SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Telcoin
Prepared by:	Sherlock
Lead Security Expert:	mstpr-brainbot
Dates Audited:	January 12 - January 15, 2024
Prepared on:	February 22, 2024

Introduction

Telcoin creates low-cost, high-quality financial products for every mobile phone user in the world.

Scope

Repository: telcoin/telcoin-audit

Branch: main

Commit: 83b50e15be708dffec5d96695ab187348b52f2c1

For the detailed scope, see the contest details.

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
2	3

Issues not fixed or acknowledged

Medium	High
0	0

Issue H-1: `StakingRewardsManager::topUp(...)` Misallocates Funds to `StakingRewards` Contracts

Source: https://github.com/sherlock-audit/2024-01-telcoin-judging/issues/16

Found by

Aamirusmani1552, Arz, DMoore, IvanFitro, VAD37, fibonacci, ggg_ttt_hhh, ravikiran.web3, sakshamguruji, zzykxx

Summary

The `StakingRewardsManager::topUp(...)` contract exhibits an issue where the specified `StakingRewards` contracts are not topped up at the correct indices, resulting in an incorrect distribution to different contracts.

Vulnerability Detail

The `StakingRewardsManager::topUp(...)` function is designed to top up multiple `StakingRewards` contracts simultaneously by taking the indices of the contract's addresses in the `StakingRewardsManager::stakingContracts` array. However, the flaw lies in the distribution process:

```
    function topUp(
        address source,
@>        uint256[] memory indices
    ) external onlyRole(EXECUTOR_ROLE) {
@>        for (uint i = 0; i < indices.length; i++) {
            // get staking contract and config
            StakingRewards staking = stakingContracts[i];
            StakingConfig memory config = stakingConfigs[staking];

            // will revert if block.timestamp <= periodFinish
            staking.setRewardsDuration(config.rewardsDuration);

            // pull tokens from owner of this contract to fund the staking
 ↪  contract
            rewardToken.transferFrom(
                source,
                address(staking),
                config.rewardAmount
            );

            // start periods
            staking.notifyRewardAmount(config.rewardAmount);
            emit ToppedUp(staking, config);
        }
    }
```

GitHub: [254-278]

The rewards are not appropriately distributed to the `StakingRewards` contracts at the specified indices. Instead, they are transferred to the contracts at the loop indices. For instance, if intending to top up contracts at indices `[1, 2]`, the actual top-up occurs at indices `[0, 1]`.

Impact

The consequence of this vulnerability is that rewards will be distributed to the incorrect staking contract, leading to potential misallocation and unintended outcomes

Code Snippet

Here is a test for PoC:

Add the below given test in `StakingRewardsManager.test.ts` File. And use the following command to run the test

```
npx hardhat test --grep "TopUp is not done to intended staking rewards contracts"
```

TEST:

```
it("TopUp is not done to intended staking rewards contracts", async function () {
    // add index 2 to indices
    // so topup should be done to index 0 and 2
    indices = [0, 2];

    await rewardToken.connect(deployer).approve(await
↪   stakingRewardsManager.getAddress(), tokenAmount * indices.length);

    // create 3 staking contracts
    await stakingRewardsManager.createNewStakingRewardsContract(await
↪   stakingToken.getAddress(), newStakingConfig);
    await stakingRewardsManager.createNewStakingRewardsContract(await
↪   stakingToken.getAddress(), newStakingConfig);
    await stakingRewardsManager.createNewStakingRewardsContract(await
↪   stakingToken.getAddress(), newStakingConfig);

    // topup index 0 and 2
    await expect(stakingRewardsManager.connect(deployer).topUp(await
↪   deployer.address, indices))
        .to.emit(stakingRewardsManager, "ToppedUp");


    // getting the staking contract at index 0, 1 and 2
    let stakingContract0 = await stakingRewardsManager.stakingContracts(0);
    let stakingContract1 = await stakingRewardsManager.stakingContracts(1);
    let stakingContract2 = await stakingRewardsManager.stakingContracts(2);

    // Staking contract at index 2 should be empty
    expect(await rewardToken.balanceOf(stakingContract2)).to.equal(0);

    // Staking contract at index 0 and 1 should have 100 tokens
    expect(await rewardToken.balanceOf(stakingContract0)).to.equal(100);
    expect(await rewardToken.balanceOf(stakingContract1)).to.equal(100);

});
```

Output:

```
AAMIR@Victus MINGW64 /d/telcoin-audit/telcoin-audit (main)
$ npx hardhat test --grep "TopUp is not done to intended staking rewards
↪   contracts"


  StakingRewards and StakingRewardsFactory
    topUp
       TopUp is not done to intended staking rewards contracts (112ms)


  1 passing (2s)

