Quantstamp

Solv Protocol- Vault Guardian

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Safe Guard
Timeline	2024-01-29 through 2024-02-09
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Solv Vaul Guardian Contracts Architecture
Source Code	• https://github.com/solv-finance/solv-vault-guardian #9962cc8
Auditors	• Jonathan Mevs Auditing Engineer • Jennifer Wu Auditing Engineer • Nikita Belenkov Auditing Engineer

Documentation quality	Low
Test quality	Medium
Total Findings	20 Fixed: 14 Acknowledged: 5 Mitigated: 1
High severity findings	3 Fixed: 3
Medium severity findings	2 Fixed: 1 Mitigated: 1
Low severity findings	7 Fixed: 3 Acknowledged: 4
Undetermined severity findings	0
Informational findings	8 Fixed: 7 Acknowledged: 1

Summary of Findings

In this audit, we reviewed Solv's Vault Guardian implementation. This Guard provides additional functionality to Solv's Safe multisig by limiting what functions can be invoked on what contracts as well as what tokens can be transferred or approved. These protections are enforced through Authorizations that are added to the Guard that will revert the transaction if any non-approved actions are invoked on the Safe. Currently, the Guard supports integrations with GMXV1, GMXV2, Solv Funds, Agni and Lendle. Also, they support Cobo Argus tooling which allows for roles to be created and added to the Safe.

We identified 2 high severity issues during the audit that were made possible by inadequate authorization checks. Representing the Authorizations as mappings from the target address to the Authorization itself can solve both of these issues. The third high severity issue we found is due to the insufficient test suite. As the Solv developers are the individuals most knowledgeable of the code base, unit tests should be included to confirm intended functionality. Coverage is very low with tests missing for the Agni, GMXV1, Cobo Argus and Lendle Authorizations. During the fix review, we strongly encourage the team to provide an updated test suite with improved coverage and testing of all interactions with their integrations.

Solv's team was very responsive and helpful throughout the audit. We would like to applaud their commitment to security.

Fix Review The Solv team has fixed or acknowledged all issues with sufficient reasoning. SOLV-5 remains mitigated, as Safe 1.3 and Safe 1.4 do not offer support for Guard checks when transactions are invoked through a module. The Solv team is aware of this and is planning on keeping module integration disabled until Safe 1.5 release. We are also pleased to see a significant improvement to the test suite. SOLV-3 provides more details on specifics for future improvement. We applaud Solv's commitment to security.

ID	DESCRIPTION	SEVERITY	STATUS
SOLV-1	Overlapping Authorization Checks	High	Fixed
SOLV-2	Failing Authorization Check Does Not Prohibit Function Call	High	Fixed
SOLV-3	Lack of Comprehensive Testsuite	High	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
SOLV-4	`ERC20ApproveAuthorization` and `ERC3525ApproveAuthorization` Can Potentially Support Arbitrary Transactions	Medium	Fixed
SOLV-5	Bypass Solv Vault Guardian Checks Using Module Execution	Medium	Mitigated
SOLV-6	Insufficient Protection Against Slippage Attacks	Low	Acknowledged
SOLV-7	Privileged Roles and Ownership	Low	Acknowledged
SOLV-8	Guardian Function Selector Insufficient for Upgradeable Contract	Low	Acknowledged
SOLV-9	Inconsistent Native Transfer Checks in Solv Vault Guardian	Low	Acknowledged
SOLV-10	All `authorizations` and `ACL` Should Support the `BaseAuthorization` or `BaseACL` Interface	Low	Fixed
SOLV-11	Gas Concerns	Low	Fixed
SOLV-12	ACL Contracts Should Not Have Any Public Functions that Are Not Implemented by the Protocol Being Checked	Low	Fixed
SOLV-13	`repayWithBalance()` Not Supported upon Initialzation	Informational	Fixed
SOLV-14	Unlocked Pragma	Informational	Fixed
SOLV-15	Allowance Double-Spend Exploit	Informational	Fixed
SOLV-16	It Should Not Be Possible to `setContractACL` if `_allowedContractToFunctions` Does Not Have an Entry for Contract	Informational	Fixed
SOLV-17	It Is Not Possible to Add or Remove Tokens From Whitelist	Informational	Acknowledged
SOLV-18	Code Typos	Informational	Fixed
SOLV-19	'Dead' Code	Informational	Fixed
SOLV-20	`transferGovernance()` should use 2-step process	Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors

- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/solv-finance/solv-vault-guardian(80300ac901a0019448bc87b39cfebde1f27b1025) Files: src/

Files Excluded

Repo: https://github.com/solv-finance/solv-vault-guardian(80300ac901a0019448bc87b39cfebde1f27b1025) Files: test/ lib/

Findings

SOLV-1 Overlapping Authorization Checks

High Fixed

Update

Marked as "Fixed" by the client. Addressed in: `5198188c01af22698f2a2a865138f29cbf553bb1`. The client provided the following explanation:

> Modified as recommended. Removed ERC20 Approve-related functions from GMX V1 and GMX V2. If ERC20 Approve and Transfer are needed, use ERC20Authorization.

The ERC20 functions have been removed from GMX V1 and GMX V2. The implementation of a mapping enforces the independence of the authorizers.

File(s) affected: `GMXV1Authorization.sol`, `GMXV2Authorization.sol`

Description: Upon deployment, both `GMXV1Authorization` and `GMXV2Authorization` add support for the ERC20 transfer and approve functions on a select number of whitelisted tokens. As a result, the Safe can freely invoke any transfer or approve operations on tokens held by the Safe, and bypass any of the necessary logic included in the `ERC20ApproveAuthorization` or `ERC20TransferAuthorization` contracts. Further, only the `GMXV1Authorization` would need to be enabled for there to be transfers, or approvals, of any whitelisted tokens from the SafeAccount. The overlap present can lead to an inconsistent state of approval, due to the fact that one authorizer would reject the transaction, whilst the other would allow it.

Recommendation: Enforce the independence of Authorizers. There should be no overlap of selectors in Authorizers. Approved function selectors should only exist in a single Authorizer. This fix should be applied with SOLV-2 in mind.

SOLV-2 Failing Authorization Check Does Not Prohibit Function Call

High Fixed

Update

Marked as "Fixed" by the client. Addressed in: `d70ee54634af4466feec4a1d7c9716628f60290e`, `c5febb2ad9f3278236f04829aba400e034c8d837`. The client provided the following explanation:

Modified as recommended. The implementation logic has been updated to a mapping (to ⇒ authorization) pattern. The authorization corresponding to 'to' must pass the check for the transaction to be successful, and each 'to' can only have one authorization. If the authorization check fails for any 'to', the transaction is not allowed.

Fixed according to our recommendation. Now, if the authorization associated with the target address fails, the transaction will revert.

File(s) affected: `SolvVaultGuardianBase.sol`

Description: The Solv Vault Guardian is designed to regulate multi-signature operations through a guard mechanism within the Safe. It utilizes the `checkTransaction()` function to validate the legitimacy of operations against configured rules for each `BaseAuthorization`. An operation proceeds if it meets at least one rule. The implementation allows for independent checks across multiple authorization contracts, with the process halting upon the first successful authorization. This design can inadvertently facilitate the execution of operations, such as token transfers and approvals if just one among potentially multiple authorization checks is lenient or explicitly permits the action.

Exploit Scenario:

1. The Safe integrates `ERC20TransferAuthorization` and `GMXV1Authorization`. `ERC20TransferAuthorization` allows token transfers to the safe wallet only. `GMXV1Authorization` allows token transfers.
2. The Safe owners initiate a token transfer to Alice.
3. The `ERC20TransferAuthorization` fails its check, but the `GMXV1Authorization` succeeds.
4. The transaction to the unapproved recipient Alice proceeds due to the successful authorization from `GMXV1Authorization`, despite the initial restrictive conditions set by `ERC20TransferAuthorization`.

