shieldify

Adapters

Possum Labs

SECURITY REVIEW

Date: 7 May 2024

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and have secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Possum Labs - Adapters

Adapters are customizable plugins that enable Possum products to interact with external protocols in many different ways.

While the variety of use cases for Adapters is limitless, the first Adapter will be built on top of the v2 Portals. The Adapter will launch alongside Portals v2, and will enable users to claim upfront yield from any v2 Portal in any liquid token available on Arbitrum's DEXes. Technically, the user will still request upfront yield in PSM, but behind the scenes, the Adapter will swap that PSM for the requested token via select DEXs or aggregators on Arbitrum.

More Adapters can be added to Portals (or any future Possum product) at any time, and there's no limit to the number which can be added to any product. For example, future Adapters may add the following functions to Portals v2:

- Deposit any token into any Portal
- Automated upfront yield strategies (e.g., automatically buy leverage long ETH position with upfront yield)
- Provide liquidity with upfront yield

Take a deep dive into Possum's documentation here.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The bug bounty race was conducted in the span of two weeks (April 22nd until May 2nd) and a total of 47 smart contract security researchers participated. This SPP was the second layer in PossumLabs' planned security methodology for the codebase, which had been audited beforehand.

Overall, the code is written to very high standards and implements very solid security practices. Shieldify's Private Pool race contributed by identifying issues of varying severity, including weak slippage, susceptibility to reorgs and unintended business logic, among other less severe findings.

Shieldify's team extends their gratitude to Possum Labs' team for their exceptional work communication and responsiveness.

5.1 Protocol Summary

Project Name	**Possum Labs - Adapters**
Repository	Adapters
Type of Project	DeFi
Audit Timeline	11 days
Review Commit Hash	c4406549fd8dfd96839381bbbbf48266af798ed0
Fixes Review Commit Hash	9637c1ae722d92e6ec0c11e9d8a7bc7d4d31e19c

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/Portal/PortalV2MultiAsset.sol	436
src/AdapterV1.sol	424
src/interfaces/IAdapterV1.sol	15
src/interfaces/IPortalV2MultiAsset.sol	3
Total	**878**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **2**
- **Medium** issues: **3**
- **Low** issues: **12**

ID	Title	Severity	Status
[H-01]	`AdapterV1.sol` Contract Always Uses the Entire Balance of PSM of the Adapter, Which Can Lead to Weak Slippage	High	Fixed
[H-02]	Users Fund Loss When Adding Liquidity to Ramses Since Remaining Tokens Are Not Paid Out	High	Fixed
[M-01]	Anyone Can Call `executeMigration()` Once `migrationDestination` Is Set	Medium	Fixed
[M-02]	Susceptibility to Reorg Attacks	Medium	Acknowledged
[M-03]	Parameters and Return Values Mismatch in Contracts and Interfaces	Medium	Fixed
[L-01]	Remaining Tokens Should Be Sent to `_swap.recevier()` Instead of `msg.sender` in `swapOneInch()` Function	Low	Fixed
[L-02]	`totalPrincipalStaked` Is Not Decreased When NFT Is Minted	Low	Fixed
[L-03]	`acceptMigrationDestination()` Can Be Called Multiple Times to Brick Migration Clearing All the Funds in the New Adapter	Low	Fixed
[L-04]	Missing Recipient Address Validation in `sellPortalEnergy()` Function of `AdaptersV1` Contract	Low	Fixed

ID	Title	Severity	Status
[L-05]	`increaseAllowances()` Can Not Be Called Again Until the Entire Allowance of `principalToken` Is Used	Low	Fixed
[L-06]	The Owner Can Bypass the Voting Process to Steal All Users Funds	Low	Fixed
[L-07]	Staking Dependency Reduces User Influence in Migration	Low	Fixed
[L-08]	Incorrect Event Emission in `buyPortalEnergy()` Function	Low	Fixed
[L-09]	There Is No Function to Withdraw the Funds Which is Received from `receive()` or `fallback()` Function	Low	Fixed
[L-10]	Inconsistent Input Validation in `burnPortalEnergyToken()` Function	Low	Fixed
[L-11]	Ensure Sanity Checks on the `swapDescription.flags`	Low	Fixed
[L-12]	Attempting to Vote After 50 Percent + Have Been Reached Will Always Revert	Low	Acknowledged

7. Findings

[H-01] `AdapterV1.sol` Contract Always Uses the Entire Balance of PSM of the Adapter, Which Can Lead to Weak Slippage

Severity

High Risk

Reported By

EgisSec

Description

When `setPortalEnergy` is called with `mode = 1` the protocol will first swap PSM for WETH using 1inch, then use PSM and WETH to add liquidity to a Ramses pair.

