBalance_.amount1 < amount1,
        "PALMTerms: send back amount1 > amount1"
    );

    uint256 emolumentAmt0 = _getEmolument(
        decreaseBalance_.amount0,
        emolument
```

```
    );
    uint256 emolumentAmt1 = _getEmolument(
        decreaseBalance_.amount1,
        emolument
    );

    {
        IERC20 token0 = decreaseBalance_.vault.token0();
        IERC20 token1 = decreaseBalance_.vault.token1();

        if (emolumentAmt0 > 0)
            token0.safeTransfer(termTreasury, emolumentAmt0);
        if (emolumentAmt1 > 0)
            token1.safeTransfer(termTreasury, emolumentAmt1);

        token0.safeTransfer(
            decreaseBalance_.to,
            decreaseBalance_.amount0 - emolumentAmt0
        );
        token1.safeTransfer(
            decreaseBalance_.to,
            decreaseBalance_.amount1 - emolumentAmt1
        );
        token0.approve(
            address(decreaseBalance_.vault),
            amount0 - decreaseBalance_.amount0
        );
        token1.approve(
            address(decreaseBalance_.vault),
            amount1 - decreaseBalance_.amount1
        );
    }
    {
        (uint256 init0, uint256 init1) = _getInits(
            mintAmount_,
            amount0 - decreaseBalance_.amount0,
            amount1 - decreaseBalance_.amount1
        );
        decreaseBalance_.vault.setInits(init0, init1);
    }

    decreaseBalance_.vault.mint(mintAmount_, me);

```

```
    emit DecreaseLiquidity(
        msg.sender,
        address(decreaseBalance_.vault),
        emolumentAmt0,
        emolumentAmt1
    );
}
```

The current implementation will almost always success, even if the price has been changed and therefore changed the ratio/amounts of tokens pulled out with the transaction.

That's because it's pulling all the funds back first and taking the desired amounts, unless the caller is pulling all or at least a majority of the liquidity, this will most certainly succeed.

This can be a problem if the caller has a very specific intention with this `decreaseLiquidity()` call:

PoC

Given:

- The fair market price of ETH is 2000 USDC;

1. A huge swap has skewed the price in the Uni v3 pool from 2000 USDC to 1000 USDC;
2. The owner of the PALM identified that as an arb opportunity and called `decreaseLiquidity()` to pull more ETH out and take a profit;
3. Other arb txs have been minted before the `decreaseLiquidity()` tx;
4. The owner of the PALM ends up with an undesirable price for the removal of liquidity.

Recommendation

We propose a new design that requires the caller to specify the amount of shares to burn and minimum amounts as slippage control:

```
function decreaseLiquidity(
    RemoveLiquidityData memory decreaseBalance_ // from `/structs/SArrakisV2Router.sol`
)
    external
```

```
    override
{
    _requireIsOwner(vaults[msg.sender], address(decreaseBalance_.vault));

    BurnLiquidity[] memory burnPayload = resolver.standardBurnParams(
        decreaseBalance_.burnAmount,
        decreaseBalance_.vault
    );

    (uint256 amount0, uint256 amount1) = decreaseBalance_.vault.burn(
        burnPayload,
        decreaseBalance_.burnAmount,
        address(this)
    );

    require(
        amount0 >= decreaseBalance_.amount0Min &&
            amount1 >= decreaseBalance_.amount1Min,
        "PALMTerms: received below minimum"
    );

    uint256 emolumentAmt0;
    uint256 emolumentAmt1;

    if (amount0 > 0) {
        IERC20 token0 = decreaseBalance_.vault.token0();

        emolumentAmt0 = _getEmolument(
            amount0,
            emolument
        );
        token0.safeTransfer(termTreasury, emolumentAmt0);
        token0.safeTransfer(
            decreaseBalance_.receiver,
            amount0 - emolumentAmt0
        );
    }

    if (amount1 > 0) {
        IERC20 token1 = decreaseBalance_.vault.token1();

        emolumentAmt1 = _getEmolument(
            amount1,
```

```
            emolument
        );
        token1.safeTransfer(termTreasury, emolumentAmt1);
        token1.safeTransfer(
            decreaseBalance_.receiver,
            amount1 - emolumentAmt1
        );
    }

    emit DecreaseLiquidity(
        msg.sender,
        address(decreaseBalance_.vault),
        emolumentAmt0,
        emolumentAmt1
    );
}
```

Status

✓ Fixed

[WP-S7] An alternative implementation of `increaseLiquidity()`

Issue Description

Considering the fact that the vault is 100% owned by the vault's owner, `increaseLiquidity()` can be simplified to plain token transfers to the vault.

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/PALMTerms.sol#L116-L177