Recommendation: If any authorization fails, the execution should revert. All checks should be validated before allowing the execution. Further, we recommend a redesign of the Authorizations, where the target address of the transaction is mapped to a specific Authorizer. This would solve the issue of overlapping authorizations we include in SOLV-1, as the Authorization defined for the target address of the transaction would be the only Authorization considered. That means, that for any token transfer to occur, if enabled, the token's authorization would be the only one consoled. This would also solve this issue as there is only one Authorization to consider for transaction, so there would be no case where the needed Authorization would fail and the action could still take place.

Rather than looping through all authorizations, only the one that is relevant for the target address of the transaction should be checked. Additionally, this implementation will be less gas expensive as there is no more looping necessary. This allows for the scaling and support of a large number of Authorizations.

SOLV-3 Lack of Comprehensive Testsuite

High Fixed

Update

Marked as "Fixed" by the client. Addressed in: `1b0ca40ba2e9bc151dd84d2e48ad320006edba69`, `38cba72829dc1f2f1ae1585ce61c39cbf455fffd`, `c3dc3484b7952b55531627571e31924ef6d8f7bb`, `3856817a39aaf3a217417f1a3cda55dcc7e015eb`, `bdb0aa358584aff086d4c742c9bdbe9896dda136`, `01a94df2600dc940f5845fd9a475d844a513d824`, `34c6517f3030bf72f304e8afcb6984f08df96002`. The client provided the following explanation:

Modified as recommended.

1. Redesigned the testing mechanism by incorporating additional test cases and corresponding contract mockups. The entire testing process no longer necessitates forking Arbitrum.
2. Added SolvVaultGuardianForSafe14.sol to support Safe 1.4.
3. The test case coverage has reached over 90%, run forge coverage --ir-minimum

The client has made significant improvements to the test suite based on our recommendations. Branch coverage can be improved for the following files: AgniAuthorizationACL.sol, GMXV1AuthorizationACL.sol, GMXV2AuthorizationACL.sol, LendleAuthorizationACL.sol, SolvMasterFundAuthorizationACL.sol, SolvOpenEndFundAuthorizationACL.sol, BaseACL.sol, FunctionAuthorization.sol, SolvVaultGuardianBase.sol. We recommend mutation testing to further strengthen the test suite. Additionally, the client should focus on incorporating more integration tests. These tests should verify Safe compatibility with protocol authorizations and ensure that all authorizations are effectively combined for secure accounts.

Description: The current test suite for the Solv Vault Guardian lacks comprehensive coverage potentially leaving the protocol vulnerable to untested edge cases and integration issues. Specifically, the testsuite does not fully address protocol compatibility, function selector accuracy, and various transaction scenarios, including those involving native currency and tokens across different protocols such as Agni, GMXV1, GMXV2, and Lendle.

Identified Gaps:

1. **Lack of Protocol Compatibility Tests**: Lacks tests verifying Agni, GMXV1, GMX2, Lendle, and Cobo Argus protocol integrations.
2. **Lack of Safe Multisend Compatibility Tests**: Lack of tests verifying the guardian's validation of safe multisend functions.
3. **Safe Upgrade to 1.4**: No tests covering the upgrade of a Safe from version 1.3 to 1.4, missing checks for compatibility or security issues.
4. **Gnosis Safe Deployment in Tests**: The test suite relies on a deployed Safe instance, requiring access to an owner's private key, which complicates and potentially compromises testing integrity.

Recommendation: Incorporate additional tests to address the identified gaps in the test suite and aim to achieve a testing coverage of over 90%. Additionally, refactor the test suite to eliminate redundant code and ensure accurate assertion of expected values, thereby enhancing the overall quality and reliability of testing.

SOLV-4

`ERC20ApproveAuthorization` and `ERC3525ApproveAuthorization` Can Potentially Support Arbitrary Transactions

Medium | Fixed

Update

Marked as "Fixed" by the client. Addressed in: `9b4b5c4a53806bb11218af7f258f9d0d13960236`, `bbcfd1f16cb2723378057ba9ea0dcc01685c5ed8`, `b94296f79eef97e2ee0ad6b3c231d697aecdace1`. The client provided the following explanation:

> 1. Modified the functions such as addContractFunc in FunctionAuthorization to be internal. All authorizations, except for the functions hardcoded in the contract, are not allowed to dynamically add any other functions.
> 2. Added selector checks to the _authorizationCheckTransaction function in both ERC20Authorization and ERC3525Authorization.
>
> Further, the fix of SOLV-2 directly fixes this issue.

File(s) affected: `ERC20ApproveAuthorization.sol`, `ERC3525ApproveAuthorization.sol`

Description: Both `ERC20ApproveAuthroization` and `ERC3525ApproveAuthorization` are intended to enforce a whitelist of approved spenders for tokens held by the Safe. However, the lack of verification on the function selector of the transaction being invoked creates the possibility for an arbitrary transaction to be approved. As there is no ACL contract associated with these Authorizations, there is no `preCheck()` call that would fail in the case where a function is trying to be invoked that is not defined by the ACL. As a result, arbitrary functions can be more readily enabled and executed, as they only have to be added as an allowed selector to the Authorization.

The exploitability of this issue is significantly limited by the fact that the Governor address would have to be compromised to support a malicious action, as well as the fact that for the malicious action to be invoked in the first place, there would still have to be the quorum of required signatures met on the Safe. Regardless, the possiblity of such should not be included in the Guard. The following exploit scenario considers the situation where the Governor is trying to transfer all USDT from the Safe, as we think this example is easy to follow. However, it could also work with any function on any contract that the compromised Governor account chooses to add. Although, the function called would still have to have arguments that can be decoded to `(address, uint256)`.

Exploit Scenario:

1. Suppose the Governor address has been compromised. This malicious Governor's goal is to drain all USDT from the Safe, by sending it to an address that is not a whitelisted token receiver.
2. The Governor adds support for transferring USDT to `ERC20ApproveAuthorization` by calling `ERC20ApproveAuthorization.addContractFuncs(address(USDT), address(0x00), "transfer(address,uint256)")`.
3. Through social engineering, or other compromised accounts, the multisig quorum is reached and `USDT.transfer(USDT.balanceOf(address(Safe)))` is invoked on the Safe.
4. The `ERC20TransferAuthorization` fails in the loop found in `SolvVaultGuardianBase`.
5. `ERC20ApproveAuthorization` succeeds in executing the transaction, although the `ERC20TransferAuthorization` does not support the recipient.
6. All USDT held by the Safe has been transferred to an unauthorized address.

Recommendation: `ERC3525ApproveAuthorization` and `ERC20ApproveAuthorization` should check the selector of the transaction data to make sure it matches one of the functions defined to be used by that Authorization.

SOLV-5

Bypass Solv Vault Guardian Checks Using Module Execution

Medium | Mitigated

Update

Marked as "Fixed" by the client. Addressed in: `93448e335cd7ed3e181f70d6641d5649ea7f61c9`. The client provided the following explanation:

> Modified as recommended.
> 1. In SolvVaultGuardianForSafe13.sol and SolvVaultGuardianForSafe14.sol, an 'enableModule' flag has been added, allowing the guardian governor to determine whether adding a module is allowed.
> 2. In Safe 1.3 and 1.4, modules will not invoke the guard. To ensure security, during actual runtime, the governor may disallow the addition of new modules.
> 3. In Safe 1.5, support for checking Module transactions will be introduced. After the official release of Safe 1.5, the Guardian will add support for Safe 1.5.
>
> This issue is inherent to the Safe 1.3 and Safe 1.4 implementations. The Solv team plans to disable modules until Safe 1.5, where the modules will go through verifications included in the Guard.