```

Tool used

- Manual Review

Recommendation

It is recommended to do the following changes:

```
    function topUp(
        address source,
        uint256[] memory indices
    ) external onlyRole(EXECUTOR_ROLE) {
        for (uint i = 0; i < indices.length; i++) {
            // get staking contract and config
-             StakingRewards staking = stakingContracts[i];
+           StakingRewards staking = stakingContracts[indices[i]];
            StakingConfig memory config = stakingConfigs[staking];

            // will revert if block.timestamp <= periodFinish
            staking.setRewardsDuration(config.rewardsDuration);

            // pull tokens from owner of this contract to fund the staking
 ↪  contract
            rewardToken.transferFrom(
                source,
                address(staking),
                config.rewardAmount
            );

            // start periods
            staking.notifyRewardAmount(config.rewardAmount);

            emit ToppedUp(staking, config);
        }
    }
```

Discussion

amshirif

https://github.com/telcoin/telcoin-audit/pull/27

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

takarez commented:

> valid because { I consider this a high severity and avalid issues; the watson was able to explain how the topUp function will perform an unintended actions by topping up from the 0 index of the array always due to lack of good implementation of the indices that was supposed to be added before the (i) }

nevillehuang

@amshirif Will this allow the stakers of the wrong contract funded to retrieve unintended rewards? If yes I will remain as high severity.

amshirif

@nevillehuang Yes this would potentially cause those who should have gotten rewards to have received less or non at all, and those who were not intended to get any or less than their desired amount to get more than they should have.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/telcoin/telcoin-audit/pull/27.

sherlock-admin

The Lead Senior Watson signed-off on the fix.

Issue H-2: Wrong parameter when retrieving causes a complete DoS of the protocol

Source: https://github.com/sherlock-audit/2024-01-telcoin-judging/issues/139

Found by

0xadrii, Arz, Tricko, eeshenggoh, fibonacci

Summary

A wrong parameter in the `_retrieve()` prevents the protocol from properly interacting with Sablier, causing a Denial of Service in all functions calling `_retrieve()`.

Vulnerability Detail

The `CouncilMember` contract is designed to interact with a Sablier stream. As time passes, the Sablier stream will unlock more TELCOIN tokens which will be available to be retrieved from `CouncilMember`.

The `_retrieve()` internal function will be used in order to fetch the rewards from the stream and distribute them among the Council Member NFT holders (snippet reduced for simplicity):

```
// CouncilMember.sol

function _retrieve() internal {
        ...
        // Execute the withdrawal from the _target, which might be a Sablier
 ↪  stream or another protocol
        _stream.execute(
            _target,
            abi.encodeWithSelector(
                ISablierV2ProxyTarget.withdrawMax.selector,
                _target,
                _id,
                address(this)
            )
        );

        ...
    }
```

The most important part in `_retrieve()` regarding the vulnerability that we'll dive into is the `_stream.execute()` interaction and the params it receives. In order to understand such interaction, we first need understand the importance of the `_stream` and the `_target` variables.

Sablier allows developers to integrate Sablier via Periphery contracts, which prevents devs from dealing with the complexity of directly integrating Sablier's Core contracts. Telcoin developers have decided to use these periphery contracts. Concretely, the following contracts have been used:

- ProxyTarget (link points to an older commit because **the proxy target contracts have now been deprecated from Sablier**): stored in the `_target` variable, this contract acts as the target for a PRBProxy contract. It contains all the complex interactions with the underlying stream. Concretely, Telcoin uses the `[withdrawMax()](https://github.com/sablier-labs/v2-periphery/blob/ba3926d2c3e059a230211077087b73afe46acf64/src/abstracts/SablierV2Proxy` `L143C6)` function in the proxy target to withdraw all the available funds from the stream (as seen in the previous code snippet).
- PRBProxy: stored in the `_stream` variable, this contract acts as a forwarding (non-upgradable) proxy, acting as a smart wallet that enables multiple contract calls within a single transaction.

  NOTE: It is important to understand that the actual lockup linear stream will be deployed as well. The difference is that the Telcoin protocol will not interact with that contract directly. Instead, the PRBProxy and proxy target contracts will be leveraged to perform such interactions.

Knowing this, we can now move on to explaining Telcoin's approach to withdrawing the available tokens from the stream. As seen in the code snippet above, the `_retrieve()` function will perform two steps to actually perform a withdraw from the stream:

It will first call the `_stream`'s `execute()` function (remember `_stream` is a PRBProxy). This function receives a `target` and some `data` as parameter, and performs a delegatecall aiming at the `target`:

```
// https://github.com/PaulRBerg/prb-proxy/blob/main/src/PRBProxy.sol

/// @inheritdoc IPRBProxy
   function execute(address target, bytes calldata data) external payable
↪  override returns (bytes memory response) {
        ...

        // Delegate call to the target contract, and handle the response.
        response = _execute(target, data);
    }
    /*//////////////////////////////////////////////////////////////////////////
                           INTERNAL NON-CONSTANT FUNCTIONS
    //////////////////////////////////////////////////////////////////////////*/

    /// @notice Executes a DELEGATECALL to the provided target with the provided
 ↪  data.
    /// @dev Shared logic between the constructor and the `execute` function.
    function _execute(address target, bytes memory data) internal returns (bytes
 ↪  memory response) {
        // Check that the target is a contract.
        if (target.code.length == 0) {
            revert PRBProxy_TargetNotContract(target);
        }

        // Delegate call to the target contract.
        bool success;
        (success, response) = target.delegatecall(data);

        ...
    }
```

