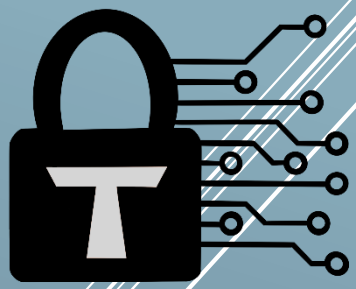
Trust Security

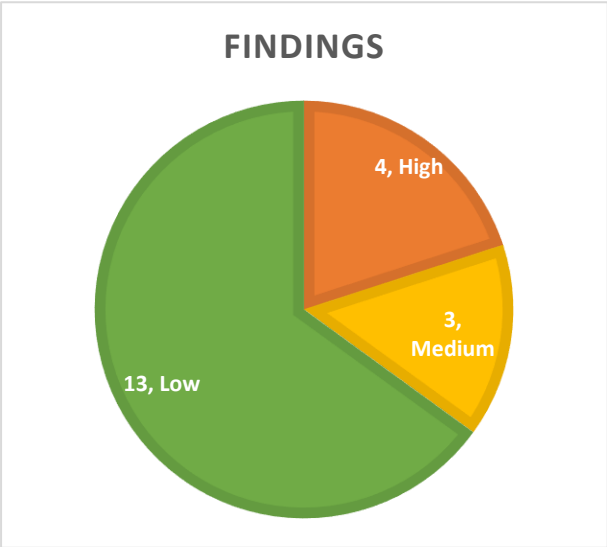


Smart Contract Audit

ZKsync SSO

26/11/25

Executive summary

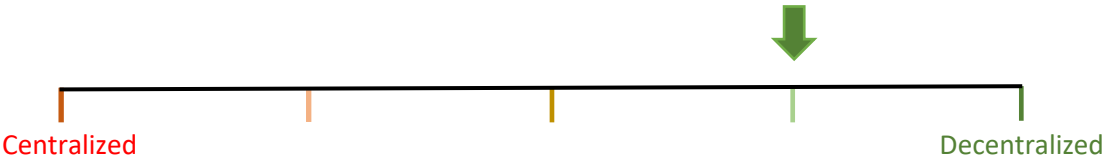


Category	Account Abstraction
Audited file count	14
Lines of Code	1498
Auditor	Trust Rvierdiiev
Time period	03/11/25 – 11/11/25

Findings

Severity	Total	Fixed	Acknowledged
High	4	4	-
Medium	3	3	-
Low	13	9	4

Centralization score



Signature

Digitally signed by Trust
Date: 2025.11.26 11:35:41 +02'00'

Document properties

Versioning

Version	Date	Description
0.1	11/11/25	Client report
0.2	24/11/25	Mitigation review
0.3	26/11/25	Final mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- core/*
- libraries/*
- modules/*
- ModularSmartAccount.sol
- MSAFactory.sol

Repository details

- **Repository URL:** https://github.com/matter-labs/zksync-sso-contracts/
- **Commit hash:** 263cacf5b5b720144773f2f0bd3929b0fff497e4
- **Mitigation review commit hash:** 37a4a9aa14fbb458cae159d4c87834df6998c0c8
- **2nd Mitigation review commit hash:** 4799a3e39e99dc6009e4066ce2da86cc98af4d4e

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

About the Auditors

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys sharing knowledge and experience with aspiring auditors through X or the Trust Security blog.

Rvierdiiev is a Web3 security researcher who participated in numerous public audit contests on multiple platforms and has a proven track record of experience. He has placed at the top of the C4 leaderboard for 2024.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project kept code as simple as possible, reducing attack risks
Documentation	**Excellent**	Project is mostly very well documented.
Best practices	**Excellent**	Project consistently adheres to industry standards.
Centralization risks	**Good**	Users may choose the desired level of decentralization.