File(s) affected: `SolvVaultGuardianBase.sol`

Description: Safe can add modules that introduce additional functionalities beyond the core Safe capabilities. While these modules enhance the utility of Safe by allowing custom features, they also carry the ability to execute arbitrary transactions once installed. Currently, transactions initiated by these modules do not undergo guardian checks, presenting a loophole if the `enableModule()` function is permitted by the guardian settings. This lack of guardian oversight for module-executed transactions can be used to bypass Solv vault guardian checks.

Exploit Scenario:

1. The guardian configuration permits the use of `enableModule()`.
2. Safe owners collectively decide to add a new module by invoking `enableModule()`.
3. The installed module performs arbitrary transactions without undergoing the Solv Vault Guardian's checks.

Recommendation: Implement guardian checks for transactions initiated by Safe Modules to ensure all actions are validated by the guardian checks.

SOLV-6 Insufficient Protection Against Slippage Attacks

Low | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> No modification is necessary.
> 1. Guardian ensure the safety by controlling the direction of fund flow
> 2. Guardian does not controls the amount of funds such as transfer limits and slippage.

File(s) affected: `SolvVaultGuardianBase.sol`

Description: The Solv protocol introduces guardian checks through the Authorization mechanism before transactions of the Safe are executed. These checks, depending on the protocol the Safe is interacting with, primarily focus on assessing tokens in, tokens out, and the recipients of swapped funds. The aim is to ensure transactions comply with the allowed types and adhere to the rules set by the Authorization contract. However, the framework does not effectively guard against slippage attacks. It allows Safe owners to set potentially exploitative transaction parameters, such as a zero minimum for tokens out during swaps, potentially bypassing the protective intent of the Authorization checks.

Exploit Scenario:

1. Safe account owners collaborate to initiate a token swap transaction.
2. By setting the minimum amount for tokens out to zero, the owners can exploit a gap in the Authorization checks, which fail to account for slippage.
3. The transaction meets the Authorization checks based on tokens in, tokens out, and receiver criteria.
4. Malicious actors manipulate market prices around the vault's transaction.
5. The transaction proceeds with slippage leading to unexpected losses of funds.

Recommendation: Improve the Authorization mechanism to include explicit slippage controls. Introduce dynamic slippage checks that reflect current market conditions and prevent Safe account owners from manipulating transaction parameters to their advantage. Additionally, this risk should be clearly documented to users.

SOLV-7 Privileged Roles and Ownership

Low | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> The governor privileges of SolvVaultGuardian will be determined through a multi-signature or voting process, followed by a Timelock, and then the Guardian's mode. The multi-signature will be managed by the Advisory Council

File(s) affected: `SolvVaultGuardianForSafe13.sol`, `AgniAuthorization.sol`, `GMXV1Authorization.sol`, `GMXV2Authorization.sol`, `LendleAuthorization.sol`, `ERC20ApproveAuthorization.sol`, `ERC20TransferAuthorization.sol`, `ERC3525ApproveAuthorization.sol`, `SolvOpenEndFundAuthorization.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. Although all of the functions below can be invoked by the Governor role, we would like to note that actually invoking the functions that are added to the guard still has to go through the multisig process, which significantly mitigates the privilege this Governor has.

The Governor can invoke the following functions on the Solv Vault Guard defined in `SolvVaultGaurdianForSafe`:

- `setGuardAllowed()` to toggle whether an address for the Guard of the Safe can be assigned or not.
- `setNativeTokenTransferAllowed()` to toggle whether direct native token transfers can be invoked from the Safe or not.
- `addNativeTokenReceiver()` and `removeNativeTokenReceiver()` to define the set of addresses can be transferred native token directly.
- `addAuthorizations()` and `removeAuthorizations()` to define the set of Authorizations stored in the contract.

- `setAuthorizationEnabled()` allowing the Governor to specify which of the Authorizations stored in the contract will be enabled and therefore enforced.
- `transferGovernance()` to a new address.
- `forbidGovernance()` preventing any Governor privileged calls to ever be called again.

For all Authorizations reviewed, the Governor can invoke the following functions:

- `addContractFuncs()`, `removeContractFuncs()`, `addContractFuncSig()`, and `removeContractFuncSig()` to define a set of contracts and functions in those contracts that can be invoked by the Safe.
- `setContractACL()` to define the address of the ACL that will be used to further verify transactions.
- `transferGovernance()` to a new address.
- `forbidGovernance()` preventing any Governor privileged calls to ever be called again.

In `SolvMasterFundAuthorizationACL`, the Governor can invoke the following functions:

- `addPoolIdWhitelist()` and `removePoolIdWhitelist()` to define the set of Solv pool Ids that will be supported by the Guard.

In `ERC3525ApproveAuthorization` and `ERC20ApproveAuthorization` the Governor can invoke the following functions to change the state of the ACL:

- `addTokenSpenders()` and `removeTokenSpenders()` to define the set of addresses that can be approved to spend tokens from the Safe.

In `ERC20TransferAuthorization` the Governor can invoke the following functions:

- `addTokenReceivers()` and `removeTokenReceivers()` to define the set of which tokens can be transferred to which addresses from the Safe.

Recommendation: We recommend clearly documenting these privileges to end users as well as closely protecting the account assigned to the Governor to prevent the account being compromised which could lead to misuse of the listed privileged functions.

SOLV-8
Guardian Function Selector Insufficient for Upgradeable Contract

Low | Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> No modification is necessary. The Guardian is primarily used for Solv Fund investments. If there are contract upgrades for the corresponding investment targets of the strategy, the investment strategy needs to be reassessed, and the corresponding selector configurations adjusted accordingly. To avoid such issues during operations, simulate execution can be employed. In the next upgrade of the Guardian, support for configuring the implementation contract address via proxy will be added. This will help prevent such issues by checking if the implementation contract address has changed.

File(s) affected: `FunctionAuthorization.sol`

Description: The `FunctionAuthorization._checkSingleTx()` of the Solv Vault Guardian relies on a static mapping of function selectors before proceeding with ACL checks to validate transactions. This design does not accommodate the mutable design of upgradeable contracts, where function selectors might change after upgrades. Particularly for upgradeable contracts with payable fallback functions, this static mapping approach can lead to erroneous behavior, as the guardian may authorize or block transactions based on outdated information. Additionally, if an upgradeable contract were approved, the implementation of the approved function is subject to change.

Recommendation: Implement a dynamic updating mechanism for function selector mappings in upgradeable contract designs or directly reference implementation contracts to ensure mappings stay current with contract interfaces. Alternatively, the team can ensure that no proxy addresses are approved, only implementations.

SOLV-9
Inconsistent Native Transfer Checks in Solv Vault Guardian

Low | Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> No modifications are necessary. When msg.value is greater than 0 and data.length is also greater than 0, the Native Transfer Authorization is not applicable. It will instead enter the check strategy corresponding to Authorization. Therefore, checkNativeTransfer is only intended for cases where ETH is transferred directly.

File(s) affected: `SolvVaultGuardianBase.sol`, `FunctionAuthorization.sol`

Description: The Solv Vault Guardian implements a mechanism to validate native token transfers, through the `_checkNativeTransfer()` function to validate transactions based on a list of approved receivers `nativeTokenReceiver`. This check is invoked only when the transaction's calldata is empty. However, this approach does not account for payable functions where the calldata is non empty, thereby bypassing the `_checkNativeTransfer()` validation when `msg.value` is sent alongside calldata. This design choice introduces a discrepancy in how native token transfers are validated, potentially allowing for native token disbursements without proper authorization checks when calldata is present.

