SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Mover
Prepared by: Sherlock
Lead Security Expert: WATCHPUG
Dates Audited: October 18 - October 25, 2022
Prepared on: November 22, 2022

Introduction

Mover is a permissionless protocol exploring metaverse savings. It is a suite of products in NFT, web3, and DeFi space to create a new open savings experience.

Scope

The single goal of the contracts is to get user funds (native token or ERC-20 token), swap it to USDC (PoS USDC on Polygon) and bridge it to specified static address on L1 Eth, on which user debit card settlement would be initiated.

```
ExchangeProxy.sol
RLPReader.sol
SafeAllowanceReset.sol
ByteUtil.sol
SafeAllowanceResetUpgradeable.sol
HardenedTopupProxy.sol
ContractWhitelist.sol
```

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.

Issues found

Medium	High
1	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

WATCHPUG
GalloDaSballo
minhquanym

0x52
Jeiwan
hansfriese

berndartmueller

Issue H-1: Attacker can steal the accumulated topup fees in the `topupproxy` contract's balance

Source: https://github.com/sherlock-audit/2022-10-mover-judging/issues/112

Found by

minhquanym, Jeiwan, 0x52, hansfriese, WATCHPUG, GalloDaSballo, berndartmueller

Summary

The accumulated fees in the `topupproxy` contract's balance can be stolen by an attacker by using malicious `_bridgeTxData` and using `1inch`'s as `targetAddress`.

Vulnerability Detail

This attack vector is enabled by multiple traits of the `topupproxy` contract:

1. Shared whitelist Per to deploy script, the same `trustedregistry` will be shared among `exchangeproxy` and `topupproxy`.

Therefore, the 2 whitelisted swap aggregator contracts will also be allowed to be called on `topupproxy`:

- 0x Proxy
- 1inch Proxy

And the 2 whitelisted bridge contracts can be called on `exchangeproxy`:

- Synapse
- Across

2. Unlimited allowance rather than only the amount of the current topup to the bridge's `targetAddress` At L414, the `targetAddress` will be granted an unlimited allowance rather than just the amount of the current transaction.

https://github.com/sherlock-audit/2022-10-mover/blob/main/cardtopup_contract/contracts/HardenedTopupProxy.sol#L414

3. `1inch` can be used to pull an arbitrary amount of funds from the caller and execute arbitrary call The design of `1inch`'s `AggregationRouterV4` can be used to pull funds from the `topupproxy` and execute arbitrary external call:

https://polygonscan.com/address/0x1111111254fb6c44bAC0beD2854e76F90643097d#code

See L2309-2321.

4. The topup fee will be left in the contract's balance https://github.com/sherlock-audit/2022-10-mover/blob/main/cardtopup_contract/contracts/HardenedTopupProxy.sol#L348-L352

Combining all the 3 above together, the attacker can call `CardTopupPermit()->_processTopup()->1inchswap()` and drain all the funds in the contract:

- _token: cardTopupToken
- _bridgeType: 0
- _bridgeTxData:
  - targetAddress: 1inch Proxy
  - callData:
    * amount: all the topupproxy's balance
    * srcReceiver: attacker's address

Impact

All the accumulated fees can be stolen by the attacker.

Code Snippet

https://polygonscan.com/address/0x1111111254fb6c44bAC0beD2854e76F90643097d#code

https://github.com/sherlock-audit/2022-10-mover/blob/main/cardtopup_contract/contracts/HardenedTopupProxy.sol#L348-L352

https://github.com/sherlock-audit/2022-10-mover/blob/main/cardtopup_contract/migrations/5_connect_contracts.js#L54-L61

Tool used

Manual Review

Recommendation

1. The accumulated fees should not be left in the contract;
2. Only give the whitelisted `targetAddress` the allowance of the amount (`_amount`) transferred into the `topupproxy` contract within this transaction from the caller;

3. The whitelist should not be shared.

Discussion

McMannaman

Duplicate of https://github.com/sherlock-audit/2022-10-mover-judging/issues/60 https://github.com/sherlock-audit/2022-10-mover-judging/issues/37 https://github.com/sherlock-audit/2022-10-mover-judging/issues/38 https://github.com/sherlock-audit/2022-10-mover-judging/issues/52

although this is the best and most comprehensive of them all.

I think that it's a low vulnerability (user funds are not affected by this and fees are harvested from time to time anyway in the normal flow of operation). But, regardless -- this issue has a valid point.

hrishibhat

Hey @McMannaman, since #38 is also considered a duplicate, which is considered medium. Shouldn't the rest of the issues be medium too?

amozgov

@hrishibhat correct, as @McMannaman mentioned - this is not a "high" vulnerability since no user funds are at risk, there is a tag "disagree with severity"

Evert0x

We will not change the severity of this issue as protocol funds are at risk.