In order to do this, the user has to specify 2 slippages, 1 for the 1inch swap and 1 when calculating how much liquidity needs to be added to the Ramses pair.

Let's go over the code one step at a time.

First, we have to sell our portal tokens to get PSM which is sent to the adapter. Second, we do a swap using 1inch which will swap some amount of the received PSM for WETH, when the swap is done the WETH is received in the adapter. Third, we get `PSMBalance` and `WETHBalance` which are then passed to `_addLiquidity()` which will calculate `amountPSM` and `amountWETH` which need to be transferred to the Ramses pair.

1inch has a case, where the tokens that were used for the swap were actually more than necessary, so the unspent amount is refunded to `msg.sender`. This is what this part of the code is used for.

```
// @dev Send remaining tokens back to the user if not called from
    addLiquidity
if (!_forLiquidity) {
     uint256 remainAmount = _swap.psmAmount - spentAmount_;
     if (remainAmount > 0) PSM.safeTransfer(msg.sender, remainAmount);
}
```

This will refund any amount that wasn't used to `msg.sender`, but this only happens if `forLiquidity = false`, but when we call `sellPortalEnergy()`, we use `forLiquidity = true` which won't refund any unused amount.

If 1inch refunds any PSM, it will later be used when calling `_addLiquidity()`, as we use `PSMBalance = PSM.balanceOf(address(this))`, which uses the entire PSM balance of the contract.

This is the actual problem because the tokens aren't refunded to `msg.sender`, `_addLiquidity()` will be called with a larger `PSMBalance` than the user expected, which will drastically affect his slippage `minPSM` because now the amount of PSM that will be used to add liquidity is larger than he expected.

There is no way to know if 1inch will use up all of the PSM for the swap, due to market conditions and when the tx will be executed, so there is no way to predict `minPSM` in the case where 1inch refunds PSM to the Adapter.

Impact

Inflated/unfair slippage used for user

Attack Scenario

To better visualize this, I'll give an example. The correlation between PSM and WETH is simplified to better understand the issue.

1. Alice wants to sell her portal tokens and then add liquidity in the Ramses pair.
2. She calls `sellPortalTokens()`.
3. She sells 200 PSM worth of portal tokens and she wants to use 100 PSM to swap for 1 WETH, the other 100 she wants to use to add liquidity to the pool.
4. 1inch uses up only 50 PSM for 1 WETH, so the other 50 PSM are refunded directly to the Adapter.
5. `PSMBalance` is now 150, as the code uses the entire balance of the Adapter to calculate how much liquidity needs to be added.
6. Alice specified 90 PSM as `minPSM`, as she expected only 100 PSM to be used to add liquidity, but now 150 is used.
7. Due to market conditions, `amountPSM = 100`, but `PSMBalance` is 150, meaning that the price dropped a lot more than Alice would have wanted.
8. Because Alice specified `minPSM = 90` the tx goes through and Alice let a tx which slipped in price a lot more than she would have agreed to if she knew that the 150 PSM would be used to add liquidity instead of 100 PSM.

Location of Affected Code

File: src/AdapterV1.sol#L597-L600

Recommendation

It's best to always refund any unused PSM from the 1Inch swap, this will make the function much more predictable and won't result in cases where a tx goes through with a much more inflated PSM amount, thus making the user's slippage inflated and very high.

Team Response

Fixed as proposed.

[H-02] Users Fund Loss When Adding Liquidity to Ramses Since Remaining Tokens Are Not Paid Out

Severity

High Risk

Reported By

carrotsmuggler

Description

User funds are lost when adding liquidity to Ramses since the remaining tokens are not paid out.

Impact

Users can lose funds when they sell portal energy and choose to add liquidity to the PSM WETH pool.

Attack Scenario

When users sell portal energy, they can choose one of three modes.

```
if (_mode == 0) {
    PSM.safeTransfer(_recipient, amountReceived);
} else if (_mode == 1) {
    addLiquidity(swap);
} else {
    swapOneInch(swap, false);
}
```

If they choose mode == 1, part of their gained PSM tokens is swapped into WETH, and the two tokens are then added into the Ramses WETH-PSM liquidity pool.

The issue is that there can be tokens still remaining in the contract after liquidity addition, which are not reimbursed to the user.