```
function increaseLiquidity(
    IncreaseBalance calldata increaseBalance_, // memory instead of calldata to set values
    uint256 mintAmount_
)
    external
    override
    noLeftOver(
        increaseBalance_.vault.token0(),
        increaseBalance_.vault.token1()
    )
{
    _requireMintNotZero(mintAmount_);
    _requireProjectAllocationGtZero(
        increaseBalance_.projectTknIsTknZero,
        increaseBalance_.amount0,
        increaseBalance_.amount1
    );
    _requireIsOwner(vaults[msg.sender], address(increaseBalance_.vault));

    (uint256 amount0, uint256 amount1, ) = _burn(
        increaseBalance_.vault,
        address(this),
        resolver
    );

    // Transfer to termTreasury the project token emolment.
    increaseBalance_.vault.token0().safeTransferFrom(
        msg.sender,
        address(this),
        increaseBalance_.amount0
    );
```

```
    increaseBalance_.vault.token1().safeTransferFrom(
        msg.sender,
        address(this),
        increaseBalance_.amount1
    );

    increaseBalance_.vault.token0().approve(
        address(increaseBalance_.vault),
        increaseBalance_.amount0 + amount0
    );

    increaseBalance_.vault.token1().approve(
        address(increaseBalance_.vault),
        increaseBalance_.amount1 + amount1
    );

    {
        Inits memory inits;
        (inits.init0, inits.init1) = _getInits(
            mintAmount_,
            increaseBalance_.amount0 + amount0,
            increaseBalance_.amount1 + amount1
        );

        increaseBalance_.vault.setInits(inits.init0, inits.init1);
    }

    increaseBalance_.vault.mint(mintAmount_, address(this));

    emit IncreaseLiquidity(msg.sender, address(increaseBalance_.vault));
}
```

https://github.com/ArrakisFinance/vault-v2-core/blob/702a343cea74dc9a551d6098926448c8e62d798b/contracts/ArrakisV2.sol#L52-L98

```
function mint(uint256 mintAmount_, address receiver_)
    external
    nonReentrant
    returns (uint256 amount0, uint256 amount1)
{
```

```
@@ 57,81 @@

    amount0 = FullMath.mulDivRoundingUp(mintAmount_, current0, denominator);
    amount1 = FullMath.mulDivRoundingUp(mintAmount_, current1, denominator);

    _mint(receiver_, mintAmount_);

    // transfer amounts owed to contract
    if (amount0 > 0) {
        token0.safeTransferFrom(msg.sender, me, amount0);
    }
    if (amount1 > 0) {
        token1.safeTransferFrom(msg.sender, me, amount1);
    }

    emit LogUncollectedFees(fee0, fee1);
    emit LogMint(receiver_, mintAmount_, amount0, amount1);
}
```

Recommendation

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/PALMTerms.sol#L116-L177

```
function increaseLiquidity(
    IncreaseBalance calldata increaseBalance_ // memory instead of calldata to set values
)
    external
    override
{
    increaseBalance_.vault.token0().safeTransferFrom(
        msg.sender,
        increaseBalance_.vault,
        increaseBalance_.amount0
    );
    increaseBalance_.vault.token1().safeTransferFrom(
        msg.sender,
        increaseBalance_.vault,
        increaseBalance_.amount1
```

```
    );

    emit IncreaseLiquidity(msg.sender, address(increaseBalance_.vault));
}
```

Status

✓ **Fixed**

[WP-I8] Multiple methods on `PALMTerms` will result in a rug-like behavior

Informational

Issue Description

There are 3 methods on `PALMTerms` that are intended for regular operations, which should not result in a significant change in liquidity:

- increaseLiquidity()
- renewTerm()
- decreaseLiquidity()

The current implementation of these functions will actually withdraw ALL the liquidity first, then call `vault.mint()` to put the money back:

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/PALMTerms.sol#L223-L305