In the `_retrieve()` function, the target where the call will be forwarded to is the `_target` parameter, which is a ProxyTarget contract. Concretely, the delegatecall function that will be triggered in the ProxyTarget will be `withdrawMax()`:

```
// https://github.com/sablier-labs/v2-periphery/blob/ba3926d2c3e059a230211077087 ⌋
 ↪  b73afe46acf64/src/abstracts/SablierV2ProxyTarget.sol#L141C5-L143C6

function withdrawMax(ISablierV2Lockup lockup, uint256 streamId, address to)
 ↪  external onlyDelegateCall {
    lockup.withdrawMax(streamId, to);
}
```

As we can see, the `withdrawMax()` function has as parameters the `lockup` stream contract to withdraw from, the `streamId` and the address `to` which will receive the available funds from the stream. The vulnerability lies in the parameters passed when calling the `withdrawMax()` function in `_retrieve()`. As we can see, the first encoded parameter in the `encodeWithSelector()` call after the selector is the `_target`:

```
// CouncilMember.sol

function _retrieve() internal {
        ...
        // Execute the withdrawal from the _target, which might be a Sablier
 ↪  stream or another protocol
```

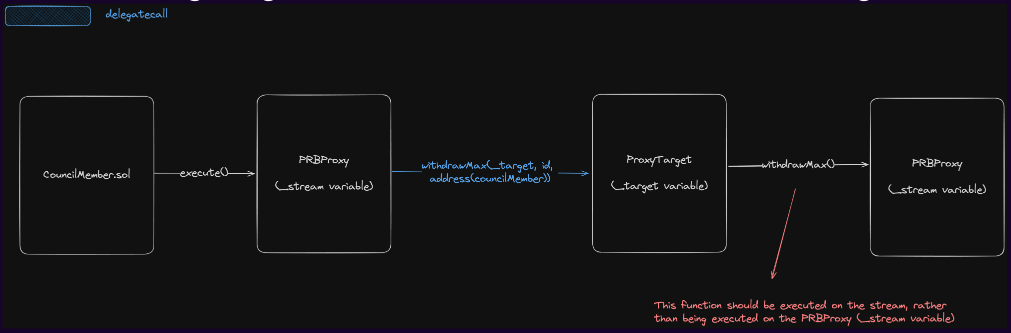
```
        _stream.execute(
            _target,
            abi.encodeWithSelector(
                ISablierV2ProxyTarget.withdrawMax.selector,
                _target,    // <------- This is incorrect
                _id,
                address(this)
            )
        );

        ...
    }
```

This means that the proxy target's `withdrawMax()` function will be triggered with the `_target` contract as the `lockup` parameter, which is incorrect. This will make all calls eventually execute `withdrawMax()` on the PRBProxy contract, always reverting.

The parameter needed to perform the `withdrawMax()` call correctly is the actual Sablier lockup contract, which is currently not stored in the `CouncilMember` contract.

The following diagram also summarizes the current wrong interactions for clarity:


delegatecall
CouncilMember.sol
execute()
PRBProxy
(_stream variable)
withdrawMax(_target, id, address(councilMember))
ProxyTarget
(_target variable)
withdrawMax()
PRBProxy
(_stream variable)
This function should be executed on the stream, rather than being executed on the PRBProxy (_stream variable)

Impact

High. ALL withdrawals from the Sablier stream will revert, effectively causing a DoS in the _retrieve() function. Because the _retrieve() function is called in all the main protocol functions, this vulnerability essentially prevents the protocol from ever functioning correctly.

Proof of Concept

Because the current Telcoin repo does not include actual tests with the real Sablier contracts (instead, a `TestStream` contract is used, which has led to not unveiling this vulnerability), [I've created a repository](https://github.com/0xadrii/telcoin-proof-of-concept) where the poc can be executed (the repository will be public after the audit finishes (on 15 jan. 2024 at 16:00 CET)). The `testPoc()` function shows how any interaction (in this case, a call to the `mint()` function) will fail because the proper Sablier contracts are used (PRBProxy and proxy target):

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console2} from "forge-std/Test.sol";
import {SablierV2Comptroller} from
↪  "@sablier/v2-core/src/SablierV2Comptroller.sol";
import {SablierV2NFTDescriptor} from
↪  "@sablier/v2-core/src/SablierV2NFTDescriptor.sol";
import {SablierV2LockupLinear} from
↪  "@sablier/v2-core/src/SablierV2LockupLinear.sol";
import {ISablierV2Comptroller} from
↪  "@sablier/v2-core/src/interfaces/ISablierV2Comptroller.sol";
import {ISablierV2NFTDescriptor} from
↪  "@sablier/v2-core/src/interfaces/ISablierV2NFTDescriptor.sol";
import {ISablierV2LockupLinear} from
↪  "@sablier/v2-core/src/interfaces/ISablierV2LockupLinear.sol";

import {CouncilMember, IPRBProxy} from "../src/core/CouncilMember.sol";
import {TestTelcoin} from "./mock/TestTelcoin.sol";
import {MockProxyTarget} from "./mock/MockProxyTarget.sol";
import {PRBProxy} from "./mock/MockPRBProxy.sol";
import {PRBProxyRegistry} from "./mock/MockPRBProxyRegistry.sol";

import {UD60x18} from "@prb/math/src/UD60x18.sol";
import {LockupLinear, Broker, IERC20} from
↪  "@sablier/v2-core/src/types/DataTypes.sol";
import {IERC20 as IERC20OZ} from
↪  "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract PocTest is Test {