When swapping, the final balances are recorded.

```
swapOneInch(_swap, true);

// code

/// @dev This contract shouldn't hold any tokens, so we pass all tokens.
uint256 PSMBalance = PSM.balanceOf(address(this));
uint256 WETHBalance = WETH.balanceOf(address(this));
```

But the actual balances to be used are calculated again. These actual balances are the ones used for minting LP.

```
(uint256 amountPSM, uint256 amountWETH) = _addLiquidity(PSMBalance,
   WETHBalance, minPSM, minWeth);

// Liq addition
PSM.safeTransfer(pair, amountPSM);
WETH.safeTransfer(pair, amountWETH);
IRamsesPair(pair).mint(_swap.recevier);
```

The issue is that the `_addLiquidity()` function might not use up all the balance, and the remaining will be left in this contract.

The `_addLiquidity()` function tries to calculate the optimum amounts of tokenA and tokenB.

```
uint256 amountBOptimal = quoteLiquidity(amountADesired, reserveA,
   reserveB);

if (amountBOptimal <= amountBDesired) {

if (amountBOptimal < amountBMin) {
    revert ErrorsLib.InvalidAmount();
}
(amountA, amountB) = (amountADesired, amountBOptimal)

// code
```

This is dependent on the current price of WETH-PSM in the Ramses pool, which can fluctuate. This is why the contract also has `amountAMin`, which is a slippage parameter, reverting the minimum is not met.

But if the minimum is met, and if the minimum is lower than the balance in the contract, then there will be a remaining balance in the contract which is not transferred to the user.

Proof of Concept (PoC)

Say Alice gets 100 PSM tokens. Current WETH:PSM price is 1:1

1. Alice passes in parameters to swap 50 PSM to 50 WETH via 1inch with 1% slippage. For liquidity addition, Alice again passes in `amountAmin` and `amountBmin` to 49 each, since setting them to 50 exactly will make the transaction revert any time there is an exchange on the Ramses pool.
2. Alice's 50 PSM gets swapped to 50 WETH. Contract now has 50 WETH, 50 PSM.

3. An attacker changes the price in the Ramses pool, so now 50.5 WETH = 49.5 PSM
4. Contract calculates the optimum liq addition params as 50 WETH and 49.01 PSM
5. The remaining 0.99PSM stays in the contract and is not refunded to Alice
6. Bob then calls any function which calls the 1inch swap, and swaps out the 0.99 PSM in the contract in some shitcoin liquidity pool via 1inch, netting him those tokens.

Location of Affected Code

File: src/AdapterV1.sol#L650-L652

```
PSM.safeTransfer(pair, amountPSM);
WETH.safeTransfer(pair, amountWETH);
IRamsesPair(pair).mint(_swap.recevier);
```

Recommendation

Refund the WETH and PSM balance of the contract after adding liquidity.

Team Response

Fixed as proposed.

[M-01] Anyone Can Call `executeMigration()` Once `migrationDestination` Is Set

Severity

Medium Risk

Reported By

zanderbyte

Description

The current implementation allows anyone to execute the `executeMigration()` function once the `migrationDestination` is set, bypassing the intended logic of requiring a >50% vote based on capital. The function `executeMigration()` is used to mint the Portal NFT and prepare user stakes transfer to a new Adapter. The used checks inside are insufficient.

Attack Scenario

1. The owner set `migrationDestination`
2. `executeMigration()` is called
3. `migrationTime` equals 0, it is only set when this condition is true in the `acceptMigrationDestination()` function:

```
if (votesForMigration > totalPrincipalStaked / 2 && migrationTime == 0) {
    migrationTime = block.timestamp + TIMELOCK;
}
```

so we pass that check:

```
// @dev Ensure that the timelock has passed
if (block.timestamp < migrationTime && migrationTime > 0) {
    revert ErrorsLib.isTimeLocked();
}
```

4. We also pass the next one, it is `false` by default.:

```
// @dev Ensure that the migration (minting of NFT) can only be performed
   once
if (successMigrated == true) {
    revert ErrorsLib.hasMigrated();
}
```

The rest of the function is executed, and s`uccessMigrated` is set to `true` even if the users haven't voted and >50% quorum isn't reached.