McMannaman

The fixes are in https://github.com/viaMover/2022-10-mover/pull/1

jacksanford1

Bringing over some comments from https://github.com/viaMover/2022-10-mover/pull/1

McMannaman Added reentrancy protection (also for issue #120) Plus an additional check that only the USDC amount expected is deducted from contract when bridging regardless of bytes call data. https://github.com/sherlock-audit/2022-10-mover-judging/issues/112

WatchPug

1. It's better to ensure that the whitelist is not shared between the two contracts. Otherwise, the attacker can still steal the topup fee from HardenedTopupProxy by using 1inch as targetAddress in their _bridgeTxData. Can you also make the changes required to the deploy script to reflect that?

2. Seems like the attacker can still steal the exchange fee sitting on the exchangeProxyContract.

McMannaman

1. I have updated the migrations to reflect that whitelists would be separated (and 2 child contracts just to keep migrations-compatible).
2. Could you please elaborate on how the attacker could steal exchange fee on the exchangeProxyContract? The fees are (if they would be non-zero) in USDC-only (the target token would be USDC), or, more generally in some single token, fees could be claimed before token change, before, e.g. hypothetically, to USDT. And if we know the target token, then lines https://github.com/viaMover/2022-10-mover/blob/fix-112-reentrancyamountcheck/cardtopup_contract/contracts/HardenedTopupProxy.sol#L443 and https://github.com/viaMover/2022-10-mover/blob/fix-112-reentrancyamountcheck/cardtopup_contract/contracts/ExchangeProxy.sol#L198 should protect from draining fees:

(non-swap scenario):

1. an amount is stated as parameter when calling topup, then that amount is transferred to the Topup proxy;
2. no swap is called;
3. bridged amount is checked to exactly match provided amount (regardless of what is provided/called in the bridge data/call);

(swap scenario):

1. an amount is stated as parameter when calling topup, then that amount is transferred to the Topup proxy;
2. swap is called, the actual received amount in USDC is now the amount we're working with (regardless of what is provided/called in the bridge data/call) -- deducting fees on both proxies;
3. bridged amount is checked to exactly match amount stated by Exchange proxy (regardless of what is provided/called in the bridge data/call);

so there are several assumptions we're working with:

- fees are collected in single token type (otherwise they can be stolen, yes);
- exchange proxy is callable only by Transfer proxy (a require https://github.com/viaMover/2022-10-mover/blob/fix-112-reentrancyamountcheck/cardtopup_contract/contracts/ExchangeProxy.sol#L151);
- if user uses some manipulation to escape (avoid paying own) fees (don't know how this is achievable though without reentrancy) -- this is violation of terms of use, even if possible, should be of little rationale to user;

Please point if I'm missing something (no code examples needed, just a description would be enough).

@jack-the-pug

WatchPug

- fees are collected in single token type (otherwise they can be stolen, yes);

Yeah, I think this is the case where the accumulated fees on the `exchangeProxyContract` can be stolen.

I agree that this is not a major risk, though.

Issue M-1: `exchangeFee` can be escaped

Source: https://github.com/sherlock-audit/2022-10-mover-judging/issues/120

Found by

WATCHPUG

Summary

Comparing the before and after balance of the swap call for the swapped amount can be exploited to escape the `exchangeFee` by wrapping the actual swap inside a fake swap.

Vulnerability Detail

The attacker can reenter with another `CardTopupPermit()->_processTopup()->IExchangeProxyexecuteSwapDirect()` at L174 to claw back the fee:

1. Swap minAmount with 1inch, inside the 1inch swap at `ExchangeProxy.solL174`, reenter and `HardenedTopupProxy.solCardTopupPermit();`

2. The inner swap is the actual amount: $1M, which should pay for$

As a result, the user successfully escaped most of the `exchangeFee`.

Impact

User can escape the `exchangeFee`.

Code Snippet

https://github.com/sherlock-audit/2022-10-mover/blob/main/cardtopup_contract/contracts/HardenedTopupProxy.sol#L336-L343

https://github.com/sherlock-audit/2022-10-mover/blob/main/cardtopup_contract/contracts/ExchangeProxy.sol#L160-L185

Tool used

Manual Review

Recommendation

Consider adding `nonReentrant()` modifier to all the 3 non-view methods in the `HardenedTopupProxy`:

- CardTopupPermit();
- CardTopupTrusted();
- CardTopupMPTProof().

Discussion

McMannaman

This is a valid point. Would be fixed by adding nonReentrant modifiers.

jack-the-pug

Fix confirmed

McMannaman

The fixes are in https://github.com/viaMover/2022-10-mover/pull/2