Security Assessment for Yamato Protocol

Report by Ulam Labs

Over-collateralized stablecoin

Findings and Recommendations Report Presented to:

Yamato Team

September 4, 2023 Version: 1.0

Presented by:

Ulam Labs

Grabiszynska 163/502,

53-332 Wroclaw, POLAND

Executive Summary

Overview

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Ulam Labs Security Teams took to identify and validate each issue, and any applicable recommendations for remediation.

Scope

The audit has been conducted on the commit **d0c954e2dc0eceee257b0927ef45815d6dfec23a** of Yamato public GitHub Repository.

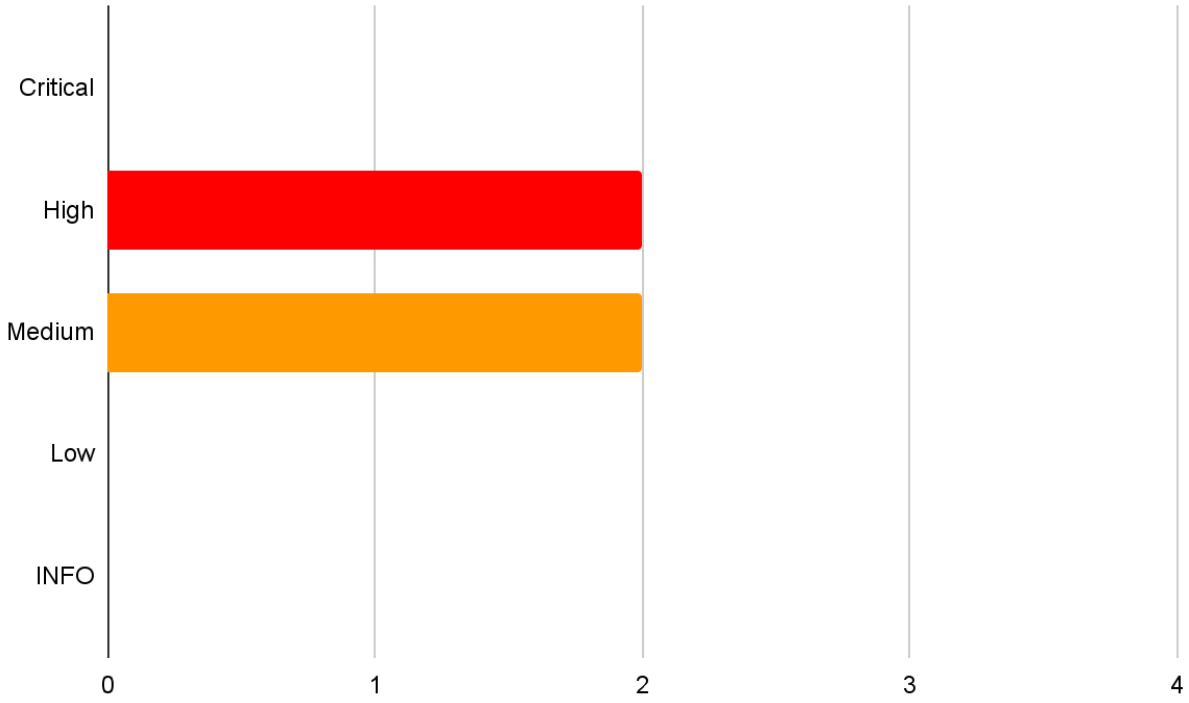

Chart 1: Findings by severity.

Key findings

During the Security Assessment, following findings have been discovered:

- 0 findings with a CRITICAL severity rating,
- 2 findings with a HIGH severity rating,
- 2 findings with a MEDIUM severity rating,
- 0 findings with a LOW severity rating.
- 0 findings with an INFO severity rating.

Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of the contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Technical analysis & findings

Some pledges could be impossible to redeem

Finding ID: **YP-1**
Contract: YamatoRedeemerV4.sol
Severity: High
Status: Open

Description

At the beginning of while loop in **runRedeem** method pledge is removed from the ranked queue.

```
while (true) {

   address _pledgeAddr = _prv6.rankedQueuePop(vars._nextICR);
```

It is crucial to put such pledge back on queue, just like it is done here:

```
if (

   vars._toBeRedeemedFragment == 0 &&

   _ICRpertenk >= vars._mcrPertenk

) {

   vars._skippedPledges[vars._skipCount] = _pledge;

   vars._skipCount++;

   continue; /* To skip until next poppables. This must be upserted below
to refresh obsoleted priority. */

}
```

Skipped pledges are then appended to bulked pledges:

```
for (uint256 i; i < vars._maxCount; ) {

   vars._bulkedPledges[vars._maxCount + i] = vars._skippedPledges[i];

   unchecked {

       ++i; // Note: gas saving

   }

}
```

Bulked pledges are the pledges to be redeemed and at the end, all of those are upserted:

```
/*

   External tx: bulkUpsert and LICR update

*/

uint256[] memory _priorities = _prv6.bulkUpsert(vars._bulkedPledges);
```

However, there are three cases, when pledges are lost.

1. Pledge address is zero

   ```
   if (_pledgeAddr == address(0)) {

      vars._nextICR++;

      continue; /* That rank has been exhausted */

   }
   ```

   This is expected behavior.