Proof of Concept (PoC)

```
// @notice This function mints the Portal NFT and transfers user stakes
   to a new Adapter
// @dev Timelock protected function that can only be called once to move
   capital to a new Adapter
function executeMigration() external isMigrating {
// @dev Ensure that the timelock has passed
    if (block.timestamp < migrationTime  && migrationTime > 0) {
        revert ErrorsLib.isTimeLocked();
    }

// @dev Ensure that the migration (minting of NFT) can only be performed
   once
    if (successMigrated == true) {
        revert ErrorsLib.hasMigrated();
    }

// @dev Mint an NFT to the new Adapter that holds the current Adapter
   stake information
// @dev IMPORTANT: The migration contract must be able to receive ERC721
   tokens
    successMigrated = true;
    totalPrincipalStaked = 0;
    PORTAL.mintNFTposition(migrationDestination);

// @dev Emit migration event
    emit EventsLib.migrationExecuted(migrationDestination);
}
```

```
function testExecuteMigrationWithoutVotes() external {
    help_stake_ETH();
    help_stake_ETH_Bob();
    help_stake_ETH_Karen();

    vm.prank(owner);
    adapter_ETH.proposeMigrationDestination(newAdapterAddr);

    vm.prank(address(1));
    adapter_ETH.executeMigration();
    assertTrue(adapter_ETH.successMigrated());
}
```

Location of Affected Code

File: src/AdapterV1.sol#L123-#L142

Recommendation

Add additional checks for reached quorum or revert if `migrationTime == 0`.

Team Response

Fixed as proposed.

[M-02] Susceptibility to Reorg Attacks

Severity

Medium Risk

Reported By

shealtielanz

Description

Reorgs attacks occur most times on L2 chains, and the `Portal` and `VirtualLP` contracts use the non-deterministic `create` method for deploying the `bToken` and `portalEnergyToken`.

Due to the use of `create` to deploy these contracts, in the case of a reorg event, the addresses will change and users who have used the previous contract's `tokens` or `tokenIDs` will lose them.

Location of Affected Code

File: src/Portal/PortalV2MultiAsset.sol#L458-L465

```
// @dev Deploy the token and update the related storage variable so that
   other functions can work
portalNFT = new PortalNFT(
    DECIMALS_ADJUSTMENT,
    name,
    symbol,
    NFT_META_DATA
);
```

File: src/Portal/VirtualLP.sol#L664-L666

```
// @dev Deploy the token and update the related storage variable so that
   other functions can work.
bToken = new MintBurnToken(name, symbol);
bTokenAddress = address(bToken);
```

Recommendation

Consider using `create2` with a non-deterministic `salt + the msg.sender` instead.

Team Response

Acknowledged, the fix will be considered.

[M-03] Parameters and Return Values Mismatch in Contracts and Interfaces

Severity

Medium Risk

Reported By

zanderbyte

Description

- There is a mismatch between contract and interface function return values. In contract PortalV2MultiAsset:

```
function getUpdateAccount(address _user, uint256 _amount, bool
   _isPositiveAmount) public view returns (
@>  uint256 lastUpdateTime,
    uint256 lastMaxLockDuration,
    uint256 stakedBalance,
    uint256 maxStakeDebt,
    uint256 portalEnergy,
    uint256 availableToWithdraw,
    uint256 portalEnergyTokensRequired
)
```

In interface IPortalV2MultiAsset :

```
function getUpdateAccount(address _user, uint256 _amount, bool
    _isPositiveAmount) external view returns (
@>   address user,
     uint256 lastUpdateTime,
     uint256 lastMaxLockDuration,
     uint256 stakedBalance,
     uint256 maxStakeDebt,
     uint256 portalEnergy,
     uint256 availableToWithdraw
);
```

- There is a mismatch between contracts and interfaces function parameters and return values. In contract AdapterV1 :

```
function getUpdateAccount(
     address _user,
     uint256 _amount,
@>   bool _isPositiveAmount
     ) public view returns (
@>   uint256 lastUpdateTime,
     uint256 lastMaxLockDuration,
     uint256 stakedBalance,
     uint256 maxStakeDebt,
     uint256 portalEnergy,
     uint256 availableToWithdraw,
     uint256 portalEnergyTokensRequired
)
```

```
function stake(uint256 _amount) external payable notMigrating
    nonReentrant
```

```
function addLiquidity(SwapData memory _swap) internal
```

In interface IAdapterV1:

```
function getUpdateAccount(
    address _user,
    uint256 _amount
@>   ) external view returns (address, uint256, uint256, uint256, uint256,
     uint256, uint256);
```

```
function stake(
@>   address _receiver,
     uint256 _amount) external;
```

```
function addLiquidity(
    address _receiver,
    uint256 amountPSMDesired,
    uint256 amountWETHDesired,
    uint256 amountPSMMin,
    uint256 amountWETHMin,
    uint256 _deadline
) external returns (uint256 amountPSM, uint256 amountWETH, uint256
   liquidity);
```

Impact

Even though the contract `PortalV2MultiAsset` itself isn't within the scope of this audit, the interface `IPortalV2MultiAsset` reveals a mismatch between return values. This mismatch between parameters and return values can potentially lead to unexpected errors since anyone who uses these interfaces will expect to receive and provide different values.

