SALUS SECURITY

JAN 2024

CODE SECURITY ASSESSMENT

SOLV PROTOCOL

Overview

Project Summary

- Name: Solv Protocol - Vault Guardian
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository: https://github.com/solv-finance/solv-vault-guardian
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Solv Protocol - Vault Guardian
Version	v3
Type	Solidity
Dates	Jan 04 2024
Logs	Dec 22 2023; Jan 04 2024; Jan 04 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	1
Total informational issues	5
Total	7

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	Centralization risk	Medium	Centralization	Acknowledged
2	SolvVaultGuardian incompatible with >=1.4.0 Safe versions	Low	Configuration	Acknowledged
3	Should check the length of the data before retrieving the selector from it	Informational	Business Logic	Resolved
4	Possible function selector clashing	Informational	Business Logic	Resolved
5	Gas optimization	Informational	Gas Optimization	Resolved
6	Redundant code	Informational	Redundancy	Resolved
7	Incorrect configuration	Informational	Configuration	Resolved

2.2 Notable Findings

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

1. Centralization risk

Severity: Medium	Category: Centralization
Target: - src/utils/Governable.sol	

Description

In the SolvVaultGuardian contract, there is a privileged governor role. The governor role has the ability to:

- set whether setGuard is allowed
- set whether to allow transfers of native tokens
- add a whitelist for native token transfers
- add and remove an authorisation
- disable governor operation

If the governor's private key is compromised, the attacker can exploit the governor role to remove all authorisations and then use forbidGovernance() to disable furthur governance. This would render the safe wallet unable to execute meaningful transactions.

Recommendation

We recommend transferring the governor role to a multi-sig account with a timelock feature for enhanced security. This ensures that no single person has full control over the account and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

2. SolvVaultGuardian incompatible with >=1.4.0 Safe versions

Severity: Low	Category: Configuration
Target: - src/SolvVaultGuardian.sol	

Description

1. Safe wallets created using version 1.4.0 or later are not compatible with the current SolvVaultGuardian implementation. This is because, since version 1.4.0, EIP-165 support has been added to the GuardManager.sol.

Since the SolvVaultGuardian contract does not include the supportsInterface() function, the call to setGuard() from the Safe wallet will revert due to the added supportsInterface() check.

2. The Safe wallet is going to add a module guard feature to its v1.5.0 release. It's adding a checkModuleTransaction() method to the guard to allow checking the module transactions before execution.

Since the SolvVaultGuardian contract does not implement the checkModuleTransaction() function, a call to execTransactionFromModule() from Safe wallets created using version 1.5.0 will revert due to a checkModuleTransaction() check to the guard.

Recommendation

1. Consider adding support for EIP-165 in the SolvVaultGuardian contract. You can refer to the implementation here.

2. Consider including a checkModuleTransaction() function with empty logic in the SolvVaultGuardian contract to make it compatible with future Safe wallet versions.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Should check the length of the data before retrieving the selector from it

Severity: Informational	Category: Business Logic

Target:
- src/common/FunctionAuthorization.sol

Description

When data.length is less than 4, the call is invalid. This check should be performed before getting the selector. However, this check is performed after the selector is retrieved.

src/common/FunctionAuthorization.sol:L153-L184

```
function _authorizationCheckTransactionWithRecursion(
    address from_,
    address to_,
    bytes calldata data_,
    uint256 value_
) internal virtual returns (Type.CheckResult memory result_) {
    if (data_.length == 0) {
        return _checkNativeTransfer(to_, value_);
    }
    bytes4 selector = _getSelector(data_);

    if (_isAllowedSelector(to_, selector) && selector ==
bytes4(keccak256(bytes(SAFE_MULITSEND_FUNC_MULTI_SEND)))) {
        result_ = _checkMultiSend(from_, to_, data_, value_);
    } else {
        if (data_.length < 4) {
            result_.success = false;
            result_.message = "FunctionAuthorization: invalid txData";
            return result_;
        }
        ...
    }
}
```

Recommendation

It is recommended that (data_.length >= 4) is checked before _getSelector.

Status

This issue has been resolved by the team with commit 6219da7.

4. Possible function selector clashing

Severity: Informational	Category: Business Logic
Target: - src/common/FunctionAuthorization.sol	

Description

Selector clashing in Solidity refers to the situation where different functions with different names end up having the same 4-byte identifier at the bytecode level.

src/common/FunctionAuthorization.sol:L165

```
function _authorizationCheckTransactionWithRecursion(
    address from_,
    address to_,
    bytes calldata data_,
    uint256 value_
) internal virtual returns (Type.CheckResult memory result_) {
    ...
    bytes4 selector = _getSelector(data_);
    if (_isAllowedSelector(to_, selector) && selector ==
bytes4(keccak256(bytes(SAFE_MULITSEND_FUNC_MULTI_SEND)))) {
        result_ = _checkMultiSend(from_, to_, data_, value_);
    } else {
        ...
    }
}
```

If the *to* address of the Safe wallet's transaction has a function whose selector clashes with the selector of "multiSend(bytes)", the FunctionAuthorization will consider this transaction as a multiSend call. It will use the _checkMultiSend internal function to handle it, which may cause an unintended revert.