Recommendation: Revise the Solv Vault Guardian to apply `_checkNativeTransfer` validations to every transaction with `msg.value`, irrespective of calldata presence. Additionally, introduce a separate mapping to dictate which function selectors are permitted to include `msg.value`, enhancing the granularity and security of authorization checks for native token transfers.

SOLV-10
All `authorizations` and `ACL` Should Support the `BaseAuthorization` or `BaseACL` Interface

Low Fixed

Update

Marked as "Fixed" by the client. Addressed in: `c4d038c9b0d1decc93abf9f287adba362f5724ed`. The client provided the following explanation:

> Modified as recommended.

Authorizations and ACL now implement ERC-165. Before Authorizations or ACL are set, `supportsInterface()` confirms their expected implementation.

File(s) affected: `SolvVaultGuardianBase.sol`

Description: The `addAuthorizations()` function allows the governor to add authorizers of type `BaseAuthorization`. It is assumed that all `authorizations[i].executor` have an implementation of the `authorizationCheckTransaction()` function. This is important because if the function is not implemented, it could lead to unexpected approvals or denials.

Recommendation: The `authorizations[i].executor` should implement EIP-165. This would allow the `addAuthorizations()` function to check the presence of the implemented function at the executor address via `executor.supportsInterface()`.

SOLV-11 Gas Concerns

Low Fixed

Update

Marked as "Fixed" by the client. Addressed in: `d70ee54634af4466feec4a1d7c9716628f60290e`, `c5febb2ad9f3278236f04829aba400e034c8d837`. The client provided the following explanation:

> Modified as recommended together with Solv-2.

This issue has been fixed along with the SOLV-2 fix.

File(s) affected: `SolvVaultGuardianBase.sol`

Description: One of the core designs of the protocol is the iteration of `BaseAuthorization`, which checks if the transaction should be allowed or not. Each loop iteration calls `executor.authorizationCheckTransaction()`, which in itself does its own checks and calls to ACL check contracts.

This system works if there are not many `BaseAuthorization`. Otherwise, the gas requirement would scale very fast. This could lead to transactions being rejected or normal transactions costing significant gas amounts due to `checkTransaction()` being invoked on all execution calls from the Safe Vault.

Recommendation: The best solution would be to redesign the check system. Our recommendation would be to have a mapping of the target contract and associated `Authorization`. This would solve the gas issue, as the iteration would only be needed for a few `Authorizations`. More details can be found in the recommendation we include for SOLV-2.

SOLV-12
ACL Contracts Should Not Have Any Public Functions that Are Not Implemented by the Protocol Being Checked

Low Fixed

Update

Marked as "Fixed" by the client. Addressed in: `c17d8406d9a5d2b02d16962a4b50e599b04f8dd2`. The client provided the following explanation:

> Modified as recommended.

Fixed as the `checkToken()` functions are now internal.

File(s) affected: `AgniAuthorizationACL.sol`, `LendleAuthorizationACL.sol`

Description: The idea behind ACL contracts is that they are used to check selector call values via the ACL contracts. This is done via a self static call with the expected selector signature. The ACL contracts implement the allowed functions in the contract. It is expected that only those functions should be exposed, whilst in the following contracts, these are also public functions:

1. `checkToken()` in `LendleAuthorizationACL`
2. `checkToken()` in `AgniAuthorizationACL`

Recommendation: Consider if these functions should be publicly callable. If not, remove them.

SOLV-13 `repayWithBalance()` Not Supported upon Initialzation

Informational | Fixed

Update

Marked as "Fixed" by the client. Addressed in: `306010057ec76d1c8546ac7ad48f76e0005ec861`. The client provided the following explanation:

> Modified as recommended. Deleted repayWithBalance() in SolvOpenEndFundAuthorizationACL.

Fixed as this function has been removed.

File(s) affected: `SolvOpenEndFundAuthorization.sol`

Description: Currently, the `SolvOpenEndFundAuthorization` does not initialize with support for the `repayWithBalance()` function despite that function being included in the `SolvOpenEndFundAuthorizationACL`.

Recommendation: If needed, initialize the the Authorization with support for `repayWithBalance()` or if not needed, remove `repayWithBalance()` from the ACL.

SOLV-14 Unlocked Pragma

Informational | Fixed

Update

Marked as "Fixed" by the client. Addressed in: `6a438116a094c459ecd7206ae641949d36b6ebce`. The client provided the following explanation:

> Modified as recommended. Version locked to 0.8.19.

File(s) affected: `All Files`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

SOLV-15 Allowance Double-Spend Exploit

Informational | Fixed

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> No modification is necessary. Solv Guardian only restricts execution permissions and cannot address existing issues with the ERC20 standard.

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain the ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls the `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through the use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`. Furthermore, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set the allowance to `0` first and verify if it was used before setting the new value.

SOLV-16
It Should Not Be Possible to `setContractACL` if `_allowedContractToFunctions` Does Not Have an Entry for Contract

Informational · Fixed

Update

Marked as "Fixed" by the client. Addressed in: `dd77a1105ff26718d69b089e5c3fc50b21f12e5a` . The client provided the following explanation:

> Modified as recommended.

`_setContractACL()` now checks that the contract is the `_contracts` list, before assigning an ACL to it.

File(s) affected: `SolvVaultGuardianBase.sol`

Description: The `_allowedContractToFunctions` stores a mapping of allowed contracts and their corresponding allowed function selectors, while the `_contractACL` mapping stores target contract addresses and corresponding ACL contract addresses.

One can add an address to `_contractACL` mapping via `setContractACL()`, but the function does not check if that target contract address is already in `_allowedContractToFunctions`, which is essential, as unless the contract is in that mapping, the flow will never reach the `_contractACL`.

Recommendation: `setContractACL()` should check if the target address contract exists in `_allowedContractToFunctions`, before adding it.

SOLV-17
It Is Not Possible to Add or Remove Tokens From Whitelist

Informational · Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> No modification is necessary. The tokens to be supported were determined at the time of strategy issuance and do not need to be changed. If a change is indeed required, deploying a new ACL and having the governor set it up would be a simpler and clearer approach.

File(s) affected: `AgniAuthorizationACL.sol`, `LendleAuthorizationACL.sol`

Description: Currently, some of the ACL implementations that allow token whitelisting only allow them to be added via `_addTokenWhitelist()` during deployment. There is no way to modify this token whitelist following deployment. This could be an issue if one of the tokens becomes compromised or is no longer supported, or, if there are additional tokens that would want to be supported by Solv in the future.

Recommendation: Consider if one should be able to remove tokens from the whitelist. If that is the case, make sure any positions are liquidated before tokens are removed, as otherwise, they would be locked in the deposited protocol.

SOLV-18 Code Typos

Informational · Fixed

Update

Marked as "Fixed" by the client. Addressed in: `ddbe9b0dc64a447d6e919add0ac8b9b86a7a3e52` . The client provided the following explanation:

> Modified as recommended.

File(s) affected: `GMXV1Authorization.sol`

Description: `GMX_REWAED_ROUTER_V2` and `GMX_REWAED_ROUTER` in `GMXV1Authorization` .

Recommendation: These address constants should be renamed to `GMX_REWARD_ROUTER_V2` and `GMX_REWARD_ROUTER` .

SOLV-19 'Dead' Code

Informational Fixed

Update

Marked as "Fixed" by the client. Addressed in: `c3eb940540bd86426411584ca0da2914ed71eee0` . The client provided the following explanation:

> Modified as recommended.