Location of Affected Code

File: src/Portal/PortalV2MultiAsset.sol#L204

File: src/interfaces/IPortalV2MultiAsset.sol#L5

File: src/AdapterV1.sol#L186

File: src/interfaces/IAdapterV1.sol#L101

File: src/interfaces/IAdapterV1.sol#L33

File: src/AdapterV1.sol#L325

Recommendation

Apply the necessary changes in the mentioned interfaces so that definitions and implementations fully match.

Team Response

Fixed as proposed.

[L-01] Remaining Tokens Should Be Sent to `_swap.recevier()` Instead of `msg.sender` in `swapOneInch()` Function

Severity

Low Risk

Reported By

EgisSec

Description

When calling `swapOneInch()` we specify `swapData()` and `_forLiquidity()`. Inside `swapData()` we specify a `receiver` who will receive the tokens.

There is a case when if `_forLiquidity = false` we check if there are any remaining PSM tokens by subtracting `spentAmount_` from `_swap.psmAmount`.

If there are any remaining tokens, they are sent to `msg.sender`. Considering we are specifying a `swapData.receiver`, if there are any remaining PSM tokens, they should be sent to `swapData.receiver`.

Impact

Inconvenience and potential gas costs, as `msg.sender` has to transfer the tokens to `swapData.receiver`.

Location of Affected Code

File: AdapterV1.sol#L599

Recommendation

Consider implementing the following change:

```
- if (remainAmount > 0) PSM.safeTransfer(msg.sender, remainAmount);
+ if (remainAmount > 0) PSM.safeTransfer(_swap.recevier, remainAmount);
```

Team Response

Fixed as proposed.

[L-02] `totalPrincipalStaked` Is Not Decreased When NFT Is Minted

Severity

Low Risk

Reported By

MaslarovK

Description

In the `AdapterV1::acceptMigrationDestination()`, `totalPrincipalStaked` is not decreased.

Location of Affected Code

File: AdapterV1.sol#L117-L135

Recommendation

Set the `totalPrincipalStaked` to 0.

Team Response

Fixed as proposed.

[L-03] `acceptMigrationDestination()` Can Be Called Multiple Times to Brick Migration Clearing All the Funds in the New Adapter

Severity

Low Risk

Reported By

shealtielanz

Description

The `acceptMigrationDestination()` function can be called multiple times by an attacker to clear all the funds the new adapter has in the PORTAL. In case the votes are enough, the PORTAL mints the migration destination an NFT holding all new current stakes of the current adapter.

The issue is that on the call to `PORTAL::mintNFTposition()`, the account of the `msg.sender` is deleted and when `portalNFT.mint()` is called it sets the balance of the recipient to the current stakedBalance, thereby clearing all the previous funds of the new adapter.

Therefore, users unaware can call the function again even after migration has occurred.

Location of Affected Code

File: AdapterV1.sol#L109

Recommendation

Ensure that once the new migration has been accepted the `acceptMigrationDestination()` cannot be called again:

```
function acceptMigrationDestination() external isMigrating {
+    require(!successMigrated, "migration has already occurred");
// code
+    successMigrated = true;
}
```

Team Response

Fixed by adding the flag and the check in `executeMigration()` function.

[L-04] Missing Recipient Address Validation in `sellPortalEnergy()` Function of `AdaptersV1` Contract

Severity

Low Risk

Reported By

DevPelz

Description

The `sellPortalEnergy()` function in the `AdaptersV1` contract lacks validation for the recipient address, which can lead to serious vulnerabilities. Depending on the selected mode, this function transfers PSM tokens or adds liquidity to a pair, potentially allowing unauthorized transfers to `address(0)` or other unintended recipients.

Impact

This vulnerability can result in unauthorized transfers of PSM tokens to `address(0)` or other unintended recipients. Additionally, if the mode selected is 1, the function adds liquidity to a pair, which could result in the minting of Ramses pair tokens to an unauthorized recipient, potentially causing significant damage or financial loss.

