SALUS SECURITY

MAY 2023

CODE SECURITY ASSESSMENT

SOLV PROTOCOL

Overview

Project Summary

- Name: Solv Protocol
- Version: v3.1
- Platform: EVM-compatible chains
- Language: Solidity
- Repository: https://github.com/solv-finance/solv-contracts-v3
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Solv Protocol - incremental audit
Version	v2
Type	Solidity
Dates	May 15 2023
Logs	May 06 2023; May 15 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	3
Total	5

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Subscriber's purchasedRecords is not updated when issueInfo.purchaseLimitInfo.max == 0	Low	Business logic	Acknowledged
2	Centralization risk	Low	Centralization	Acknowledged
3	Missing zero-value checks	Informational	Validation	Resolved
4	Inconsistent prefixes in error messages	Informational	Code quality	Resolved
5	Redundant code	Informational	Reentrancy	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Subscriber's purchasedRecords is not updated when issueInfo.purchaseLimitInfo.max == 0

Severity: Low	Category: Business logic

Target:
- markets/prime/contracts/IssueMarket.sol

Description

markets/prime/contracts/IssueMarket.sol:L119-L123

```
if (issueInfo.purchaseLimitInfo.max > 0) {
    uint256 purchased = purchasedRecords[vars.issueKey][vars.buyer] + value_;
    require(purchased <= issueInfo.purchaseLimitInfo.max, "IssueMarket: value more than
max");
    purchasedRecords[vars.issueKey][vars.buyer] = purchased;
}
```

The subscribe() function updates the purchasedRecords state variable only in the above code block. If the issueInfo.purchaseLimitInfo has max set to 0, subscribing to the product does not update purchasedRecords.

Recommendation

We recommend tracking a user's purchasedRecords even if issueInfo.purchaseLimitInfo.max is set 0.

Status

This issue has been acknowledged by the team. The team has clarified that when max == 0, there is no maximum limit for purchases, so it's not necessary to track user purchases.

2. Centralization risk

Severity: Low	Category: Centralization
Target: - sft/payable/earn/contracts/EarnConcrete.sol	

Description

The Solv Protocol has introduced the role of supervisor, responsible for setting the interest rate for floating-interest-rate products. The issuer designates the supervisor when issuing a product.

However, it's worth noting that if the supervisor does not set the interest rate for a floating-interest-rate product, the buyer's claimable value would be zero, rendering the funds unclaimable. This situation may arise if the supervisor loses its private key or acts maliciously. In such cases, the funds are locked in the contract.

Recommendation

To mitigate this risk, it's recommended to document it and inform the issuer about the potential of the buyer's funds becoming unclaimable if the supervisor fails to set the interest rate. This would help the issuer take necessary precautions and ensure that the supervisor is trustworthy and capable of fulfilling their role.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Missing zero-value checks

Severity: Informational	Category: Validation

Target:
- sft/abilities/contracts/multi-repayable/MultiRepayableDelegate.sol
- sft/payable/factory/contracts/PayableBeaconFactory.sol

Description

1. The claimTo() function could add a check, `require(claimValue_ != 0, "...")`, to prevent users from mistakenly calling the function with a claimValue_ of zero. This will fail early, saving the caller gas.

2. The transferBeaconOwnership() could add a check, `require(newOwner_ != address(0), "...")`, to prevent mistakenly transferring ownership to the zero address.

Recommendation

Consider adding the checks against zero-value.

Status

The team has resolved this issue in commit 1c5f442.

4. Inconsistent prefixes in error messages

Severity: Informational	Category: Code quality
Target: - sft/payable/earn/contracts/EarnConcrete.sol	

Description

The EarnConcrete contract uses varying prefixes for its error messages.

Specifically:

1. The prefix, “EarnConcrete: ”, is used in line 29-31.
2. The prefix, “PayableConcrete: ”, is used in line 67, 94, 95, 99, and 138.
3. Line 103-105 do not include any prefixes in their error messages.

Recommendation

It is recommended to use a consistent prefix throughout the contract for better clarity and consistency in error messages.

Status

The team has resolved this issue in commit 1c5f442.

5. Redundant code

Severity: Informational	Category: Reentrancy

Target:
- sft/payable/factory/contracts/PayableBeaconFactory.sol
- sft/payable/earn/contracts/EarnConcrete.sol

Description

1. The _self state variable in the PayableBeaconFactory contract is initialized, but it is not used elsewhere. Therefore, it can be removed.

2. sft/payable/earn/contracts/EarnConcrete.sol:L105

```
require(uint8(input_.interestType) < 2, "invalid interest type");
```

The above check is unnecessary because input_.interestType is of type enum InterestType, and uint8(input_.interestType) will not exceed the enum range.

Recommendation

Consider removing the redundant code.

Status

The team has resolved this issue in commit 1c5f442.

Appendix

Appendix 1 - Files in Scope

We performed a diff-audit of the repository at the eafe368 commit against the code at the 3f3ce5c commit.

In scope were the following contracts:

File	SHA-1 hash
markets/prime/contracts/IssueMarket.sol	c89adfe378f8f20ac679031cae03098998b46e16
sft/abilities/contracts/multi-rechargeable/MultiRechargeableConcrete.sol	8878ebfb5ab4bd4a54fee064f458966e6cac7991
sft/payable/earn/contracts/IEarnConcrete.sol	9bd62ed871707990670d1226e31f416f2e39e899
sft/payable/earn/contracts/EarnConcrete.sol	23c93fee20c687a530a8b140c3a0822100196390
sft/payable/earn/contracts/EarnDelegate.sol	4a1505e6901c85930b2eba58add28f67590786ef
sft/payable/factory/contracts/PayableBeaconFactory.sol	ce72fc32d71da022b4d4a70667d9e0cb6399779a