    ////////////////////////////////////////////////////////////////
    //                         CONSTANTS                          //
    ////////////////////////////////////////////////////////////////

   bytes32 public constant GOVERNANCE_COUNCIL_ROLE =
        keccak256("GOVERNANCE_COUNCIL_ROLE");
    bytes32 public constant SUPPORT_ROLE = keccak256("SUPPORT_ROLE");

    ////////////////////////////////////////////////////////////////
    //                          STORAGE                           //
    ////////////////////////////////////////////////////////////////

    /// @notice Poc Users
    address public sablierAdmin;
    address public user;

    /// @notice Sablier contracts
    SablierV2Comptroller public comptroller;
    SablierV2NFTDescriptor public nftDescriptor;
    SablierV2LockupLinear public lockupLinear;

    /// @notice Telcoin contracts
    PRBProxyRegistry public proxyRegistry;
    PRBProxy public stream;
    MockProxyTarget public target;
    CouncilMember public councilMember;
    TestTelcoin public telcoin;

    function setUp() public {
        // Setup users
        _setupUsers();

        // Deploy token
        telcoin = new TestTelcoin(address(this));

        // Deploy Sablier
        _deploySablier();

        // Deploy council member
        councilMember = new CouncilMember();

        // Setup stream
        _setupStream();

        // Setup the council member
        _setupCouncilMember();
    }

    function testPoc() public {
      // Step 1: Mint council NFT to user
      councilMember.mint(user);
      assertEq(councilMember.balanceOf(user), 1);

      // Step 2: Forward time 1 days
      vm.warp(block.timestamp + 1 days);

      // Step 3: All functions calling _retrieve() (mint(), burn(),
↪   removeFromOffice()) will fail
      vm.expectRevert(abi.encodeWithSignature("PRBProxy_ExecutionReverted()"));
      councilMember.mint(user);
```

```
}

function _setupUsers() internal {
    sablierAdmin = makeAddr("sablierAdmin");
    user = makeAddr("user");
}

function _deploySablier() internal {
    // Deploy protocol
    comptroller = new SablierV2Comptroller(sablierAdmin);
    nftDescriptor = new SablierV2NFTDescriptor();
    lockupLinear = new SablierV2LockupLinear(
        sablierAdmin,
        ISablierV2Comptroller(address(comptroller)),
        ISablierV2NFTDescriptor(address(nftDescriptor))
    );
}

function _setupStream() internal {

    // Deploy proxies
    proxyRegistry = new PRBProxyRegistry();
    stream = PRBProxy(payable(address(proxyRegistry.deploy())));
    target = new MockProxyTarget();

    // Setup stream
    LockupLinear.Durations memory durations = LockupLinear.Durations({
        cliff: 0,
        total: 1 weeks
    });

    UD60x18 fee = UD60x18.wrap(0);

    Broker memory broker = Broker({account: address(0), fee: fee});
    LockupLinear.CreateWithDurations memory params = LockupLinear
        .CreateWithDurations({
            sender: address(this),
            recipient: address(stream),
            totalAmount: 100e18,
            asset: IERC20(address(telcoin)),
            cancelable: false,
            transferable: false,
            durations: durations,
            broker: broker
        });
        bytes memory data =
↪   abi.encodeWithSelector(target.createWithDurations.selector,
↪   address(lockupLinear), params, "");

        // Create the stream through the PRBProxy
        telcoin.approve(address(stream), type(uint256).max);
        bytes memory response = stream.execute(address(target), data);
        assertEq(lockupLinear.ownerOf(1), address(stream));
    }

    function _setupCouncilMember() internal {
      // Initialize
      councilMember.initialize(
            IERC20OZ(address(telcoin)),
            "Test Council",
            "TC",
            IPRBProxy(address(stream)), // stream_
            address(target), // target_
            1, // id_
            address(lockupLinear)
        );

        // Grant roles
        councilMember.grantRole(GOVERNANCE_COUNCIL_ROLE, address(this));
        councilMember.grantRole(SUPPORT_ROLE, address(this));
    }

}
```

Code Snippet

https://github.com/sherlock-audit/2024-01-telcoin/blob/main/telcoin-audit/contracts/sablier/core/CouncilMember.sol#L275

Tool used

Manual Review, foundry

Recommendation

In order to fix the vulnerability, the proper address needs to be passed when calling `withdrawMax()`.

> Note that the actual stream address is currently NOT stored in `CouncilMember.sol`, so it will need to be stored (my example shows a new

`actualStream` variable)