Location of Affected Code

File: src/AdapterV1.sol#L548

File: src/AdapterV1.sol#L555

File: src/AdapterV1.sol#L590

File: src/AdapterV1.sol#L652

Recommendation

To mitigate this vulnerability, implement validation checks for the recipient address in the `sellPortalEnergy()` function. Ensure that the recipient address is not set to `address(0)` or any other unauthorized recipient before proceeding with token transfers or liquidity additions.

Team Response

Fixed.

[L-05] `increaseAllowances()` Can Not Be Called Again Until the Entire Allowance of `principalToken` ' Is Used

Severity

Low Risk

Reported By

CaeraDenoir

Description

Possible DoS depending on the amount of portal tokens in existence (future risk). This is due to `increaseAllowances()` increasing the allowance of the principal token with `safeIncreaseAllowance()` , which will revert if the allowance is not 0.

This is due to the `safeIncreaseAllowance()` itself and how it works since it has no overflow prevention.

The function `increaseAllowances()` passes the maximum amount possible for a `uint256` , which makes the entire `increaseAllowances()` revert if the oldAllowance does not equal 0. Not allowing `PSM` and `portalEnergyToken` to get their allowances increased after the first time. There is a comment stating `For ERC20 that require allowance to be 0 before increasing (e.g. USDT) add the following:` , but regardless of the token itself, the function will revert due to the SafeERC20 `safeIncreaseAllowance()` implementation.

It should be noted that if a lot of `portalEnergyToken` tokens are able to be minted, there is an easy way to deplete the allowance in the `AdapterV1` contract by calling `burnPortalEnergyToken()` and `mintPortalEnergyToken()` on loop.

Impact

The impact on the `AdapterV1.sol` is high, but highly improbable in current conditions. Any inflation on the `portalEnergyToken` total supply (regardless if it is the current implementation or a different portal contract) would make the attack viable.

Location of Affected Code

File: src/AdapterV1.sol#L771

Recommendation

Regardless of the requirement of the allowance being 0, the `principalToken.approve(address(PORTAL), 0)` should not be commented.

Team Response

Fixed as proposed.

[L-06] The Owner Can Bypass the Voting Process to Steal All Users Funds

Severity

Low Risk

Reported By

elhajin

Description

The `AdapterV1` contract allows the owner to unilaterally migrate all staked funds to a new contract without going through the voting process.

Impact

The owner can bypass the intended democratic migration process, which requires a majority of staked capital to vote for migration. By exploiting this vulnerability, the owner can redirect all staked funds and non-claimed portal energy to a malicious contract, effectively stealing assets from users.

Attack Scenario

1. The owner takes a flash loan of the principal token, acquiring an amount greater than the **total staked balance** in the `AdapterV1` contract.
2. The owner stakes the borrowed tokens through the `AdapterV1` contract, gaining a majority stake (more than 50%).
3. The owner calls `proposeMigrationDestination()` with a malicious contract address / personal address.
4. The owner then calls `acceptMigrationDestination()` to accept the migration to the malicious contract.
5. The NFT representing the staked position in the `PortalV2MultiAsset` contract is minted to the owner, who can now redeem it for all users' funds and then repay the flash loan.

The owner does not need to interact with the old adapter anymore since they control the NFT in the portal itself, which holds the staked funds.

- NOTE: All of the steps can be done in one transaction.

Proof of Concept

```
function test_PoC() public {
// Alice stakes some tokens
    uint256 amount = 1e9;
    help_setAllowances();
    vm.startPrank(alice);
    principal_USDC.approve(address(adapter_USDC), amount);
    adapter_USDC.stake(amount);
    vm.stopPrank();
```

```
// simulate taking flashloan by the owner
    address owner = adapter_USDC.OWNER();
    address richWallet = 0xd89b79f10523119d5B467d43EFe0A7710AE2d2AB;
    vm.prank(richWallet);
    principal_USDC.transfer(owner, 1e10);
    vm.startPrank(owner);
    console.log("balance  staked before attack : %e", 1e10);
    principal_USDC.approve(address(adapter_USDC), 1e10);
    adapter_USDC.stake(1e10);
// propose an address to migrate
    adapter_USDC.proposeMigrationDestination(owner);
// vote for this address
    adapter_USDC.acceptMigrationDestination();
// reedem the nft from the portal
    portal_USDC.redeemNFTposition(1);
// unstake
    (,, uint256 stakedBalance,,,,) = portal_USDC.getUpdateAccount(owner,
       0, true);
    console.log("balance  after attack : %e", stakedBalance);
}
```

- Logs:

```
Logs:
//balance  staked before attack: 1e10
//balance  after attack: 1.1e10
```

- In this scenario, the owner steals all of Alice's funds without going through a voting process.