Findings

High severity findings

TRST-H-1 Session limits can be bypassed due to flawed time constraint merging

- **Category:** Logical flaws
- **Source:** SessionKeyValidator.sol
- **Status:** Fixed

Description

The SessionKeyValidator operates by passing a series of period identifiers to each **UsageLimit** check, each determining a subrange of timestamps where the execution is valid for that particular limit. In the SessionLib, *shrinkRange()* is used to get the overlapping range from two ranges. This reduces the accepted range for all transfer policies, call policies, and constraints. Back in the validator, it tries to shrink this reduced range with the fee limit range.

```
(uint48 newValidAfter, uint48 newValidUntil) =
    sessions[sessionHash].validateFeeLimit(userOp, spec, periodIds[0]);
validAfter = newValidAfter > validAfter ? validAfter : newValidAfter;
validUntil = newValidUntil < validUntil ? validUntil : newValidUntil;
```

Unfortunately, the logic above is backwards – it takes the smaller of the two **validAfter** values and the larger of the **validUntil** values, expanding the range. Therefore, a transaction will be accepted if the current timestamp is anywhere between the lowest and highest of the two ranges. For a simple bypass, consider an Allowance-based call value, with a Lifetime-based fee value. An attacker could loop over period IDs in the past or future to accumulated unsanctioned call value, as long as the fees do not reach the **Lifetime** limit.

Recommended mitigation

Consider using the *shrinkRange()* to perform the final shrinking correctly.

Team response

Fixed.

Mitigation review

Issue has been addressed as recommended.

TRST-H-2 A LimitType of Unlimited will always revert validation

- **Category:** Logical flaws
- **Source:** SessionLib.sol
- **Status:** Fixed

Description

In *checkAndUpdate()*, the **UsageTracker** is updated based on the limit type, and an applicable time range is returned. There is an issue where the logic is not safe if passed a **limitType= Unlimited**. In this case, no usage check is needed, but the **validUntil** return value must be updated to the **max value** instead of the default **zero value**. Currently, any usage of **Unlimited** type should result in an impossible validation condition, assuming other bugs are fixed. The type can be used when only wishing to cap the maximum value per transaction, or for example when opting to verify a constraint condition without caring for tracking a total.

Recommended mitigation

In case of **Unlimited**, set **validUntil** to **type(uint48).max** and return.

Team response

Fixed.

Mitigation review

Issue has been addressed as suggested.

TRST-H-3 Fee limit check can be bypassed leading to complete draining of native token from a low-trust session

- **Category:** Logical flaws
- **Source:** SessionLib.sol
- **Status:** Fixed

Description

The *validateFeeLimit()* checks fee spending is acceptable. It performs the logic below:

```
if (userOp.paymasterAndData.length == 0) {
    uint256 gasPrice = userOp.gasPrice();
    uint256 gasLimit = userOp.unpackVerificationGasLimit() +
userOp.unpackCallGasLimit();
    uint256 fee = gasPrice * gasLimit;
    // slither-disable-next-line unused-return
    return spec.feeLimit.checkAndUpdate(state.fee, fee, periodId);
} else {
    return (0, type(uint48).max);
}
```

Note that the **gasLimit** includes the verification gas limit and call gas limit, but it ignores the **preVerificationGas** which is always paid by the account to the bundler. This causes underestimation and potential leakage of fees from the account. In fact, it allows complete draining of the native token by passing it under **preVerificationGas**, while delivering the ERC4337 transaction privately by the attacker to the mempool, earning the value as a bundler fee.

Recommended mitigation

Include **preVerificationGas** in the fee calculation logic.

Team response

Fixed.

Mitigation review

Issue has been addressed as suggested.

TRST-H-4 Session validation logic is completely bypassed due to lack of sufficient calldata validation

- **Category:** Validation flaws
- **Source:** SessionLib.sol
- **Status:** Fixed

Description

After the *validate()* function has been refactored to address TRST-R-12, hard-coded offsets are used for access to the calldata for validation:

```
// Function called: execute(bytes32 mode, bytes calldata data)
// - first 4 bytes: selector
// - next 32 bytes: mode
// - next 32 bytes: data offset
// - next 32 bytes: data length
// - next 32 bytes: data
uint256 lengthOffset = 4 + 32 + 32;
uint256 dataOffset = 4 + 32 + 32 + 32;
uint256 length =
uint256(bytes32(userOp.callData[lengthOffset:lengthOffset +
32]));
(address target, uint256 value, bytes calldata callData) =
    LibERC7579.decodeSingle(userOp.callData[dataOffset:dataOffset
+ length]);
```

The code lacks crucial validation that the data offset pointer leads to the assumed address of **4 + 32 + 32 +32**. In fact, it can lead to a location later in the calldata, which will be the one used during the *execute()* call on the account. Meanwhile, the validation would be on values that could be planted as legitimate to pass the checks but have no effect on execution.