2. Pledge ICR is above checkpoint

   ```
   if (vars._nextICR >= vars._checkpoint) {

      // Note: This case conditioned as
   ```

```
    //        vars._activePledgeLength - vars._count == 0 ||
vars._nextICR >= vars._checkpoint

    //        but removed the first clause for simplicity.

    //        It would cause worse gas consumption in pledge-lacking
cases but it's okay.

    break; /* inf loop checker */

}
```

3. Pledge ICR is equal to MCR

```
if (

    vars._nextICR == vars._mcrPercent &&

    _ICRpertenk == vars._mcrPertenk /* priority=realICR=MCR */

) {

    vars._nextICR++;

    vars._nextout = _prv6.rankedQueueNextout(vars._nextICR);

    vars._nextin = _prv6.rankedQueueTotalLen(vars._nextICR);

    continue; /* To avoid "just-on-MCR" pledges */

}
```

Impact

If the pledge is removed from queue, there is no way anyone could liquidate it. Such pledge ratio can inevitable drop below MCR, hurting TCR and affecting CJPY price.

Solution

If pledge is removed from the queue, it should be put back, by appending it to skipped pledges list.

Status

Problem acknowledged by the team, waiting for response.

Collateral can be permanently frozen

Finding ID: **YP-2**
Contract: YamatoWithdrawerV2.sol
Severity: High
Status: Open

Description

If pledge has any debt, the protocol requires collateral to be at least 0.1 ETH.

```
require(
   pledge.coll >= IYamatoV3(yamato()).collFloor(),
   "Deposit or Withdraw can't make pledge less than floor size."
);
```

There is one more problem, if user borrows CJPY, part of stablecoins is transferred to pool as fee, so it is impossible to repay the debt in full.

Impact

If the debt is not possible to be repaid, without CJPY from the market it is impossible to get collateral back.

Solution

There are few ways to deal with such problems.

- **Get fee in collateral currency**. Such solution is not compatible with current system, because protocol uses stablecoins from pool for sweep and redeem method.
- **Remove collateral floor.** With such solution, another problem is created. It is possible to create many pledges, which are more expensive to liquidate, than the value they hold.

- **Introduce contract's pledge.** To understand this solution, let's assume, there are many users with frozen collateral and little debt. All of those users create one big pledge, which will cover all their debts. They have to to pay about 135% of their debt in collateral. At the end they divide stablecoins, repay their debts and withdraw the collateral. Shared pledge has frozen collateral, which is minimal possible collateral backing stablecoins in pool.
  Unfortunately it requires users coordination, which is hard to achieve, so there introducing contract's pledge (at address this) and new exit method manipulating it would solve this problem.

Status

Problem acknowledged by the team, waiting for response.

Pledges can become undercollateralized during a market downturn

Finding ID: **YP-3**
Contract: PriorityRegistryV6.sol, YamatoRedeemerV6.sol
Severity: High
Status: Open

Description

A newly created pledge with initial collateral ratio of at least 186 (MCR + CHECKPOINT_BUFFER + 1) will never be considered for liquidation/debt adjustment.

Impact

The contract's collateral can be griefed by minting CJPY while Ethereum is expensive against JPY and not interacting with the pledge as the price goes down. Historically, Ethereum has been a volatile asset and a 47% decrease of value is highly likely in the coming years. An attacker can hedge the Ethereum collateral locked inside Yamato with an ETH short and will become profitable once the ICR drops below 100 (47% drop in ETH/JPY exchange rate). This would

result in protocol insolvency.

Note that the attacker doesn't have to time the market top perfectly to execute this, as they can keep withdrawing the collateral as ETH price surges, to maintain the 186% ICR. The cost of such an attack lies only in the funding cost of the short position.

Solution

Public method could be added to one of Yamato contracts, its purpose would be to re-evaluate a given pledge's ICR in the Priority Registry and update it.

Redeemer and sweeper can be front runned

Finding ID: **YP-4**
Contract: YamatoRedeemerV4.sol and YamatoSweeperV2.sol
Severity: Medium
Status: Open

Description

Both sweep and redeem methods provides compensation for sender, but both can be front runned.

Impact

Miners can easily front-run each of such requests without any risk. Original transaction is then just a waste of gas. It can disincentivize users to keep collateral ratio healthy.

Solution

Gas compensation should be provided only for those, who want to sweep or redeem using their own funds. In such case, front running is much harder, because it requires most of the mainers to borrow some CJPY (and lock some collateral). If most of miners had some CJPY, problem with front running would not exist anymore.

Borrow fees can be easily avoided

Finding ID: **YP-5**
Contract: YamatoWithdrawerV2.sol
Severity: Medium
Status: Open

Description

Fees depends on ICR. ICR can be high during borrow, but in the next block collateral can be withdrawn without consequences.

Impact

There is no incentive to maintain high ICR, because there is no penalty caused by low ICR.

Solution

Everytime collateral is withdrawn, the ICR before and after withdrawal should be calculated. Then fee should be calculated for both cases, and the difference should be paid.

Status

Problem acknowledged by the team, waiting for response.

Other

Severity classification

We have adopted a severity classification inspired by the Immunefi Vulnerability Severity Classification System - v2. It can be found here.