Location of Affected Code

File: src/AdapterV1.sol#L108

File: src/AdapterV1.sol#L117

Recommendation

Implement a Timelock mechanism to prevent immediate migration after a proposal, allowing users time to review and respond to proposed changes.

Team Response

Fixed by implementing a 7-day Timelock.

[L-07] Staking Dependency Reduces User Influence in Migration

Severity

Low Risk

Reported By

shealtielanz

Description

The owner proposes a new migration destination, users vote to accept it or remain with the current adapter but after an owner has proposed a new adapter there's no going back due to the `notMigrating()` and `isMigrating()` modifiers and how they are used on some important functions.

The migration destination address can't be changed and therefore the current adapter will not be able to be used again. The users can never be able to `stake()` in the current adapter even if they do not want to migrate to a new adapter, this will force them to vote for the new migration destination.

This is due to the modifiers placed on the respective functions outlined above, causing the calls to those functions to revert when a migration destination has been proposed but was not voted for by the Possum users.

Location of Affected Code

File: src/AdapterV1.sol#L88-L100

```
modifier notMigrating() {
    if (migrationDestination != address(0)) {
        revert ErrorsLib.isMigrating();
    }
    _;
}

modifier isMigrating() {
    if (migrationDestination == address(0)) {
        revert ErrorsLib.notMigrating();
    }
    _;
}
```

Proof of Concept

Add to AdapterV1Test.t.sol:

```
function test_Proposed_MigrationDestination_Rejected_By_Users() external
   {
    help_stake_ETH();
// The owner proposes a new migration
    vm.prank(owner);
    adapter_ETH.proposeMigrationDestination(newAdapterAddr);

    assertEq(adapter_ETH.migrationDestination(), newAdapterAddr);
```

```
// The user doesn't want to migrate so don't vote even after 3 months
    vm.roll(3 * 30);
    assertEq(adapter_ETH.votesForMigration(), 0);

// Stake reverts
    vm.deal(alice, 1e19);
    vm.expectRevert();
    vm.prank(alice);
    adapter_ETH.stake{value: 1e19}(1e19);

// The owner can't reset the migration destination to solve the issue
    vm.expectRevert();
    vm.prank(owner);
    adapter_ETH.proposeMigrationDestination(newAdapterAddr);
}
```

Recommendation

Consider setting a duration for voting and if users do not accept the proposed migration destination during the stipulated period, because in a scenario where the users refuse to vote for the new migration contract the adapter and all its funds will be DoSed since it's not possible to migrate and it is not possible to use the current adapter.

Team Response

Fixed by removing the `notMigrating()` modifier from the `sellPortalEnergy()` function.

[L-08] Incorrect Event Emission in `buyPortalEnergy()` Function

Severity

Low Risk

Reported By

DevPelz

Description

The `buyPortalEnergy()` function in the codebase has a discrepancy in event emission. The event emission within the function is not consistent with the original declaration of the event. Specifically, the event emission utilizes `msg.sender` twice instead of the recipient address as specified by the event.

Impact

This inconsistency may lead to confusion and incorrect data representation within the system. It could potentially mislead stakeholders relying on event logs for tracking transactions and auditing purposes.

Location of Affected Code

File: src/AdapterV1.sol#L476-L481

```
function buyPortalEnergy(
    address _recipient,
    uint256 _amountInputPSM,
    uint256 _minReceived,
    uint256 _deadline
) external notMigrating {
// code

// @dev Emit the event that Portal Energy has been purchased
    emit EventsLib.AdapterEnergyBuyExecuted(
        msg.sender,
        msg.sender,
        amountReceived
    );
}
```

Recommendation

Amend the event emission within the `buyPortalEnergy()` function to align with the original event declaration:

```
emit EventsLib.AdapterEnergyBuyExecuted(
    msg.sender,
-   msg.sender,
+   _recipient,
    amountReceived
);
```

Team Response

Fixed as proposed.

[L-09] There Is No Function to Withdraw the Funds Which is Received from `receive()` or `fallback()` Function

Severity

Low Risk

Reported By

Atharv181

Description

The smart contract lacks a function to withdraw funds received through the `receive()` or `fallback()` function, potentially leading to a loss of funds if they cannot be retrieved.

Impact

The funds will be stuck in the contract, as there is no way to withdraw them.