```
function _retrieve() internal {
        ...
        // Execute the withdrawal from the _target, which might be a Sablier
↪   stream or another protocol
        _stream.execute(
            _target,
            abi.encodeWithSelector(
                ISablierV2ProxyTarget.withdrawMax.selector,
-                _target,
+       actualStream
                _id,
                address(this)
            )
        );

        ...
    }
```

Discussion

amshirif

https://github.com/telcoin/telcoin-audit/pull/43

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

takarez commented:

> valid because { This is valid a a dupp of 086; the watson claims its hight but will still make it meduim due to the impact mentioned in issue 086; but making it the best report as the POC is well written and implemented}

nevillehuang

@amshirif Is there anyway the admin can unblock DoS in withdrawals?

amshirif

@nevillehuang No, a new contract with these fixes would need to be deployed to prevent DoS because those two values had to be the same prior to the fix.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/telcoin/telcoin-audit/pull/43.

sherlock-admin

The Lead Senior Watson signed-off on the fix.

Issue H-3: CouncilMember:burn renders the contract inoperable after the first execution

Source: https://github.com/sherlock-audit/2024-01-telcoin-judging/issues/199

Found by

0xAsen, 0xLogos, 0xadrii, 0xlamide, 0xmystery, 0xpep7, Aamirusmani1552, Arz, BAICE, Bauer, DenTonylifer, HonorLt, Ignite, IvanFitro, Jaraxxus, Kow, Krace, VAD37, alexbabits, almurhasan, araj, bitsurfer, dipp, fibonacci, ggg_ttt_hhh, gqrp, grearlake, jah, m4ttm, mstpr-brainbot, popeye, psb01, r0ck3tz, ravikiran.web3, sakshamguruji, sobieski, sonny2k, tives, ubl4nk, vvv, ydlee, zhuying, zzykxx

Summary

The CouncilMember contract suffers from a critical vulnerability that misaligns the balances array after a successful burn, rendering the contract inoperable.

Vulnerability Detail

The root cause of the vulnerability is that the `burn` function incorrectly manages the `balances` array, shortening it by one each time an ERC721 token is burned while the latest minted NFT still withholds its unique `tokenId` which maps to the previous value of `balances.length`.

```
// File: telcoin-audit/contracts/sablier/core/CouncilMember.sol
210:    function burn(
        ...
220:        balances.pop(); // <= FOUND: balances.length decreases, while latest
 ↪  minted nft withold its unique tokenId
221:        _burn(tokenId);
222:    }
```

This misalignment between existing `tokenIds` and the `balances` array results in several critical impacts:

1. Holders with tokenId greater than the length of balances cannot claim.
2. Subsequent burns of tokenId greater than balances length will revert.
3. Subsequent mint operations will revert due to tokenId collision. As `totalSupply` now collides with the existing `tokenId`.

```
// File: telcoin-audit/contracts/sablier/core/CouncilMember.sol
173:    function mint(
        ...

