PossumLabs CyberCash Audit Report

Version 1.0

Mahdi Rostami

February 1, 2025

PossumLabs CyberCash Audit Report

Mahdi Rostami

January 31, 2025

Prepared by: Mahdi Rostami [https://x.com/0xmahdirostami]

Table of Contents

CyberCash Report Security Review

A security review of the [CyberCash], [GitHub] smart contracts was done by [mahdi rostami] .
This audit report includes all the vulnerabilities, issues and code improvements found during the security review.

Disclaimer

“Audits are a time, resource and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.”

- Secureum

Impact

- **High** Issues that lead to the loss of user funds. Such issues include:
  - Direct theft of any user funds, whether at rest or in motion.
  - Long-term freezing of user funds. Theft or long term freezing of unclaimed yield or other assets.
  - Protocol insolvency
- **Medium** Issues that lead to an economic loss but do not lead to direct loss of on-chain assets. Examples are:
  - Gas griefing attacks (make users overpay for gas)
  - Attacks that make essential functionality of the contracts temporarily unusable or inaccessible.
  - Short-term freezing of user funds.
- **Low** Issues where the behavior of the contracts differs from the intended behavior (as described in the docs and by common sense), but no funds are at risk.

Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

Executive summary

Overview

Project Name	CyberCash
Repository	[Link]
Commit Hash	[d766a22]
Docs	No Docs
Methods	Manual Review

Scope

File
All files in repo

Compatibilities

- Solc Version: =0.8.19

Known Issues

None

Issues found

Severity	Count
High	0
Medium	1
Low	3

Severity	Count
Info/Gas	3

Findings

Medium Severity

[M-1] Contract doesn't prevent burn score transfer to migrator and liquidity pool

Description

The contract does not burn or reward for the migrator and liquidity provider (LP). An attacker could transfer the burn score to these addresses, breaking some functions.

Impact

the contract will show a higher balance than the actual balance for those addresses, and if attackers repeat it they gain some tokens which is not accepted.

Proof of Concept (PoC)

```
// Transfer the burnScore from one user to another & verify new reward accrual
function test_TransferBurnScoreToMigrator() public {
    helper_Migrator_setCashAddress();
    helper_Migrator_addTokenMigration();

    uint256 amountSend = 1e21;

    vm.startPrank(treasury);
    // Step 1: Send tokens to Alice, treasury accrues burnScore
    cyberCash.transfer(Alice, amountSend);

    // Step 2: Let rewards accrue
    vm.warp(block.timestamp + oneYear);

    // Step 3: Transfer burnScore to migrator
    cyberCash.transferBurnScore(address(migrator), cyberCash.burnScore(treasury));

    // Step 4: Let rewards accrue & check
    vm.warp(block.timestamp + oneYear);

    uint256 mint = MINT_PER_SECOND * oneYear;

```

```
23      assertEq(cyberCash.balanceOf(address(migrator)), mint); // migrator
            earns ~1Bn
24      vm.stopPrank();
25
26      uint256 psmAmount = 1e6;
27      uint256 migrationOne = 1e6;
28
29      // Treasury sends psm to Alice
30      vm.startPrank(treasury);
31      IERC20(psm).transfer(Alice, psmAmount);
32
33      // Alice sets approval
34      vm.startPrank(Alice);
35      IERC20(psm).approve(address(migrator), 1e6);
36
37      // Scenario 1: Alice migrates
38      vm.expectRevert("ERC20: transfer amount exceeds balance"); //@audit
39      migrator.migrate(psm, migrationOne);
40      vm.stopPrank();
41
42      vm.startPrank(treasury);
43      // Step 1: Send tokens to Alice, treasury accrues burnScore
44      cyberCash.transfer(Alice, amountSend);
45
46      // Step 2: Let rewards accrue
47      vm.warp(block.timestamp + oneYear);
48
49      // Step 3: Transfer burnScore to migrator
50      cyberCash.transferBurnScore(address(migrator), cyberCash.burnScore(
            treasury));
51
52      // Step 4: Let rewards accrue & check
53      vm.warp(block.timestamp + oneYear);
54
55      mint = MINT_PER_SECOND * oneYear;
56
57      assert(cyberCash.balanceOf(address(migrator)) > 2 * mint); //
            migrator gains some real balance  //@audit
58      vm.stopPrank();
59
60      // Alice sets approval
61      vm.startPrank(Alice);
62      IERC20(psm).approve(address(migrator), 1e6);
63
64      // Scenario 1: Alice migrates
65      migrator.migrate(psm, migrationOne);  //@audit
66      vm.stopPrank();
67  }
```

As shown, the test reverts because the migrator assumes its balance is more than the real balance. If an attacker transfers the burn score again, the migrator will gain some real balance.