Location of Affected Code

File: src/AdapterV1.sol#L682

Recommendation

Implement a function that allows the contract owner or authorized users to withdraw funds stored in the contract. This function should include appropriate access controls and validation checks to prevent unauthorized withdrawals. Additionally, consider adding event logging to track withdrawal activities for transparency purposes.

Team Response

Fixed by implementing a refund function - `salvageToken()`.

[L-10] Inconsistent Input Validation in `burnPortalEnergyToken()` Function

Severity

Low Risk

Reported By

DevPelz

Description

The `burnPortalEnergyToken()` function in the `AdaptersV1` contract claims to rely on input validation from the Portal but does not enforce it. Specifically, the function does not prevent the recipient address from being set to `address(0)`, although the input validation for this check is implemented in the portal contract.

Impact

This vulnerability could lead to unexpected behavior or potential misuse of the `burnPortalEnergyToken()` function. Allowing the recipient address to be set to `address(0)` without validation increases the risk of unauthorized token burns or unintended consequences.

Attack Scenario

An attacker could exploit this vulnerability by calling the `burnPortalEnergyToken()` function with `address(0)` as the `recipient`, bypassing the intended validation check in the portal contract. This could result in unauthorized token burns or manipulation of the portal's internal state.

Location of Affected Code

File: src/AdapterV1.sol#L711

Recommendation

To mitigate this vulnerability, implement the same input validation check for the `recipient` address in the `AdaptersV1` contract as in the `Portal` contract. This ensures consistency and adherence to requirements and specifications. The check should verify that the `recipient` address is not set to `address(0)` before proceeding with the token burn operation.

Team Response

Fixed as proposed.

[L-11] Ensure Sanity Checks on the `swapDescription.flags`

Severity

Low Risk

Reported By

shealtielanz

Description

The `flags` of the `SwapDescription` struct determine the process of a swap on `1inch` and it is not checked possibly causing incorrect swap types to be executed on `1inch`.

On the call to `swapOneInch()` the protocol intends that all `swaps` are performed with the full amount specified for the `swap`, without any amount refunded to the `adapter` after the `swap`, however, this behavior is not constrained allowing for partial `swaps` to be performed via `1inch`.

The issue is that the `SwapDescription.flag` determines whether the swap will be a full or partial swap, but the flag is not checked before the call to `ONE_INCH_V6_AGGREGATION_ROUTER.swap()` allowing for partial swaps to occur on the call to `swapOneInch()`.

Location of Affected Code

File: AdapterV1.sol#L495-L509

```
function swapOneInch(address _caller, SwapData memory _swap, bool
    _forLiquidity) internal {
// code

// @dev Swap via the 1Inch Router
// @dev Allowance is increased in a separate function to save gas
@>  (, uint256 spentAmount_) = ONE_INCH_V6_AGGREGATION_ROUTER.swap(
    IAggregationExecutor(_executor), _description, _data);
```

Recommendation

Check the `flags` of the `SwapDescription` to ensure against partial swaps.

Team Response

Fixed as proposed.

[L-12] Attempting to Vote After 50%+ Have Been Reached Will Always Revert

Severity

Low Risk

Reported By

PlamenTSV

Description

The `acceptMigrationDestination()` function is meant to be callable by anyone in order to vote for the new migration. The voting mechanism works by minting the NFT for the current Adapter state when \codex{50%+} quorum is reached, which is fine since staking is locked. However, voting is still enabled after reaching quorum, even though it will always revert.

Impact

People who have not yet voted but want to do so might encounter unexpected reversions when attempting to vote. It's essential to provide users with the opportunity to achieve a quorum of 70-80% for feedback purposes on the protocol.

Attack Scenario

Any attempts to vote inside `acceptMigrationDestination()` when a quorum has been reached would lead to a revert. This happens because attempting to call `PORTAL.mintNFTposition(migrationDestination)` a second time would revert since the account inside the portal is already deleted, so it will revert to being empty.

Location of Affected Code

File: Portal/PortalV2MultiAsset.sol#L472-L494

File: AdapterV1.sol#L129-L134

Recommendation

Consider the following improvements:

1. Allowing users to vote for a higher than 50%+1 quorum, which could act as a good feedback loop for the protocol to know what migrations are liked by users
2. Add sufficient events, since they are currently missing.

3. Add a safeguard check that would return/revert early like:
   `if(successMigrated == true) return/revert` and emit a necessary event for the user to know This way the votes variable would increase, but it will not reattempt minting.

Team Response

Acknowledged.

shieldify

Thank you!