            balances.push(0);
            _mint(newMember, totalSupply());// <= FOUND
        }
```

This mismanagement creates a cascading effect, collectively rendering the contract inoperable. Following POC will demonstrate the issue more clearly in codes.

POC

Run `git apply` on the following patch then run `npx hardhat test` to run the POC.

Result

> CouncilMember mutative burn Success inoperable contract after burn (90ms) 1 passing (888ms)

The Passing execution of the POC confirmed that operations such as `claim`, `burn` & `mint` were all reverted which make the contract inoperable.

Impact

The severity of the vulnerability is high due to the high likelihood of occurence and the critical impacts on the contract's operability and token holders' ability to interact with their assets.

Code Snippet

https://github.com/sherlock-audit/2024-01-telcoin/blob/main/telcoin-audit/contracts/sablier/core/CouncilMember.sol#L220

Tool used

VsCode

Recommendation

It is recommended to avoid popping out balances to keep alignment with uniquely minted tokenId. Alternatively, consider migrating to ERC1155, which inherently manages a built-in balance for each NFT.

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

takarez commented:

> valid because { this is a valid findings because the watson explain how again the burn function will break a functionality just like the previous issue thus making it a dupp of 109}

nevillehuang

See comments here for duplication reasons.

amshirif

https://github.com/telcoin/telcoin-audit/pull/31

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/telcoin/telcoin-audit/pull/31.

sherlock-admin

The Lead Senior Watson signed off on the fix.

Issue M-1: The `CouncilMember` contract DoS due to the `_retrieve` function revert

Source: https://github.com/sherlock-audit/2024-01-telcoin-judging/issues/47

Found by

0xadrii, Arz, Ignite, Tricko, fibonacci

Summary

The `_retrieve`function is called before any significant state changes. This function executes the withdrawal from the `_target`, which might be a Sablier stream or another protocol. `SablierV2Lockup` reverts if withdrawable amount equals to `0`.

https://github.com/sablier-labs/v2-core/blob/b0016437ef3cc8606e1100965dd911d7e658b40b/src/abstracts/SablierV2Lockup.sol#L297-L299
https://github.com/sablier-labs/v2-core/blob/b0016437ef3cc8606e1100965dd911d7e658b40b/src/abstracts/SablierV2Lockup.sol#L270-L272

Funds are distributed over time. And even if there are always funds in the protocol for distribution, after calling the `_retrieve` function, a new distribution will not be available until another period of time has passed.

This means that any interaction with the `CouncilMember` contract will be unavailable during this time.

Vulnerability Detail

1. If the protocol for distributing funds employs a strategy that permits funds to be released once within a specific timeframe (for instance, 1 day, 1 week, or 1 month), this implies that the `CouncilMember` contract will execute its tasks error-free only once during this period.

2. The `removeFromOffice` function calls the `_retrieve`function at the beginning to retrieve and distribute any pending TELCOIN for all council members, and transfer token ownership at the end.

https://github.com/sherlock-audit/2024-01-telcoin/blob/main/telcoin-audit/contracts/sablier/core/CouncilMember.sol#L267-L295

The `_update` function, which is called before each transfer, is overridden and also calls the `_retrieve` function.

https://github.com/sherlock-audit/2024-01-telcoin/blob/main/telcoin-audit/contracts/sablier/core/CouncilMember.sol#L321-L331

Thus, during the `removeFromOffice` function, the `_retrieve` function will be called twice, which will always result in revert, since after the first distribution of funds, when called again, the withdrawable amount will be `0`.

3. Also, according to the sponsor's comment, the council members are semi-trusted. A malicious member can prevent others from interacting with the contract. For example:

- Member A wants to claim their allocated amounts of TELCOIN
- Member B fron-runs member's A transaction and call the `retrieve` function
- Member's A transaction reverts because the `_retrieve` function is called again but there are no more withdrawable amount.

Impact

Denial of Service of the `CouncilMember` contract over a period of time, depending on the fund distribution strategy. The `removeFromOffice` function always fails, leading to the necessity to use the `transferFrom` function, which does not call `_withdrawAll`, potentially breaking the state of the contract

Code Snippet

https://github.com/sherlock-audit/2024-01-telcoin/blob/main/telcoin-audit/contracts/sablier/core/CouncilMember.sol#L267-L295

Tool used

Manual Review

Recommendation

Check amount before executing withdrawal or wrap call in a `try/catch` block. Also consider abandoning the `removeFromOffice` function, use `transferFrom` instead and move `_withdrawAll` call to `_update` function.

Discussion

amshirif

https://github.com/telcoin/telcoin-audit/pull/37

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

takarez commented:

> valid because { valid and a dupp of 051 with a non standard recommendation than 051}

nevillehuang

@amshirif I think this could possibly be medium severity, given there is no definite loss of funds other than when a malicious council member can be prevented from being removed. The difference between this and #139 is it doesn't affect withdrawals of council members. Also I think #141 and #98 are the most comprehensive report, with #118 highlighting a front-running issue. (but sherlock automated tool selected this)

amshirif

@nevillehuang Yes I agree

0xf1b0

Escalate

I disagree with the severity. It shares the same impact as #139, as both are results of the `_retrieve` function reverting. However, the root cause of the revert is different.

This issue also affects the withdrawal, as the withdrawal process itself includes the `_retrieve` function call. The Vulnerability Detail section provides scenarios 1 and 3, which illustrate how withdrawals can potentially be halted.

sherlock-admin

> Escalate
>
> I disagree with the severity. It shares the same impact as #139, as both are results of the `_retrieve` function reverting. However, the root cause of the revert is different.
>
> This issue also affects the withdrawal, as the withdrawal process itself includes the `_retrieve` function call. The Vulnerability Detail section provides scenarios 1 and 3, which illustrate how withdrawals can potentially be halted.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

@0xf1b0 Can you provide a coded PoC so that I can analyze the differences in root cause? I think this might be a duplicate of #139

0xArz

@nevillehuang The root cause in https://github.com/sherlock-audit/2024-01-telcoin-judging/issues/139 is that when calling `_retrieve()` it will always revert because we are calling a wrong address.

The root cause here is that `_retrieve()` reverts when withdrawing 0 amounts, in some functions like mint() it is called 2 times - first called in the function and then its called the second time in ERC721.update() which will fail the second time because we already withdrew the max. Or for example a stream is used where the withdrawable amount is 0 for some time - unlocking in steps etc.

The impact of this issue is that we call only mint 1 CouncilMember nft because the first time mint() is called, `_retrieve()` is called only once, after that all calls to mint(),burn() and removeFromOffice() will revert because `_retrieve()` is called 2 times.

The 1 council member can still claim the rewards but if a dynamic stream is used and the council member calls the public `retrieve()` he can then fail to claim his rewards for some time until more rewards are unlocked. Although because we will only have 1 council member this will lead to unfair distribution of the rewards