Mitigation

Restrict those addresses in `transferBurnScore`.

Remark

Fixed. [commit]

Low Severity

[L-1] Contract is not compatible with more than 18 decimal tokens

Description

Currently, our contract assumes that all approved tokens will have 18 decimals. However, this assumption may be false in the future, potentially leading to issues if tokens with more than 18 decimals are used.

For tokens with fewer than 18 decimals, owners could set a higher ratio to compensate. For example:

- For a token with 17 decimals, a 1:1 migration could use a ratio of `RATIO_PRECISION * 10`.

However, for tokens with more than 18 decimals, owners have limited flexibility. With RATIO_PRECISION set to 1000, the ratio could range from 1 to 1000. This creates a limitation for owners who want to provide more tokens.

For instance, if a token has 20 decimals and an owner wants to offer a 1:100 ratio, they would need to set the ratio to 0.1, which is not allowed given the current constraints.

Impact

Currently, there is little real-world impact, as owners can create separate contracts for unusual cases. However, this approach is suboptimal and may lead to unnecessary complexity.

Mitigation

To address this issue, we propose implementing a decimal normalization mechanism. This would allow the contract to handle tokens with varying decimal precisions more effectively, providing greater flexibility and reducing potential conflicts.

Remark

Fixed. [commit]

[L-2] Unnecessary increase in total supply of tokens that were never minted

Description

Currently, the rewards earned between the deployment time and the first transfer are added to the pending mint. However, these rewards will never be minted, leading to an incorrect increase in the total supply.

Impact

The total supply increases despite the fact that these tokens will not be minted.

Proof of Concept (PoC)

```
function test_pendingMint() public {
    vm.warp(block.timestamp + oneYear);

    // Transfer tokens
    vm.prank(treasury);
    cyberCash.transfer(Alice, 13e17);
    console.log(cyberCash.pendingMints());
}
```

Logs Before Mitigation:

```
[PASS] test_pendingMint() (gas: 170495)
Logs:
  999999999999999999990144000
```

Logs After Mitigation:

```
[PASS] test_pendingMint() (gas: 175560)
Logs:
  0
```

Mitigation

To address this issue, check for the first call to `burnsAndRewards` and avoid updating the pending mint on the first call:

```
if (first) {
    first = 0;
} else {
    pendingMints = (pendingMints + addedRewards) - rewards;
}
```

Remark

Fixed. [commit]

[L-3] Non minted tokens are added to pending mint due to total burn start point

Description

Currently, due to an initial total burned value of 1, over time, some rewards are not being minted. Instead, these rewards are added to pending mint and incorrectly increase the total supply.

Impact

A wrong increase in total supply occurs as these tokens will not minted as intended.

Proof of Concept (PoC)

```
function test_pendingMint() public {
    vm.warp(block.timestamp + oneYear);

    // Transfer tokens
    vm.prank(treasury);
    cyberCash.transfer(Alice, 13e17);
    console.log(cyberCash.pendingMints());
}
```

Logs Before Mitigation:

```
[PASS] test_pendingMint() (gas: 194180)
Logs:
  153846153847
```

Logs After Mitigation:

```
[PASS] test_pendingMint() (gas: 215290)
Logs:
  1
```

Mitigation

To address this issue, check for the first call to `burnsAndRewards` and decrease the total burned value by 1 for that first call:

```
if (first) {
    if (burnedFromTx > 0) {
         totalBurned += burnedFromTx - 1;
         first = 0;
   }
} else {
    totalBurned += burnedFromTx;
}
```

Remark

Fixed. [commit]

Info Gas

- Boolean equality is not required

If x is a boolean, there is no need to do if(x == true) or if(x == false). Just use if(x) and if(!x) respectively.

Instances: s

```
1             if (allowedTokens[_token] == true) revert IsAllowed();
```

Fixed. [commit]

- No mechanism to withdraw allowed tokens from Migrator.sol

All allowed tokens will be locked in the migration contract, as there is currently no function to withdraw them. This poses an issue since the owner may want to either withdraw or burn these tokens.

Acknowledged.

- Limitation on adding additional migrators and special addresses

The contract owner currently lacks the ability to add new migrators for unconventional tokens in the future. Additionally, there is no option to include special addresses, such as farm contracts, that do not require a burn score.

Acknowledged.