Related Issue(s): SWC-131, SWC-135

Description: "Dead" code refers to code which is never executed and hence makes no impact on the final result of running a program. Dead code raises a concern, since either the code is unnecessary or the necessary code's results were ignored.

There is dead code in the following places:

1. `ETH` in `ERC20ApproveAuthorization`
2. `ETH` in `ERC20TransferAuthorization`
3. `GLP_REWAED_ROUTER` in `GMXV1AuthorizationACL`
4. `repayWithBalance()` in `SolvOpenEndFundAuthorizationACL`

Recommendation: We recommend that the team looks into the to further investigate the issue and remove the dead code if not needed.

SOLV-20 `transferGovernance()` should use 2-step process

Informational Fixed

Update

Marked as "Fixed" by the client. Addressed in: `da928cbcf14bd05ca49f87cc94026bf24b1c982a` . The client provided the following explanation:

> Modified as recommended.

File(s) affected: `Governable.sol`

Description: The owner of the contracts can call `transferGovernance()` to transfer the ownership to a new address. If an uncontrollable address is accidentally provided as the new owner address, then the contract will no longer have an active owner, and functions with the `onlyGovernor()` modifier can no longer be executed.

Recommendation: Consider using OpenZeppelin's `Ownable2Step` contract to adopt a two-step ownership pattern in which the new owner must accept their position before the transfer is complete.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – 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 post an immediate risk, but is relevant to security best practices or Defence in Depth.

- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `a51...ce9 ./src/SolvVaultGuardianForSafe13.sol`
- `90a...7aa ./src/external/IOpenFundMarket.sol`
- `da4...129 ./src/external/IERC3525.sol`
- `5d3...d42 ./src/common/Type.sol`
- `f3d...a0a ./src/common/FunctionAuthorization.sol`
- `02a...98e ./src/common/SolvVaultGuardianBase.sol`
- `3cf...12a ./src/common/BaseACL.sol`
- `a24...7d4 ./src/common/BaseAuthorization.sol`
- `17a...b21 ./src/utils/Governable.sol`
- `d18...a4d ./src/utils/Multicall.sol`
- `238...26c ./src/libraries/Path.sol`
- `670...209 ./src/libraries/BytesLib.sol`
- `7c7...d00 ./src/authorizations/ERC20TransferAuthorization.sol`
- `c26...ba4 ./src/authorizations/SolvOpenEndFundAuthorizationACL.sol`
- `980...af7 ./src/authorizations/ERC3525ApproveAuthorization.sol`
- `575...08d ./src/authorizations/CoboArgusAdminAuthorization.sol`
- `4ce...7a1 ./src/authorizations/ERC20ApproveAuthorization.sol`
- `354...1ea ./src/authorizations/SolvOpenEndFundAuthorization.sol`
- `44d...ca9 ./src/authorizations/gmxv1/GMXV1AuthorizationACL.sol`
- `0d7...3d7 ./src/authorizations/gmxv1/GMXV1Authorization.sol`
- `50d...365 ./src/authorizations/solv-master-fund/SolvMasterFundAuthorization.sol`
- `fc2...de6 ./src/authorizations/solv-master-fund/SolvMasterFundAuthorizationACL.sol`
- `504...139 ./src/authorizations/agni/AgniAuthorization.sol`
- `0cd...ef0 ./src/authorizations/agni/AgniAuthorizationACL.sol`
- `bb8...9dc ./src/authorizations/lendle/LendleAuthorizationACL.sol`
- `0ad...043 ./src/authorizations/lendle/LendleAuthorization.sol`
- `033...d7d ./src/authorizations/gmxv2/GMXV2Authorization.sol`
- `2d4...66e ./src/authorizations/gmxv2/GMXV2AuthorizationACL.sol`

Tests

- `bdc...a96 ./test/CEXArbitrageFullFlow.t.sol`
- `5ae...ed8 ./test/SolvVaultGuardian.t.sol`
- `6b9...f04 ./test/external/IOpenFundMarket.sol`
- `55a...75b ./test/external/OpenFundTestHelper.sol`
- `7db...e49 ./test/external/IOpenFundRedemption.sol`
- `75a...550 ./test/common/SolvVaultGuardianBaseTest.sol`
- `1ce...bf2 ./test/authorizations/SolvMasterFundAuthorization.t.sol`
- `f42...e2e ./test/authorizations/ERC20TransferAuthorization.t.sol`
- `728...ea3 ./test/authorizations/GMXV2Authorization.t.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

We filtered the issues highlighted by Slither and consolidated the relevant issues in this report.

Test Suite Results

The test suite relies on deployed Gnosis Safe instances, requiring access to an owner's private key, which complicates the test suite. We modified the test suite setup to deploy a Safe locally to collect test coverage.