The Solidity compiler does not ensure bytes are tightly packed. The intention in R-12, which remains, is that the data offset pointer must be required to have a hard-coded pointer value, not that the data is assumed to be at a hard-coded location.

Recommended mitigation

As described above, ensure calldata bytes **[4+32:4+32+32]** point to **4+32+32**.

Team response

Fixed.

Mitigation review

Issue has been addressed as suggested.

Medium severity findings

TRST-M-1 Session creations can suffer DoS attacks

- **Category:** Frontrunning issues
- **Source:** SessionKeyValidator.sol
- **Status:** Fixed

Description

Users can create a session by passing a SessionSpec. The session hash is stored in the global **sessionSigner** mapping under the signer provided in the spec, without any proof of ownership of the public key. The mapping is confirmed to be empty before setting the signer. This raises an issue where anyone can frontrun a *createSession()* in the mempool, or any key known in advance, by sending their own *createSession()* transaction with the victim's signer.

Recommended mitigation

Two approaches can be considered:

- The session creation could provide proof of ownership, where a signature is provided that the originating account is authorized to use this SessionSpec.
- Multiple accounts using the same spec and signer could be accepted. Each account can only use the signer a single time, so one additional mapping depth is added for the **sessionSigner**. This needs to be done while considering the safety of the entire SessionLib and SessionKeyValidator surface.

Team response

Fixed.

Mitigation review

The following lines were introduced as a fix:

```
// Check address ownership to prevent DoS (since we only allow unique
signers)
address recovered = ECDSA.recover(sessionHash, proof);
require(recovered == sessionSpec.signer, SessionLib.InvalidSigner(recovered,
sessionSpec.signer));
```

The approach above is inadequate because the **proof** can be observed in the mempool and replicated together with the **sessionSpec**. The missing validation is, as described in the first point above, that the originating account (msg.sender of the *createSession()*) should be part of the signed data. Then, a frontrunning attack would not pass the signature check except on the intended account, which can be considered fine as the attacker would pay for the session creation on behalf of the honest transaction.

Team response

Fixed.

Mitigation review

The signed buffer now includes msg.sender, which fixes the issue.

TRST-M-2 Account breaks operability with any ERC7579 Validators despite intentions

- **Category:** Compliance issues
- **Source:** ModularSmartAccount.sol
- **Status:** Fixed

Description

The account is described to operate with any compliant validator. However, the first 20 bytes of the **userOp.signature** passed to the Validator module are the validator address. This goes against the specification, as the reference implementation states:

/// The MSA MUST clean up the userOp before sending it to the validator.

As all currently implemented ERC7579 validators expect their own structured authentication data to be passed from the zero-th byte of the signature, this logic essentially confines integrated validators to the set specially created for ZKsync SSO contracts.

Recommended mitigation

Two options are considered:

- Removal of the validator header of the signature before passing to the module.
- Encoding of the validator via some other field – the **nonce** is commonly used for this purpose. If spare bytes are an issue, a validator 4-byte identifier could be passed.

Team response

Fixed.

Mitigation review

Issue has been addressed by stripping the first 20 bytes of the signature field before passing to validators.

TRST-M-3 The Registry records may be stale and cause unexpected behavior

- **Category:** Logical flaws
- **Source:** RegistryAdapter.sol
- **Status:** Fixed

Description

When a user calls *setRegistry()* to update their trusted attesters, if the attester length is positive then *trustAttesters()* is invoked. The logic is insufficient in case the passed attester count is empty and the registry has not changed – the user would expect it to no longer accept any attestations, but the same list and threshold would still be in effect.

Recommended mitigation

The *trustAttesters()* call needs to be performed in any case when the Registry is not set to **address(0)**.

Team response

Fixed.

Mitigation review

Issue has been addressed as suggested.

Low severity findings

TRST-L-1 Fallback function does not ensure selector is valid, leading to execution of wrong logic

- **Category:** Validation issues
- **Source:** ModuleManager.sol
- **Status:** Fixed

Description

The *fallback()* function reads the requested user function using the logic below:

```
fallback() external payable {
    FallbackHandler storage $fallbackHandler =
$moduleManager().$fallbacks[msg.sig];
    address handler = $fallbackHandler.handler;
    bytes1 calltype = $fallbackHandler.calltype;
```

The issue is that **msg.sig** will always read the first four bytes of calldata, even if less are defined. The rest are filled as zeroes. As a result, if the contract receives calldata of **0x112233**, and has a registered fallback for **0x11223300**, the fallback would be identified and called.

