Rhinestone Smart Sessions (2025-07) Security Audit

: Rhinestone Smart Sessions

July 30, 2025

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Beginning on June 20, 2025, ChainLight conducted a 5-day security audit of the Rhinestone Smart Contract. The audit focused on thoroughly examining the Smart Session's operational integrity and its resilience against potential bypass attempts.

Summary of Findings

The audit revealed a total of **4** issues, categorized by severity as follows:

- **Medium:** 2 issue
- **Low:** 1 issues
- **Informational:** 1 issue

Audit Overview

Scope

Name	Rhinestone Smart Sessions (2025-07) Security Audit
Target / Version	• Git Repository (`https://github.com/erc7579/smartsessions`) ◦ PR122 (https://github.com/erc7579/smartsessions/pull/122): `cd48082edf9ee49a6f29bdac27a01d9a72c436d3` ◦ PR159 (https://github.com/erc7579/smartsessions/pull/159): `6042af15eb4a6330f651fcccefaafff0c4eb91e6` ◦ PR162 (https://github.com/erc7579/smartsessions/pull/162): `643bc0c160f8cd16a8b0fecd135f49d9ff462120`
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	**0**	• N/A
High	**0**	• N/A
Medium	**2**	• `SmartSessions-001` • `SmartSessions-003`
Low	**1**	• `SmartSessions-002`
Informational	**1**	• `SmartSessions-004`
Note	**0**	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	`SmartSessions-001`	Insufficient gas control in `SimpleGasPolicy` leading to uncontrolled ETH expenditure	**Medium**	**Patched**
2	`SmartSessions-002`	Mismatch in call type support between `ValueLimitPolicy` and enforcement logic	**Low**	**Patched**
3	`SmartSessions-003`	Missing whitelist reset in `ContractWhitelistPolicy.initializeWithMultiplexer()`	**Medium**	**Patched**
4	`SmartSessions-004`	Minor Suggestions	**Informational**	**Patched**

#1 `SmartSessions-001` Insufficient gas control in `SimpleGasPolicy` leading to uncontrolled ETH expenditure

ID	Summary	Severity
`SmartSessions-001`	The `SimpleGasPolicy` only limits the gas limit and does not restrict the gas price, allowing sessions to spend more ETH than the user intended.	Medium

Description

The `SimpleGasPolicy` as currently implemented restricts only the gas limit for `userOp`. It does not account for the `maxGasPrice`, meaning that while the maximum computational steps are capped, the actual cost in ETH can still be arbitrarily high if a high gas price is set by the session. This gap allows for potentially excessive ETH expenditure despite the policy's presence.

Impact

Medium

The current `SimpleGasPolicy` does not properly enforce the gas limits set by the user, allowing sessions to pay more than intended. This undermines the effectiveness of the policy in controlling account expenditure and may lead to unexpected financial losses from the user's account.

Recommendation

It is recommended to modify it to calculate the user operation's `maxGasPrice` and multiply it by the gas limit to determine actual gas consumption, then set a corresponding limit.

Remediation

Patched

The issue has been resolved as recommended.

#2 `SmartSessions-002` Mismatch in call type support between `ValueLimitPolicy` and enforcement logic

ID	Summary	Severity
`SmartSessions-002`	The handling of `CALLTYPE_STATIC` and `CALLTYPE_DELEGATECALL` cases in `ValueLimitPolicy.checkUserOpPolicy()` is inconsistent.	Low

Description

The `ValueLimitPolicy.checkUserOpPolicy()` is designed to control value transfers. While `CALLTYPE_STATIC` operations inherently do not transfer value and should typically pass validation, the current implementation does not explicitly account for them.

Although `CALLTYPE_STATIC` may appear acceptable from a value-transfer perspective, the `SmartSession._enforcePolicies()` function currently permits only `CALLTYPE_SINGLE` and `CALLTYPE_BATCH`, excluding both `CALLTYPE_STATIC` and `CALLTYPE_DELEGATECALL` entirely. Given this inconsistency, and to ensure clear and predictable behavior, it is recommended that both CALLTYPE_STATIC and CALLTYPE_DELEGATECALL be explicitly disallowed within `ValueLimitPolicy.checkUserOpPolicy()` to align with the current enforcement logic in `SmartSession`.

Impact

Low

There is an inconsistency between `ValueLimitPolicy.checkUserOpPolicy()` and `SmartSession._enforcePolicies()` regarding supported call types.

Recommendation

It is recommended to explicitly treat both `CALLTYPE_STATIC` and `CALLTYPE_DELEGATECALL` as `UnsupportedCallType` within the `ValueLimitPolicy.checkUserOpPolicy()`.

Remediation

Patched

The issue has been resolved as recommended.

#3 `SmartSessions-003` Missing whitelist reset in `ContractWhitelistPolicy.initializeWithMultiplexer()`

ID	Summary	Severity
`SmartSessions-003`	The `ContractWhitelistPolicy.initializeWithMultiplexer()` function lacks a mechanism to clear previously stored `$targets`, potentially leading to an accumulation of outdated or unintended whitelist entries.	**Medium**

Description

The `initializeWithMultiplexer()` function within the `ContractWhitelistPolicy` is responsible for setting up the list of whitelisted contract targets. However, the current implementation does not clear previously stored targets before setting the new list. As a result, when the function is called multiple times, old contract addresses may remain in the whitelist alongside the newly provided ones. This can lead to an incorrect whitelist containing outdated or unintended contracts, which undermines the integrity of the access control logic.

Impact

Medium

The `initializeWithMultiplexer` function does not clear the previously set whitelist contracts, which may result in unintended contracts remaining in the whitelist.

Recommendation

It is recommended to implement a process within `ContractWhitelistPolicy.initializeWithMultiplexer()` to explicitly delete or clear all previously stored `$targets` before initializing the new `$targets`.

Remediation

Patched

The issue has been resolved as recommended.

#4 `SmartSessions-004` Minor Suggestions

ID	Summary	Severity
`SmartSessions-004`	The description includes multiple suggestions for preventing incorrect settings caused by operational mistakes, mitigating potential issues, and improving code maturity and readability.	Informational

Description

1. In `ERC20SpendingLimitPolicy.checkAction()` and `UsageLimitPolicy._checkUsageLimit()`, storage values are incremented before final validation, which can lead to increments even on `VALIDATION_FAILED` cases.

2. `ERC20SpendingLimitPolicy.initializeWithMultiplexer()` lacks a `require(tokens.length == limits.length);` check.

3. `msgSender` is used inconsistently across `SimpleGasPolicy`, `TimeFramePolicy`, `UniActionPolicy`, `UsageLimitPolicy`, and `ValueLimitPolicy` contract instead of `multiplexer` for mapping storage variables.

4. `initializeWithMultiplexer()` in `SimpleGasPolicy`, `UsageLimitPolicy`, and `ValueLimitPolicy` does not revert if a limit is initialized to 0.

5. `TimeFramePolicy.initializeWithMultiplexer()` does not revert if `validUntil` is not 0 and `validAfter` is greater than `validUntil`.

6. A redundant condition `config.validUntil() == 0` exists in `TimeFramePolicy.check1271SignedAction()`.

7. The comment in `ContractWhitelistPolicy.isContractWhitelisted()` refers to time frame explanations incorrectly.

8. The `childIndex` variable in `ArgPolicyTreeLib.createNotNode()` could be more descriptive as `leftChildIndex`.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

It has been patched as recommended.

Revision History

Version	Date	Description
1.0	July 30, 2025	Initial version

ChainLight