amshirif

This is not a duplicate of #139, and it does not share the same impact. #139 is more serious as it essentially prevents the withdrawal ability from ever working.

nevillehuang

Agree with sponsor @amshirif, unless @0xArz @0xf1b0 can show a PoC of the issue showing an impact that prevents withdrawals/affects rewards claiming.

0xArz

> Agree with sponsor @amshirif, unless @0xArz @0xf1b0 can show a PoC of the issue showing an impact that prevents withdrawals/affects rewards claiming.

I agree, funds can be stuck but the DoS is only temporarily. However if we only have 1 council member then 100% of the funds are distributed to him which imo is quite a big problem as council members are semitrusted and other members that were supposed to receive funds will not receive anything but its up to you to decide whether this defines high severity or no.

nevillehuang

@0xArz I am abit confused by your statement. How can there be other council members that were supposed to receive funds when there is only 1 council member decided by the governance?

0xArz

@nevillehuang Yeah there will only be 1 council member but for example lets say the governance wanted to have 3 council members, they will fail to set the other members after the first one because the retrieve reverts. So instead of having 3 council members there will only be 1 and he will receive 100% of the funds while the other members that were supposed to be set will not receive anything because they were not set

0xf1b0

> Agree with sponsor @amshirif, unless @0xArz @0xf1b0 can show a PoC of the issue showing an impact that prevents withdrawals/affects rewards claiming.

Doesn't case 3 from the Vulnerability Detail, where malicious actor can front-run every transaction with `retrieve` call, show this impact? No one will be able to withdraw funds.

nevillehuang

@0xArz Acknowledge this possibility given `mint()` and `burn()` can possibly be bricked too. However, since the first council member still get their intended rewards, admins can then choose to not topup rewards thereafter. So I believe this is just a DoS scenario.

Evert0x

Planning to reject escalation and keep issue state as is.

The provided context and discussion fail to make the case for high severity as the impact is limited to specific actors and scenarios.

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xf1b0: rejected

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/telcoin/telcoin-audit/pull/37.

sherlock-admin

The Lead Senior Watson signed off on the fix.

Issue M-2: Sablier stream update in `CouncilMember.sol` can cause loss of funds if the streamed balance is not withdrawn.

Source: https://github.com/sherlock-audit/2024-01-telcoin-judging/issues/99

Found by

Aamirusmani1552, Tricko

Summary

The vulnerability in the `CouncilMember` contract pertains to the failure to withdraw streamed tokens during a contract stream update, potentially resulting in fund loss for both the contract and the entire council.

Vulnerability Detail

Sablier Streams facilitate token streaming on a per-second basis, involving a sender who initiates the stream and a receiver who receives the streamed tokens. The receiver can withdraw tokens up to the elapsed seconds from the stream's start. The responsibility to claim streamed tokens lies with the receiver, as stated in the documentation and Sablier stream contracts (read the cancel stream docs here) . Once tokens are streamed, the sender cannot withdraw them.

Also sender has the authority to cancel the the stream and claim back the un-streamed amount. But streamed Balance upto the elapsed time can still be claimed by the receiver or person who is approved by the receiver only.

Check the `SablierV2Lockup::cancel()` here : https://github.com/sablier-labs/v2-core/blob/b0016437ef3cc8606e1100965dd911d7e658b40b/src/abstracts/SablierV2Lockup.sol#L153-L168

Docs for the same could be find here : https://docs.sablier.com/contracts/v2/guides/stream-management/cancel

As we check from the resources given above, if a stream is canceled only the un-streamed balance will be available for the sender to withdraw. Rest if for the receiver.

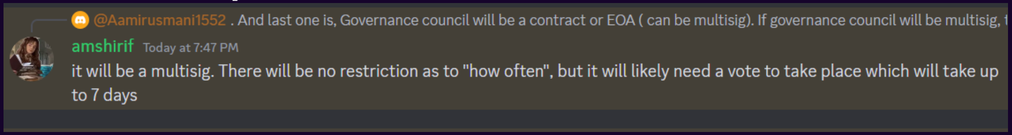
The issue arises in the `CouncilMember` contract's functions (`CouncilMember::updateStream(...)`, `CouncilMember::updateID(...)`, and `CouncilMember::updateTarget(...)`) as they do not check whether the entire streamed amount has been withdrawn from the Sablier stream before updating the stream states in the contract. Consequently, if there is an active streamed balance in the Sablier stream, the `CouncilMember` contract will not be able to withdraw it. And now the balance is lying idle in the Sablier stream contract.

However, the previously streamed balance can be reclaimed by adding the old stream back to the `CouncilMember` contract, provided the sender is aware that the streamed balance has not been withdrawn. Nonetheless, complications may arise if modifications are made to the `CouncilMember` contract following the stream update. For instance, the removal of a Council Member could lead to the omitted member not receiving their balance, while the addition of a new member may result in every old member receiving fewer tokens and new members gaining tokens share. This can happen because all update stream functions are handled by the role `GOVERNANCE_COUNCIL_ROLE` in the `CouncilMember` contract. And if it is a multi-sig or governance then it would required a vote to happend in order to perform the new updated. And sponsor confirmed that the multi-sig can be added for this role. Here is the conversation:

Question Asked by me:

> And last one is, Governance council will be a contract or EOA (can be multisig). If governance council will be multisig, then how often can it make updates to the contracts?

Answer from Sponsor:



@Aamirusmani1552 . And last one is, Governance council will be a contract or EOA (can be multisig). If governance council will be multisig,

amshirif Today at 7:47 PM

it will be a multisig. There will be no restriction as to "how often", but it will likely need a vote to take place which will take up to 7 days

So if this is the case then new update will be done after some time and a lot of things might happen in that time.

Also the sender's awareness play important role in this. Two scenarios may unfold because of this:

1. The stream has fully distributed its balance, and the sender assumes that the funds have been appropriately allocated to the `CouncilMember` contract.
2. The stream needs to be prematurely canceled for specific reasons, requiring the addition of a new stream.

In both of the scenarios if the sender is unaware then it will be complete loss of tokens.

Impact

Council members face potential token losses due to the inability to withdraw streamed balances.

Code Snippet

https://github.com/sherlock-audit/2024-01-telcoin/blob/main/telcoin-audit/contracts/sablier/core/CouncilMember.sol#L229C3-L257C1

Tool used

- Manual Review

Recommendation

To mitigate this potential issue, the following actions are advised:

1. Implement Sablier Hooks:

Sablier offers essential hooks to address scenarios where the receiver is a contract. These hooks enable the receiver contract to update its state accurately. While these hooks are optional, Sablier strongly recommends their implementation. Of particular relevance in this context is the onStreamCanceled hook, triggered by the Sablier stream contract when the sender cancels the stream. By incorporating this hook in the CouncilMember contract, the receiver can invoke the _retrieve() function upon stream cancellation, ensuring the withdrawal of the entire streamed balance.

File: CouncilMember.sol