```
function decreaseLiquidity(
    DecreaseBalance calldata decreaseBalance_,
    uint256 mintAmount_
)
    external
    override
    noLeftOver(
        decreaseBalance_.vault.token0(),
        decreaseBalance_.vault.token1()
    )
{
    _requireMintNotZero(mintAmount_);
    _requireIsOwner(vaults[msg.sender], address(decreaseBalance_.vault));

    address me = address(this);

    (uint256 amount0, uint256 amount1, ) = _burn(
        decreaseBalance_.vault,
        me,
        resolver
```

```
    );
@@ 244,295 @@

    decreaseBalance_.vault.mint(mintAmount_, me);

    emit DecreaseLiquidity(
        msg.sender,
        address(decreaseBalance_.vault),
        emolumentAmt0,
        emolumentAmt1
    );
}
```

However, `vault.mint()` won't actually put the liquidity back to the Uniswap v3 position, until the next `rebalance()` :

https://github.com/ArrakisFinance/vault-v2-core/blob/702a343cea74dc9a551d6098926448c8e62d798b/contracts/ArrakisV2.sol#L52-L98

```
function mint(uint256 mintAmount_, address receiver_)
    external
    nonReentrant
    returns (uint256 amount0, uint256 amount1)
{
    require(mintAmount_ > 0, "MA");
    require(
        restrictedMint == address(0) || msg.sender == restrictedMint,
        "R"
    );
    address me = address(this);
    uint256 totalSupply = totalSupply();
    (
        uint256 current0,
        uint256 current1,
        uint256 fee0,
        uint256 fee1
    ) = totalSupply > 0
            ? UnderlyingHelper.totalUnderlyingWithFees(
                UnderlyingPayload({
                    ranges: ranges,
```

```
                    factory: factory,
                    token0: address(token0),
                    token1: address(token1),
                    self: me
                })
            )
            : (init0, init1, 0, 0);
    uint256 denominator = totalSupply > 0 ? totalSupply : 1 ether;
    /// @dev current0 and current1 include fees and left over (but not admin balances)

    amount0 = FullMath.mulDivRoundingUp(mintAmount_, current0, denominator);
    amount1 = FullMath.mulDivRoundingUp(mintAmount_, current1, denominator);

    _mint(receiver_, mintAmount_);

    // transfer amounts owed to contract
    if (amount0 > 0) {
        token0.safeTransferFrom(msg.sender, me, amount0);
    }
    if (amount1 > 0) {
        token1.safeTransferFrom(msg.sender, me, amount1);
    }

    emit LogUncollectedFees(fee0, fee1);
    emit LogMint(receiver_, mintAmount_, amount0, amount1);
}
```

This creates a gap period between the `increaseLiquidity()` / `renewTerm()` / `decreaseLiquidity()` transaction and the `rebalance()` .

During that period, it would look just like the protocol pulled the rug, and the liquidity will be much thinner than before (depends on the percentage owned by the protocol).

We believe this is undesirable and also unnecessary.

Recommendation

See [WP-S5], [WP-S6], [WP-S7].

Status

✓ Fixed

[WP-L9] Wrong function name

Low

Issue Description

https://github.com/ArrakisFinance/vault-v2-agreement/blob/0b8fc8f69de3a1c662e623e5faa0b86603a5bc41/contracts/functions/FTerms.sol#L95-L97

```
95  function _requireAddressNotZero(uint256 mintAmount_) pure {
96      require(mintAmount_ > 0, "Terms: mintAmount zero.");
97  }
```

Should be named `_requireMintAmountNotZero`.

Status

✓ Fixed

[WP-N10] `PALMTermsStorage#noLeftOver` Misleading modifier Name

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L45-L53

```
    modifier noLeftOver(IERC20 token0_, IERC20 token1_) {
        uint256 token0Balance = token0_.balanceOf(address(this));
        uint256 token1Balance = token1_.balanceOf(address(this));
        _;
        uint256 leftOver0 = token0_.balanceOf(address(this)) - token0Balance;
        uint256 leftOver1 = token1_.balanceOf(address(this)) - token1Balance;
        if (leftOver0 > 0) token0_.transfer(msg.sender, leftOver0);
        if (leftOver1 > 0) token1_.transfer(msg.sender, leftOver1);
    }
```

`noLeftOver` can be misundertood as `require(leftOverX == 0, "...")`.

Consider renaming to `collectLeftOver` or `sweepLeftOver`.

Status

✓ Fixed

[WP-N11] Unnecessary type casting

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L231-L243