Recommendation

It is recommended to verify if the *to* address is the MultiSend contract address before proceeding with the _checkMultiSend() function.

Status

This issue has been resolved by the team with commit 6219da7.

5. Gas optimization

Severity: Informational	Category: Gas Optimization

Target:
- src/common/FunctionAuthorization.sol

Description

src/common/FunctionAuthorization.sol:L153-L184

```
function _authorizationCheckTransactionWithRecursion(
    address from_,
    address to_,
    bytes calldata data_,
    uint256 value_
) internal virtual returns (Type.CheckResult memory result_) {
    ....
    if (_isAllowedSelector(to_, selector) && selector ==
bytes4(keccak256(bytes(SAFE_MULITSEND_FUNC_MULTI_SEND)))) {
        result_ = _checkMultiSend(from_, to_, data_, value_);
    } else {
        ...
        if (_isAllowedSelector(to_, selector)) {
            ...
        }
        ...
    }
}
```

Frequently, the transactions would go to the highlighted path. However, *_isAllowedSelector(to_, selector)* is executed twice in this case.

Recommendation

It is recommended to refactor the code to save gas.

For example:

```
...
if(!_isAllowedSelector(to_, selector) {
    result_.success = false;
    result_.message = "FunctionAuthorization: not allowed function";
    return;
}

if(selector == bytes4(keccak256(bytes(SAFE_MULITSEND_FUNC_MULTI_SEND)))) {...}
else {...}
...
```

Status

This issue has been resolved by the team with commit 6219da7.

6. Redundant code

Severity: Informational	Category: Redundancy

Target:
- src/authorizations/gmxv2/GMXV2AuthorizationACL.sol
- src/authorizations/SolvOpenEndFundAuthorizationACL.sol

Description

1. There are unused events in the SolvOpenEndFundAuthorizationACL contract:

- RepayablePoolIdAdded(bytes32 indexed repayablePoolId)
- RepayablePoolIdRemoved(bytes32 indexed repayablePoolId)

2. There are unused events in the GMXV2AuthorizationACL contract:

- RemoveGmxPool(address indexed gmToken)

Recommendation

Consider removing the unused events.

Status

This issue has been resolved by the team with commit 250903c.

7. Incorrect configuration

Severity: Informational	Category: Configuration
Target: - foundry.toml	

Description

To use a specific Solidity compiler version for the project, one should specify the solc_version in foundry.toml.

However, in the current foundry.toml configuration file, a *solc-compiler-version* is used, which will be ignored by the Foundry framework. As a result, Foundry will compile the project using an auto-detected Solidity version instead of using Solidity version 0.8.17.

foundry.toml:L5

```
solc-compiler-version = "0.8.17"
```

Recommendation

It is recommended to correct the configuration.

For example:

```
solc_version = "0.8.17"
```

Status

This issue has been resolved by the team with commit 250903c.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 55063e:

File	SHA-1 hash
SolvVaultGuardian.sol	8d00c02bd8c854393245a2b30c3abc455ec5921f
BaseACL.sol	1d5b141f2064ff2a69e76b2d8da6b5770e1c478c
BaseAuthorization.sol	00e8bf91641d3dbb03093797ab1fc4b986e33ac3
FunctionAuthorization.sol	0399eded37d40e64e5ae2861757abfd84952293e
Type.sol	a4e10402db569e2924972962a0e14c28c10dbb9f
Governable.sol	a6351054fce6b500065c5131c99eb40561a728f7
Multicall.sol	c6b69f8c23ada7a424ea692213394f2a26884a4b
GMXV1Authorization.sol	c2768aa52abd8c0f10183119ee5bab33447a49a0
GMXV1AuthorizationACL.sol	f384ce7b8316d0b6876ccb4ce6c0f2fd01adffea
GMXV2Authorization.sol	2c287d9dab8dfb2f8eee91407776ad5dcb8147d4
GMXV2AuthorizationACL.sol	3ef5bd3aea6cd52c492ba21457c76e7763904d17
CoboArgusAdminAuthorization.sol	6fd1594d8a7cfa10551c8df965b39adc8502b2b7
ERC20TransferAuthorization.sol	bfc345f2c3370e007153585b54efc9d770dc97af
SolvOpenEndFundAuthorization.sol	b9efafced4649e366583054471fb98396e5057f2
SolvOpenEndFundAuthorizationACL.sol	6170fba87afbdfb86aa406d947e9479a653a867a