Update The test suite has been significantly improved. All 222 tests are passing.

```
[⠃] Compiling...
[⠃] Compiling 109 files with 0.8.19
[⠊] Solc 0.8.19 finished in 19.56s
Compiler run successful!

Running 5 tests for test/libraries/Path.t.sol:PathTest
[PASS] test_DecodeFirstPool() (gas: 28895)
[PASS] test_GetFirstPool() (gas: 64474)
[PASS] test_HasMultiplePools() (gas: 26544)
[PASS] test_NumPools() (gas: 26303)
[PASS] test_SkipToken() (gas: 64555)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 5.63ms

Running 6 tests for test/libraries/BytesLib.t.sol:BytesLibTest
[PASS] test_RevertWhenSliceOutOfBounds() (gas: 3716)
[PASS] test_RevertWhenToAddressOutOfBounds() (gas: 3601)
[PASS] test_RevertWhenToUint24OutOfBounds() (gas: 3578)
[PASS] test_Slice() (gas: 5803)
[PASS] test_ToAddress() (gas: 1058)
[PASS] test_ToUint24() (gas: 924)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 6.23ms

Running 17 tests for test/authorizations/LendleAuthorization.t.sol:LendleAuthorizationTest
[PASS] test_AddTokenWhitelist() (gas: 856312)
[PASS] test_AuthorizationInitialStatus() (gas: 7984)
[PASS] test_Borrow() (gas: 45451)
[PASS] test_Deposit() (gas: 45050)
[PASS] test_Repay() (gas: 44879)
[PASS] test_RevertWhenAddInvalidTokenWhitelist() (gas: 787010)
[PASS] test_RevertWhenBorrowWithInvalidAsset() (gas: 46181)
[PASS] test_RevertWhenBorrowWithInvalidBehalf() (gas: 44939)
[PASS] test_RevertWhenDepositWithInvalidAsset() (gas: 46226)
[PASS] test_RevertWhenDepositWithInvalidBehalf() (gas: 44390)
[PASS] test_RevertWhenRepayWithInvalidAsset() (gas: 46141)
[PASS] test_RevertWhenRepayWithInvalidBehalf() (gas: 44416)
[PASS] test_RevertWhenSwapBorrowRateModeWithInvalidAsset() (gas: 40802)
[PASS] test_RevertWhenWithdrawWithInvalidAsset() (gas: 46731)
[PASS] test_RevertWhenWithdrawWithInvalidBehalf() (gas: 43790)
[PASS] test_SwapBorrowRateMode() (gas: 39548)
[PASS] test_Withdraw() (gas: 44916)
Test result: ok. 17 passed; 0 failed; 0 skipped; finished in 8.11ms
```

```
Running 12 tests for test/authorizations/GMXV1Authorization.t.sol:GMXV1AuthorizationTest
[PASS] test_AuthorizationInitialStatus() (gas: 11094)
[PASS] test_Claim() (gas: 25245)
[PASS] test_Compound() (gas: 25247)
[PASS] test_HandleRewards() (gas: 25923)
[PASS] test_MintAndStakeGlp() (gas: 40480)
[PASS] test_MintAndStakeGlpETH() (gas: 39494)
[PASS] test_RevertMintAndStakeGlpWithInvalidToken() (gas: 41667)
[PASS] test_RevertWhenUnstakeAndRedeemGlpETHToInvalidReceiver() (gas: 45990)
[PASS] test_RevertWhenUnstakeAndRedeemGlpToInvalidReceiver() (gas: 46586)
[PASS] test_RevertWhenUnstakeAndRedeemGlpWithInvalidToken() (gas: 43852)
[PASS] test_UnstakeAndRedeemGlp() (gas: 44815)
[PASS] test_UnstakeAndRedeemGlpETH() (gas: 44209)
Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 6.77ms

Running 6 tests for test/authorizations/ERC3525Authorization.t.sol:ERC3525AuthorizationTest
[PASS] test_ApproveErc3525Token() (gas: 56271)
[PASS] test_AuthorizationInitialStatus() (gas: 21674)
[PASS] test_RevertWhenApproveToUnauthorizedSpender() (gas: 42986)
[PASS] test_RevertWhenApproveWithUnauthorizedToken() (gas: 29480)
[PASS] test_RevertWhenUpdateTokenSpendersByNonGovernor() (gas: 23284)
[PASS] test_UpdateTokenSpenders() (gas: 2107535)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 7.77ms

Running 12 tests for test/authorizations/ERC20Authorization.t.sol:ERC20AuthorizationTest
[PASS] test_ApproveErc20Token() (gas: 49629)
[PASS] test_AuthorizationInitialStatus() (gas: 34977)
[PASS] test_RemoveToken() (gas: 3045259)
[PASS] test_RevertWhenApproveToUnauthorizedSpender() (gas: 52109)
[PASS] test_RevertWhenApproveUnauthorizedToken() (gas: 42134)
[PASS] test_RevertWhenTransferToUnauthorizedReceiver() (gas: 30086)
[PASS] test_RevertWhenTransferUnauthorizedToken() (gas: 24139)
[PASS] test_RevertWhenUpdateTokenReceiversByNonGovernor() (gas: 23739)
[PASS] test_RevertWhenUpdateTokenSpendersByNonGovernor() (gas: 23577)
[PASS] test_TransferErc20Token() (gas: 29231)
[PASS] test_UpdateTokenReceivers() (gas: 2839697)
[PASS] test_UpdateTokenSpenders() (gas: 3045113)
Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 6.89ms

Running 9 tests for test/authorizations/SolvMasterFundAuthorization.t.sol:SolvMasterFundAuthorizationTest
[PASS] test_AddPoolIdWhitelist() (gas: 138776)
[PASS] test_Redeem() (gas: 40517)
[PASS] test_RemovePoolIdWhitelist() (gas: 110644)
[PASS] test_RevertWhenRedeemToUnheldRedemptionId() (gas: 65422)
[PASS] test_RevertWhenSubscribeToUnallowedPool() (gas: 41744)
[PASS] test_RevertWhenSubscribeToUnheldShareId() (gas: 65500)
[PASS] test_RevokeRedeem() (gas: 39470)
[PASS] test_Subscribe() (gas: 40627)
[PASS] test_SubscribeWithGivenShareId() (gas: 63787)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 8.65ms

Running 16 tests for test/common/FunctionAuthorization.t.sol:FunctionAuthorizationTest
[PASS] test_AddFuncsSigWithAcls() (gas: 189567)
[PASS] test_AddFuncsSigWithNoAcl() (gas: 161465)
[PASS] test_AddFuncsWithAcls() (gas: 195605)
[PASS] test_AddFuncsWithNoAcl() (gas: 167552)
[PASS] test_Fallback() (gas: 5680)
[PASS] test_OnlyCaller() (gas: 16462)
[PASS] test_RemoveFuncs() (gas: 159566)
[PASS] test_RemoveFuncsSig() (gas: 152032)
[PASS] test_RevertWhenAddEmptyFuncs() (gas: 11820)
[PASS] test_RevertWhenAddEmptyFuncsSig() (gas: 11658)
[PASS] test_RevertWhenRemoveEmptyFuncs() (gas: 11671)
[PASS] test_RevertWhenRemoveEmptyFuncsSig() (gas: 11369)
[PASS] test_RevertWhenSetAclToUnauthorizedContract() (gas: 15243)
[PASS] test_RevertWhenSetInvalidAcl() (gas: 198826)
[PASS] test_SetAcl() (gas: 191527)
[PASS] test_SupportsInterface() (gas: 7902)
Test result: ok. 16 passed; 0 failed; 0 skipped; finished in 8.00ms

Running 14 tests for test/authorizations/AgniAuthorization.t.sol:AgniAuthorizationTest
```

```
[PASS] test_AddTokenWhitelist() (gas: 1146216)
[PASS] test_AuthorizationInitialStatus() (gas: 7940)
[PASS] test_ExactInput() (gas: 59022)
[PASS] test_ExactInputSingleParams() (gas: 51349)
[PASS] test_RevertWhenAddInvalidTokenWhitelist() (gas: 1076875)
[PASS] test_RevertWhenExactInputSingleParamsWithEthValue() (gas: 45368)
[PASS] test_RevertWhenExactInputSingleParamsWithInvalidRecipient() (gas: 47679)
[PASS] test_RevertWhenExactInputSingleParamsWithInvalidTokenIn() (gas: 50128)
[PASS] test_RevertWhenExactInputSingleParamsWithInvalidTokenOut() (gas: 52574)
[PASS] test_RevertWhenExactInputWithEthValue() (gas: 46749)
[PASS] test_RevertWhenExactInputWithInvalidRecipient() (gas: 49049)
[PASS] test_RevertWhenExactInputWithInvalidTokenIn() (gas: 53042)
[PASS] test_RevertWhenExactInputWithInvalidTokenInBetween() (gas: 55322)
[PASS] test_RevertWhenExactInputWithInvalidTokenOut() (gas: 60287)
Test result: ok. 14 passed; 0 failed; 0 skipped; finished in 8.30ms

Running 28 tests for test/authorizations/GMXV2Authorization.t.sol:GMXV2AuthorizationTest
[PASS] test_AddGmxPool() (gas: 1724365)
[PASS] test_AuthorizationInitialStatus() (gas: 26065)
[PASS] test_BuyGMToken() (gas: 73972)
[PASS] test_RevertWhenAddGmxPoolWithInvalidTokenAddress() (gas: 1684906)
[PASS] test_RevertWhenCall_CreateDeposit_Directly() (gas: 27672)
[PASS] test_RevertWhenCall_CreateWithdrawal_Directly() (gas: 27495)
[PASS] test_RevertWhenCall_SendTokens_Directly() (gas: 24039)
[PASS] test_RevertWhenCall_SendWnt_Directly() (gas: 23926)
[PASS] test_RevertWhenDeposit_CreateWithInvalidFallbackContract() (gas: 69528)
[PASS] test_RevertWhenDeposit_CreateWithInvalidGmTokenReceiver() (gas: 71489)
[PASS] test_RevertWhenDeposit_CreateWithUnauthorizedPool() (gas: 67689)
[PASS] test_RevertWhenDeposit_NotSendToken_InNotEthCase() (gas: 70985)