```
function withdrawVaultBalance(
        address vault_,
        uint256 amount_,
        address payable to_
    ) external override requireAddressNotZero(vault_) {
        address vaultAddr = address(vault_);
        IPALMManager manager_ = IPALMManager(manager);
        (uint256 balance, , , , ) = manager_.vaults(vaultAddr);
        _requireIsOwner(vaults[msg.sender], vaultAddr);
        manager_.withdrawVaultBalance(vault_, amount_, to_);

        emit LogWithdrawVaultBalance(msg.sender, vaultAddr, to_, balance);
    }
```

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L179-L199

```
function setVaultData(address vault_, bytes calldata data_)
        external
        override
        requireAddressNotZero(vault_)
    {
        address vaultAddr = address(vault_);
        _requireIsOwnerOrDelegate(
            delegateByVaults[vault_],
            vaults[msg.sender],
            vaultAddr
        );
        IPALMManager(manager).setVaultData(vault_, data_);

        emit LogSetVaultData(
```

```
            delegateByVaults[vault_] != address(0)
                ? delegateByVaults[vault_]
                : msg.sender,
            vaultAddr,
            data_
        );
    }
```

https:
//github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L201-L221

```
function setVaultStratByName(address vault_, string calldata strat_)
        external
        override
        requireAddressNotZero(vault_)
    {
        address vaultAddr = address(vault_);
        _requireIsOwnerOrDelegate(
            delegateByVaults[vault_],
            vaults[msg.sender],
            vaultAddr
        );
        IPALMManager(manager).setVaultStraByName(vault_, strat_);

        emit LogSetVaultStratByName(
            delegateByVaults[vault_] != address(0)
                ? delegateByVaults[vault_]
                : msg.sender,
            vaultAddr,
            strat_
        );
    }
```

https:
//github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L223-L229

```
function setDelegate(address vault_, address delegate_) external override {
        address vaultAddr = address(vault_);
        _requireIsOwner(vaults[msg.sender], vaultAddr);
        _setDelegate(vault_, delegate_);

        emit LogSetDelegate(msg.sender, vaultAddr, delegate_);
    }
```

Status

✓ Fixed

[WP-G12] Using `EnumerableSet.AddressSet` can avoid unnecessary storage reads

Gas

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L249-L258

```
    function _addVault(address creator_, address vault_) internal {
        address[] storage vaultsOfCreator = vaults[creator_];

        for (uint256 i = 0; i < vaultsOfCreator.length; i++) {
            require(vaultsOfCreator[i] != vault_, "PALMTerms: vault exist");
        }

        vaultsOfCreator.push(vault_);
        emit AddVault(creator_, vault_);
    }
```

https://github.com/ArrakisFinance/v2-palm//blob/fe25734501e6cc7f429d34bbf8f70d86845c51c9/contracts/abstracts/PALMTermsStorage.sol#L36

```
    mapping(address => address[]) public vaults;
```

Recommendation

```
    function _addVault(address creator_, address vault_) internal {
        EnumerableSet.AddressSet storage vaultsOfCreator = vaults[creator_];

        require(!vaultsOfCreator.contains(vault_), "PALMTerms: vault exist");

        vaultsOfCreator.add(vault_);
        emit AddVault(creator_, vault_);
```

```
    }
```

Status

✓ Fixed

[WP-N13] Unused imports

Issue Description

contracts/PALMManager.sol

- ”IPALMManager” is unused
- ”IUniswapV3Pool” is unused
- ”SafeERC20” is unused
- ”IERC20” is unused

contracts/PALMTerms.sol

- ”_requireTokenMatch” is unused
- ”_requireIsOwnerOrDelegate” is unused

contracts/abstracts/PALMTermsStorage.sol

- ”FullMath” is unused
- ”SafeCast” is unused
- ”_getInits” is unused
- ”_requireTokenMatch” is unused
- ”_getEmolument” is unused
- ”_requireProjectAllocationGtZero” is unused
- ”_requireTknOrder” is unused
- ”_burn” is unused

Status

✓ **Fixed**

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.