```
+ import { ISablierV2LockupRecipient } from
↪   "@sablier/v2-core/src/interfaces/hooks/ISablierV2LockupRecipient.sol";

    contract CouncilMember is
        ERC721EnumerableUpgradeable,
        AccessControlEnumerableUpgradeable
+    ISablierV2LockupRecipient
    {

+    function onStreamCanceled(
+        uint256 streamId,
+        uint128, /* senderAmount */
+        uint128 /* recipientAmount */
+    )
+        external
+        pure
+    {
+        _retrieve();
+    }
```

```
    }
```

2. Check Stream Depletion in Update Function:

In the stream update function, verify whether the stream is depleted or not. If not, withdraw the streamed tokens before updating the balances. It is crucial to check if the stream is depleted because if the `_retrieve()` function is directly called and the stream has been depleted (all tokens withdrawn by the receiver), invoking `stream.withdrawMax()` will revert. This could lead to a revert in the `_retrieve()` function and potentially cause a denial-of-service (DoS) situation in the stream update function.

File: CouncilMember.sol

```
+    // Syncronize the update process
+    function updateStreamData( IPRBProxy stream_,  address target_, uint256
↪   updateID ) external onlyRole(GOVERNANCE_COUNCIL_ROLE){
+      _checkIfDepleted();
+      _updateStream(stream_);
+      _updateTarget(target_);
+      _updateID(updateID);
+    }

-    function updateStream(
+    function _updateStream(
         IPRBProxy stream_
-    ) external onlyRole(GOVERNANCE_COUNCIL_ROLE) {
+   ) internal {
         _stream = stream_;
         emit StreamUpdated(_stream);
    }


    /**
     * @notice Update the target address
     * @dev Restricted to the GOVERNANCE_COUNCIL_ROLE.
     * @param target_ New target address.
     */
-    function updateTarget(
+    function _updateTarget(
         address target_
-    ) external onlyRole(GOVERNANCE_COUNCIL_ROLE) {
+   ) internal {
         _target = target_;
         emit TargetUpdated(_target);
    }


    /**
     * @notice Update the ID for a council member
     * @dev Restricted to the GOVERNANCE_COUNCIL_ROLE.
     * @param id_ New ID for the council member.
     */
-       function updateID(uint256 id_) external
↪   onlyRole(GOVERNANCE_COUNCIL_ROLE) {
+      function _updateID(uint256 id_) internal {
        _id = id_;
        emit IDUpdated(_id);
      }

+   // assuming IPRBProxy will return results like given below since we have not
↪   been provided with PRBProxy in the codebase.
+  // make changes according to the interface to below given function.
+   function _checkIfDepleted() _internal view {
+        (bool success, bytes memory data) = _stream.execute(
+            _target,
+            abi.encodeWithSelector(
+                ISablierV2ProxyTarget.isDepleted.selector,
+                _id
+            )
+        );

+      require(success, "Call failed");
+      require(abi.decode(data, (bool)), "Stream is not depleted yet.");
+   }
```

Note: Make necessary adjustments in the interfaces used.

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

takarez commented:

> invalid because { This is invalid because the funcions in question (updateStream and updateId) have a governance modifier which requires the governance to execute this action; according to sherlock its invalid}

amshirif

Duplicate issue

https://github.com/sherlock-audit/2024-01-telcoin-judging/issues/112

amshirif

https://github.com/telcoin/telcoin-audit/pull/49

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/telcoin/telcoin-audit/pull/49.

sherlock-admin

The Lead Senior Watson signed-off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.