Note that the finding is expected to lead to Low impact as the handler would receive the same short calldata and presumably revert, but there may scenarios with material impact, for example if the handler has their own fallback function which does not expect such input.

Recommended mitigation

When the incoming calldata length is positive and under 4, revert the execution.

Team response

Fixed.

Mitigation review

Issue has been fixed as recommended.

TRST-L-2 Relocating to a new Registry leaks trusted attesters on the previous registry

- **Category:** Logical flaws
- **Source:** RegistryAdapter.sol
- **Status:** Fixed

Description

The *setRegistry()* function is used by the user to select which ERC-7484 registry will be used to authenticate modules. Consider the scenario where a Registry with a particular attester set is active, and then the function is called to move to a different Registry and attester set. Since the function does not reset the old registry, a call on the old registry to *checkForAccount()* will respond as if the account is still active on it with the old configuration. Normal functionality will not be impacted as the new registry will be the one used for queries, however for

compliance and for integration safety with any contract that would query the old registry, it is recommended to keep all information fresh and accurate.

Recommended mitigation

Set the previous registry to a default value – an empty attester array and a non-zero **threshold**.

Team response

Fixed.

Mitigation review

Issue has been addressed as recommended. A zero threshold is acceptable as that restores to the default behavior of the registry.

TRST-L-3 Modules could deny forceful uninstallation

- **Category:** Gas-related attacks
- **Source:** ModuleManager.sol
- **Status:** Fixed

Description

In the *uninstall()* functions of the ModuleManager, the module call is wrapped in a try/catch which accepts error bytes. The issue is that in the case of forceful uninstallation, it is important to ensure that the call is successful and cannot lead to reverts. A malicious module can indeed avoid even forceful installation using a returnbomb attack. In this instance, the Modular receives **63/64** of the remaining gas at call time, and can use this gas allocation entirely to revert with an excessively large reason. Decoding the **reason** into memory would be too expensive leading to DoS of the entire uninstallation.

Recommended mitigation

A low-level call should be used to uninstall modules, passing a limiting gas amount and not reading the revert reason into memory, except in the non-forceful uninstallation path.

Team response

Fixed.

Mitigation review

After the fix, the code limits gas expenditure on the forceful path. While no issues are detected, it's noted that only half of the remaining gas is used to execution the module uninstallation call, which may cause issues with transaction simulators passing not enough gas to perform proper uninstallation when possible. It is recommended to ensure at the frontend that a large gas stipend is passed for the transaction.

TRST-L-4 Domain name length requirement is too strict

- **Category:** Compliance issues
- **Source:** WebAuthnValidator.sol

- **Status:** Fixed

Description

The WebAuthnValidator allows domain lengths of up to **253** bytes:

```
uint256 domainLength = bytes(domain).length;
require(domainLength >= 1 && domainLength <= 253, BadDomainLength());
```

However, the RFC allows names of up to **255** bytes in size.

```
2.3.4. Size limits

Various objects and parameters in the DNS have size limits.  They are
listed below.  Some could be easily changed, others are more
fundamental.

labels          63 octets or less

names           255 octets or less
```

Therefore, in case of a long domain which would appear under “origin” of the WebAuthn JSON, it would not be possible to add such credentials to the validator.

Recommended mitigation

Support up to 255 bytes.

Team response

Fixed.

Mitigation review

Fixed as recommended.

TRST-L-5 Malicious guardian could force heavy gas losses by the account

- **Category:** Gas-related issues
- **Source:** GuardianExecutor.sol
- **Status:** Fixed

Description

A Guardian can initiate recovery passing arbitrary bytes in **data**, and these are stored in the **pendingRecovery** mapping. Suppose a malicious Guardian stores an excessively large **data** in memory. The account owner would have to call *discardRecovery()* to abort it, suffering from heavy gas costs from a long SSTORE loop only bound indirectly by the block gas limit which throttled the recovery initiation.

Recommended mitigation

Instead of storing the entire data contents, it would be enough to store the hash on-chain, while emitting the original data in the initiation to guarantee data availability.

Team response

Fixed.

Mitigation review

Issue was addressed as recommended.