[PASS] test_RevertWhenDeposit_NotSendWnt() (gas: 69353)
[PASS] test_RevertWhenDeposit_SendTokenToInvalidAddress() (gas: 77683)
[PASS] test_RevertWhenDeposit_SendUnauthorizedToken() (gas: 75337)
[PASS] test_RevertWhenDeposit_SendWntInExcess() (gas: 71516)
[PASS] test_RevertWhenDeposit_SendWntToInvalidAddress() (gas: 73599)
[PASS] test_RevertWhenMulticallWithInvalidSelector() (gas: 55274)
[PASS] test_RevertWhenWithdraw_CreateWithInvalidFallbackContract() (gas: 68798)
[PASS] test_RevertWhenWithdraw_CreateWithInvalidTokenReceiver() (gas: 70692)
[PASS] test_RevertWhenWithdraw_CreateWithUnauthorizedPool() (gas: 66905)
[PASS] test_RevertWhenWithdraw_NotSendToken() (gas: 70378)
[PASS] test_RevertWhenWithdraw_NotSendWnt() (gas: 68703)
[PASS] test_RevertWhenWithdraw_SendIncorrectToken() (gas: 72235)
[PASS] test_RevertWhenWithdraw_SendTokenToInvalidAddress() (gas: 74584)
[PASS] test_RevertWhenWithdraw_SendWntInExcess() (gas: 70889)
[PASS] test_RevertWhenWithdraw_SendWntToInvalidAddress() (gas: 73035)
[PASS] test_SellGMToken() (gas: 70871)
Test result: ok. 28 passed; 0 failed; 0 skipped; finished in 8.45ms

Running 8 tests for test/authorizations/CoboArgusAdminAuthorization.t.sol:CoboArgusAdminAuthorizationTest
[PASS] test_AddAuthorizer() (gas: 26021)
[PASS] test_AddFuncAuthorizer() (gas: 26839)
[PASS] test_AddPoolAddresses() (gas: 25675)
[PASS] test_AddRoles() (gas: 25631)
[PASS] test_CreateAuthorizer() (gas: 25978)
[PASS] test_GrantRoles() (gas: 26115)
[PASS] test_InitArgus() (gas: 25685)
[PASS] test_RevokeRoles() (gas: 26114)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 1.54ms

Running 7 tests for test/utils/Governable.t.sol:GovernableTest
[PASS] test_ForbidGovernance() (gas: 15221)
[PASS] test_Govern() (gas: 23024)
[PASS] test_RevertWhenAcceptGovernanceByNonPendingGovernor() (gas: 26370)
[PASS] test_RevertWhenForbidGovernanceByNonGovernor() (gas: 15728)
[PASS] test_RevertWhenGovernanceForbidden() (gas: 17030)
[PASS] test_RevertWhenTransferGovernanceByNonGovernor() (gas: 15855)
[PASS] test_TransferGovernance() (gas: 30428)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 1.13ms

Running 20 tests for
test/authorizations/SolvOpenEndFundAuthorization.t.sol:SolvOpenEndFundAuthorizationTest
[PASS] test_AuthorizationInitialStatus() (gas: 28359)
[PASS] test_RepayERC20ToRedemption() (gas: 62029)
```

```
[PASS] test_RepayERC20ToShare() (gas: 74405)
[PASS] test_RepayETHToRedemption() (gas: 62067)
[PASS] test_RepayETHToShare() (gas: 73866)
[PASS] test_RepayToRedemptionWhenCurrencyBalanceIsNotZero() (gas: 63338)
[PASS] test_RepayToRedemptionWhenInterestRateIsNegative() (gas: 63672)
[PASS] test_RepayToShareWhenCurrencyBalanceIsNotZero() (gas: 75740)
[PASS] test_RepayToShareWhenInterestRateIsNegative() (gas: 76668)
[PASS] test_RevertWhenOverRepayToRedemption() (gas: 63931)
[PASS] test_RevertWhenOverRepayToShare() (gas: 76203)
[PASS] test_RevertWhenRepayERC20ToRedemptionWithETHValue() (gas: 54594)
[PASS] test_RevertWhenRepayERC20ToShareWithETHValue() (gas: 59112)
[PASS] test_RevertWhenRepayETHToRedemptionWithInvalidValue() (gas: 54475)
[PASS] test_RevertWhenRepayETHToShareWithInvalidValue() (gas: 59036)
[PASS] test_RevertWhenRepayToInvalidContractType() (gas: 64981)
[PASS] test_RevertWhenRepayToRedemptionWhereInterestRateNotSet() (gas: 62012)
[PASS] test_RevertWhenRepayToRedemptionWithInvalidPoolId() (gas: 63622)
[PASS] test_RevertWhenRepayToShareWhereInterestRateNotSet() (gas: 70250)
[PASS] test_RevertWhenRepayToShareWithInvalidPoolId() (gas: 65722)
Test result: ok. 20 passed; 0 failed; 0 skipped; finished in 6.05ms

Running 31 tests for test/integration/SolvVaultGuardianForSafe14.t.sol:SolvVaultGuardianForSafe14Test
[PASS] test_AddAndRemoveAuthorization() (gas: 1310120)
[PASS] test_AddAndRemoveContractACL() (gas: 820339)
[PASS] test_AllowEnableModuleStatus() (gas: 20837)
[PASS] test_AllowSetGuardStatus() (gas: 20964)
[PASS] test_AuthorizationCheckSuccess() (gas: 1301573)
[PASS] test_GuardianInitialStatus() (gas: 11720)
[PASS] test_GuardianMulticall() (gas: 32630)
[PASS] test_GuardianMulticallShouldRevert() (gas: 20089)
[PASS] test_MultiSend() (gas: 91779)
[PASS] test_NativeTokenTransferStatus() (gas: 41932)
[PASS] test_NestedMultiSend() (gas: 111909)
[PASS] test_RevertEnableModuleWhenDisabled() (gas: 2755114)
[PASS] test_RevertWhenAddNativeTokenReceiversByNonGovernor() (gas: 25587)
[PASS] test_RevertWhenAllowSetGuardByNonGovernor() (gas: 24426)
[PASS] test_RevertWhenAllowTransferNativeTokenByNonGovernor() (gas: 24412)
[PASS] test_RevertWhenAnySendFails() (gas: 82847)
[PASS] test_RevertWhenAuthorizationCheckFail() (gas: 1296576)
[PASS] test_RevertWhenAuthorizationWithout165() (gas: 127292)
[PASS] test_RevertWhenDataLengthLessThanFourBytes() (gas: 59765)
[PASS] test_RevertWhenReceiverIsNotInWhitelist() (gas: 142292)
[PASS] test_RevertWhenSetGuardIsNotAllowedInInitialState() (gas: 5500517)
[PASS] test_RevertWhenSetGuardWhenDisabled() (gas: 2795468)
[PASS] test_RevertWhenTransferNativeTokenInInitialState() (gas: 98918)
[PASS] test_RevertWhenTransferNativeTokenIsDisabled() (gas: 95777)
[PASS] test_SetGuard() (gas: 2822509)
[PASS] test_SetGuardWhenReenabled() (gas: 5520180)
[PASS] test_SetModuleWhenEnable() (gas: 111032)
[PASS] test_TransferNativeToken1() (gas: 135787)
[PASS] test_TransferNativeTokenToSelf() (gas: 67822)
[PASS] test_addAndRemoveContractFunc() (gas: 302763)
[PASS] test_addAndRemoveContractFuncSig() (gas: 302668)
Test result: ok. 31 passed; 0 failed; 0 skipped; finished in 8.67ms

Running 31 tests for test/integration/SolvVaultGuardianForSafe13.t.sol:SolvVaultGuardianForSafe13Test
[PASS] test_AddAndRemoveAuthorization() (gas: 1310181)
[PASS] test_AddAndRemoveContractACL() (gas: 820281)
[PASS] test_AllowEnableModuleStatus() (gas: 20837)
[PASS] test_AllowSetGuardStatus() (gas: 20964)
[PASS] test_AuthorizationCheckSuccess() (gas: 1301514)
[PASS] test_GuardianInitialStatus() (gas: 11720)
[PASS] test_GuardianMulticall() (gas: 32630)
[PASS] test_GuardianMulticallShouldRevert() (gas: 20089)
[PASS] test_MultiSend() (gas: 91484)
[PASS] test_NativeTokenTransferStatus() (gas: 41932)
[PASS] test_NestedMultiSend() (gas: 111496)
[PASS] test_RevertEnableModuleWhenDisabled() (gas: 2755174)
[PASS] test_RevertWhenAddNativeTokenReceiversByNonGovernor() (gas: 25587)
[PASS] test_RevertWhenAllowSetGuardByNonGovernor() (gas: 24426)
[PASS] test_RevertWhenAllowTransferNativeTokenByNonGovernor() (gas: 24412)
[PASS] test_RevertWhenAnySendFails() (gas: 82907)
[PASS] test_RevertWhenAuthorizationCheckFail() (gas: 1296636)
```

```
[PASS] test_RevertWhenAuthorizationWithout165() (gas: 127292)
[PASS] test_RevertWhenDataLengthLessThanFourBytes() (gas: 59825)
[PASS] test_RevertWhenReceiverIsNotInWhitelist() (gas: 142412)
[PASS] test_RevertWhenSetGuardIsNotAllowedInInitialState() (gas: 5499593)
[PASS] test_RevertWhenSetGuardWhenDisabled() (gas: 2795588)
[PASS] test_RevertWhenTransferNativeTokenInInitialState() (gas: 99038)
[PASS] test_RevertWhenTransferNativeTokenIsDisabled() (gas: 95837)
[PASS] test_SetGuard() (gas: 2821277)
[PASS] test_SetGuardWhenReenabled() (gas: 5518092)
[PASS] test_SetModuleWhenEnable() (gas: 110893)
[PASS] test_TransferNativeToken1() (gas: 135728)
[PASS] test_TransferNativeTokenToSelf() (gas: 67763)
[PASS] test_addAndRemoveContractFunc() (gas: 302812)
[PASS] test_addAndRemoveContractFuncSig() (gas: 302716)
Test result: ok. 31 passed; 0 failed; 0 skipped; finished in 11.49ms

Ran 15 test suites: 222 tests passed, 0 failed, 0 skipped (222 total tests)
```