TRST-L-6 The WebAuthn validation function would treat the default credential mapping as valid

- **Category:** Validation issues
- **Source:** WebAuthnValidator.sol
- **Status:** Acknowledged

Description

In *webAuthVerify()*, the public key is fetched from storage as below:

```
bytes32[2] memory publicKey = publicKeys[origin][credentialId][msg.sender];
```

However, it isn't checked to ever be set. The default coordinate *(0,0)* would be sent to the P256 library.

```
bool signatureValid = P256.verifySignature(message, rs[0], rs[1],
publicKey[0], publicKey[1]);
```

The cryptographic safety of the call above is out of scope, but regardless an invalid point should never be sent to the verification library.

Recommended mitigation

In case the key is *(0,0)*, revert immediately.

Team response

Solady's *P256.verify()* returns false when verifying with *(0, 0)* key.

TRST-L-7 SessionKeyValidator does not protect from Registry calls

- **Category:** Validation issues
- **Source:** SessionKeyValidator.sol
- **Status:** Fixed

Description

The *isBannedCall()* checker prevents any catastrophic-damage calls from within the session. It lacks a check for the account's Registry as a target. A malicious session execution could reset the attesters of the account, leading to DoS of most account functions from this point and until the registry is recovered by the owner directly.

Recommended mitigation

Add a check that the **target** is not the Registry in *isBannedCall()*.

Team response

Fixed.

Mitigation review

Issue was addressed as suggested.

TRST-L-8 Sessions are overlapping at boundaries, slightly breaking allowance limits

- **Category:** Time-sensitivity issues
- **Source:** SessionLib.sol
- **Status:** Fixed

Description

When using an Allowance limit, the validity period is inclusive:

[period * limit.period, (period+1) * limit.period]

This means two adjacent periods share a timestamp, so two separate periods can be used to pass an allowance check for the same block – breaking the guarantee of a maximum amount for a given period.

Recommended mitigation

Since ERC4337 considers the range inclusive, reduce the **validUntil** by one to make the periods not overlapping.

Team response

Fixed.

Mitigation review

Fixed as recommended.

TRST-L-9 The SessionKeyValidator doesn't implement storage slot gaps, making it difficult to upgrade when derived by another contract

- **Category:** Inheritance issues
- **Source:** SessionKeyValidator.sol
- **Status:** Fixed

Description

The SessionKeyValidator is designed to be behind a proxy while also supporting derivation through virtual methods. However, it doesn't include storage slot gaps to support upgrades and fixes to the base contract after it is inherited. The GuardianExecutor in contrast does follow this best practice.

Recommended mitigation

Introduces a storage gap similar to the GuardianExecutor.

Team response

Fixed.

Mitigation review

Addressed as recommended.

TRST-L-10 Some validators allow arbitrary appending of bytes to signature, raising verification costs

- **Category:** Encoding issues
- **Source:** WebAuthnValidator.sol, SessionKeyValidator.sol
- **Status:** Acknowledged

Description

In ERC4337, all UserOperation fields are hashed and are part of the **userOpHash** passed to the *validate()* function, except the signature. Since validation functions usually ensure the **userOpHash** is signed by some method, the only field that can potentially be mutated by an attacker while the UserOperation has not been processed is the **signature** field. A possible griefing attack is possible on the WebAuthnValidator and SessionKeyValidator. These decode the signature using *abi.decode()*, yet don't bind the signature length to match just the decoded and necessary bytes. An attacker could simply view a victim UserOperation and extend the signature with trailing zeroes to charge the user for additional costs of copying calldata to memory. The attack is capped to loss of the delta between the true verification cost and the maximum accepted cost, and is contingent on bundler preferring to process additional calldata in order to profit from more verification gas, limiting its impact.

Recommended mitigation

Ensure the signature is tightly bounded – all internal dynamic arrays are of a particular length and there are no trailing bytes.

Team response

"The gas limit reduces the potential impact of this scenario, as you mention in the description, and then the user may switch bundlers, avoiding bundlers that behave this way."

TRST-L-11 Lack of ERC165 support in EOAKeyValidator

- **Category:** Compliance issues
- **Source:** EOAKeyValidator.sol
- **Status:** Fixed

Description

The SSO contracts intend to support ERC165 for interface detection. However, the EOAKeyValidator does not adhere to the specification, leading to detection issues for the validator.

Recommended mitigation

Implement the *supportsInterface()* function in EOAKeyValidator.

Team response

Fixed.

Mitigation review

Fixed as suggested.

TRST-L-12 Fallback module handling does not adhere to ERC7579 specification

- **Category:** Compliance issues
- **Source:** ModuleManager.sol
- **Status:** Acknowledged

Description

In the EIP7579 specification, a module is either installed or not, without the concept of a partial installation. By the specs language, the *installModule()* installs a module and MUST revert if the module is already installed, while *uninstallModule()* uninstalls a module and MUST revert if the module is not installed.

The issue is that the ModuleManager does not handle Fallback modules in the same way it does for Validator and Executor modules. For example, it can install a module multiple times, each one for a different selector, or uninstall them. While it is clear the intention was to design flexible fallback paths, there are ways to achieve that without breaking compatibility with the specification. Another instance when the specification is not followed is through modification of the **initData** / **deInitData** parameters. These should be passed unaltered into the Fallback to maintain compliance, as they are defined as arbitrary bytes from the eyes of the wallet.

Recommended mitigation

Refactor the codebase to handle Fallback modules similarly to the other types. For flexibility with adding and removing of selectors on the fly, a custom interface can be written that ZKSync-complying fallbacks would support.

Team response

"Our implementation behaves the same as the reference implementation. We are not convinced that the intention of the ERC-7579 standard was to forbid installing one fallback handler under multiple selectors."

TRST-L-13 The MSA initialization does not handle multiple-mode modules correctly

- **Category:** Logical flaws
- **Source:** ModularSmartAccount.sol
- **Status:** Acknowledged

Description

The ERC7579 specs state the same module can support different types, for example both Executor and Validator. It is expected that when installing a module as two different types, the input for *onInstall()* would have to be different for each, otherwise the module would not be able to detect which kind of module the account is opting to install. The *initializeAccount()* of the MSA looks over the modules to install and passes the same **data** for both installation types. Moreover, the function implicitly opts in for all modes the module supports, instead of allowing the initializer to opt for a subset of modules. It seems the multiple-mode option has not been considered.

Recommended mitigation

The function should accept a list of types for each module and separate **data** for each.

Team response

"Acknowledged. This is a known assumption (as documented in last commit) that during initializeAccount all modules passed in are installed exactly once and are of exactly one module type. Compatibility of initializeAccount with third-party modules was not our priority."

Additional recommendations

TRST-R-1 Validate fallback calltype is supported at installation

In *_installFallbackHandler()*, any **calltype** (bytes1) is accepted and stored in the handler, but the contract only supports **0xFE, 0x00** types. It would be best to check for errors at this stage.

TRST-R-2 Use a single allocation for a contiguous memory segment

In the ModuleManager, a bump allocator is implemented in *allocate()*. The logic only works if the assumption holds that the two calls to allocate the calldata and the sender address will return sequential addresses. While it is likely to be the case here, it is generally unrecommended to make such assumptions as the compiler could theoretically interject and allocate some value, creating an unexpected gap. It would be best to allocate the total amount in one call instead.

TRST-R-3 Silent returns when sanity checks fail in the ModuleManager and SessionLib are unrecommended

In case the *fallback()* of the account is called, it is expected that the handler **calltype** would have to be one of the supported calltypes. In case it isn't, a serious error has occurred during installation, and the best practice would be to revert to alert the user, instead of silently returning from the call.

A similar silent return exists in *remainingLimit()* of SessionLib, in case a flow that should never be possible occurs.

TRST-R-4 Unnecessary complexity in *supportsExecutionMode()*

The *supportsExecutionMode()* is clumsy, as it sets the **isSupported** variable multiple types, although the function can never return **false** through returning that variable. It would be best to eliminate that variable, and simply return **true** if all the checks pass.

TRST-R-5 Closely monitor standards that are still in draft stage

The ERC7595 and ERC7484 standards are still in draft phase. Since the account heavily relies on their behavior, it is important to stay updated on any breaking changes and to be able to upgrade the account implementation in case it is necessary.

TRST-R-6 Redundant check in several validation functions

In *validateUserOp()*, the EOA validator and SessionKeyValidator use *ECDSA.tryRecover()*, then checks the signer output is not zero and that there were no errors. The ECDSA library treats a zero signer as an error, so it is not necessary to check for it.

TRST-R-7 Uninstallation of several validators is error prone

In the EOAKeyValidator and the WebAuthnValidator, the *onUninstall()* receives parameters used to remove credentials or trusted EOA keys. If the validator is later installed again, any old credentials or keys would still work, making it somewhat error prone.

TRST-R-8 Ensure Solady dependency contract is deployed

The P256 library from Solady depends on the VERIFIER being deployed at a known address. Since the contracts will be deployed on a chain that's not yet launched, it is impossible to verify the contract will be deployed on that address, so a note should be made to verify it post launch.

TRST-R-9 Functions with underscore prefix should be internal

In EOAKeyValidator the *_addOwner()* function is defined as **public**. Although there are no security concerns, the function lacks validation when called directly, so it is best to set as **private** or **internal**. Moreover, functions with underscore prefix are expected to not be exposed.

TRST-R-10 The WebAuthn authentication flaw does not perform a mandated check

In the WebAuthn validator, the RP ID portion of the **authenticatorData** is never checked to be the correct domain for the credential, and instead the **origin** JSON value is used. The specification mandates that the RP ID should always be checked, since this is the value the user is able to see on the authentication device, while the JSON is not. Although there is no direct exploit when the browser operates correctly, if the browser is compromised the RP ID check represents a last line of defense, which is missing in this case.

TRST-R-11 Only emit events on state changes

In the WebAuthnValidator, the *PasskeyRemoved* event is emitted in *removeValidationKey().* However, the Passkey is only removed if *remove()* call returns true, meaning the key existed.

TRST-R-12 Tighten validation of calldata in SessionLib

The SessionLib *validate()* is critical in ensuring the safety of user calls. Some checks are missing which do not directly result in security issues, but are still highly recommended:

- The execution type should be verified to be **EXECTYPE_DEFAULT** or **EXECTYPE_TRY**, as these are the only ones supports by the MSA.
- Encapsulated calldata with 1-3 bytes should be considered invalid as it is suspicious and there are no good use cases for it.
- The offset of where the dynamic bytes data starts is not authenticated to be tightly packed. The **offset** variable should not be used, instead a hard-coded value should be required, since the *execute()* signature is known and the packed data offset can be calculated.

TRST-R-13 Removing a guardian could discard a pending recovery from a trusted guardian

In *removeGuardian()*, in case the removed guardian was active, it discards the pending recovery if it exists. The documentation says it discards the recovery if it was started by this guardian, but the current recovery could have been started by a different, trusted one. In that case, the expected behavior would be to skip the discarding action. It can be implemented by adding the initiating guardian address to the recovery struct.

TRST-R-14 Revert when shrinkRange() creates an unordered range

In *shrinkRange()*, it is possible for a range to be returned when the end is before the start value. For example, the ranges **[10,12]** and **[18, 20]** would be shrunk to **[18, 12]**. Since such a range is semantically invalid, it is recommended to revert when performing such shrinking.

TRST-R-15 Treat sessions as closed after expiration timestamp

When *getState()* is called on the SessionLib, it returns the current **session.status[account]** value for the status. In case the session **expiredAt** has already passed, it can never be used, so it could be more accurate to consider it **Closed**, rather than **Active**, at any point in the future.

TRST-R-16 Validate calls are not banned at runtime

The SessionKeyValidator ensures a created session does not allow especially dangerous call **target/selector** pairs. However, this is checked only during session construction. In case new modules were installed, the previous check is considered outdated, but a new check is never performed. It is recommended to call *isBannedCall()* within the validation flow as well.

Centralization risks

TRST-CR-1 Beacon owner is fully trusted

All accounts are deployed as BeaconProxy contracts. The admin is able to change the beacon implementation as a defensive measure, however that also means it is a completely trusted entity for the MSA.

TRST-CR-2 User can opt in to various centralized modules

The account owner may at their discretion opt-in to centralized services, while at the core they are only bound by the Beacon owner trust assumption. Some opt-in centralization paths include:

- Configured Registry attesters
- Guardians configured on the GuardiansExecutor or the GuardianBasedRecoveryExecutor

Systemic risks

TRST-SR-1 Integrations are trusted

The SSO contracts use several external dependencies. Any bugs or compromises of these could affect the safety of the MSA. Those include:

- Solady
- OpenZeppelin contracts
- ERC4337 reference implementation
- Registries
- External modules