Code Coverage

The code coverage is very low. We recommend the team improve their test suite to achieve at least 90% branch coverage, and ensure all integrations are properly tested.

Update The SOLV team has significantly improved their test suite. SOLV-3 includes more details for further test improvement.

File	% Lines	% Statements	% Branches	% Funcs
src/SolvVaultGuardianForSafe13.sol	100.00% (**2/**2)	100.00% (**2/**2)	100.00% (**0/**0)	50.00% (**1/**2)
src/SolvVaultGuardianForSafe14.sol	0.00% (**0/**4)	0.00% (**0/**6)	100.00% (**0/**0)	0.00% (**0/**3)
src/authorizations/ERC20Authorization.sol	100.00% (**53/**53)	100.00% (**78/**78)	100.00% (**24/**24)	100.00% (**18/**18)
src/authorizations/ERC3525Authorization.sol	100.00% (**35/**35)	100.00% (**47/**47)	100.00% (**20/**20)	100.00% (**9/**9)
src/authorizations/agni/AgniAuthorizationACL.sol	88.89% (**16/**18)	90.00% (**18/**20)	22.22% (**4/**18)	100.00% (**4/**4)
src/authorizations/gmxv1/GMXV1AuthorizationACL.sol	100.00% (**6/**6)	100.00% (**6/**6)	0.00% (**0/**12)	100.00% (**4/**4)
src/authorizations/gmxv2/GMXV2AuthorizationACL.sol	100.00% (**39/**39)	100.00% (**47/**47)	26.09% (**12/**46)	100.00% (**5/**5)
src/authorizations/lendle/LendleAuthorizationACL.sol	100.00% (**13/**13)	100.00% (**14/**14)	9.09% (**2/**22)	100.00% (**7/**7)
src/authorizations/solv-master-**fund/**SolvMasterFundAuthorizationACL.sol	100.00% (**21/**21)	100.00% (**31/**31)	44.44% (**8/**18)	100.00% (**9/**9)
src/authorizations/solv-open-end-**fund/**SolvOpenEndFundAuthorizationACL.sol	100.00% (**35/**35)	100.00% (**66/**66)	27.27% (**6/**22)	100.00% (**5/**5)
src/common/BaseACL.sol	95.00% (**19/**20)	96.30% (**26/**27)	37.50% (**3/**8)	88.89% (**8/**9)

File	% Lines	% Statements	% Branches	% Funcs
src/common/BaseAuthorization.sol	100.00% (**2/**2)	100.00% (**6/**6)	100.00% (**0/**0)	66.67% (**2/**3)
src/common/FunctionAuthorization.sol	98.31% (**58/**59)	98.80% (**82/**83)	50.00% (**18/**36)	100.00% (**13/**13)
src/common/SolvVaultGuardianBase.sol	98.82% (**84/**85)	99.10% (**110/**111)	78.12% (**25/**32)	100.00% (**22/**22)
src/libraries/BytesLib.sol	0.00% (**0/**15)	0.00% (**0/**15)	0.00% (**0/**14)	0.00% (**0/**3)
src/libraries/Path.sol	0.00% (**0/**7)	0.00% (**0/**10)	100.00% (**0/**0)	0.00% (**0/**5)
src/utils/Governable.sol	100.00% (**6/**6)	100.00% (**6/**6)	100.00% (**0/**0)	100.00% (**3/**3)
src/utils/Multicall.sol	63.64% (**14/**22)	70.37% (**19/**27)	62.50% (**5/**8)	80.00% (**4/**5)
test/authorizations/AgniAuthorization.t.sol	100.00% (**2/**2)	100.00% (**3/**3)	100.00% (**0/**0)	100.00% (**2/**2)
test/authorizations/GMXV2Authorization.t.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
test/authorizations/LendleAuthorization.t.sol	100.00% (**2/**2)	100.00% (**3/**3)	100.00% (**0/**0)	100.00% (**2/**2)
test/common/AuthorizationTestBase.sol	0.00% (**0/**12)	0.00% (**0/**14)	100.00% (**0/**0)	0.00% (**0/**2)
test/common/FunctionAuthorization.t.sol	100.00% (**7/**7)	100.00% (**9/**9)	100.00% (**0/**0)	100.00% (**7/**7)
test/integration/SolvVaultGuardianForSafe13.t.sol	100.00% (**31/**31)	100.00% (**47/**47)	100.00% (**0/**0)	100.00% (**10/**10)
test/integration/SolvVaultGuardianForSafe14.t.sol	96.88% (**31/**32)	97.92% (**47/**48)	100.00% (**0/**0)	100.00% (**11/**11)
test/integration/SolvVaultGuardianTestBase.sol	75.76% (**25/**33)	70.73% (**29/**41)	100.00% (**0/**0)	63.64% (**7/**11)
test/integration/SolvVaultGuardianTestCommonCase.sol	100.00% (**2/**2)	100.00% (**3/**3)	50.00% (**1/**2)	33.33% (**1/**3)
Total	89.36% (**504/**564)	90.79% (**700/**771)	45.39% (**128/**282)	87.08% (**155/**178)

Changelog

- 2024-02-13 - Initial report
- 2024-03-25 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in

formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp