Modular Account v2 Security Audit

: Modular Account v2

Nov 28, 2024

Revision 1.1

ChainLight@Theori

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.

Table of Contents

Executive Summary

Starting on Oct 17th, 2024, ChainLight of Theori audited the Alchemy Modular Account v2 for two weeks. In the audit, we primarily considered the issues/impacts listed below.

- Theft of funds
- Bypassing the validation module
- Compliance with ERC-4337 and ERC-6900
- The risk of deferred validation installation
- DoS and privilege infringement between different validation modules
- Safety of library code optimized through assembly

As a result, we identified issues as listed below.

- Total: 10
- High: 2 (Validation bypass, privilege abuse.)
- Low: 3 (Nonce collision, Denial-of-service attack, etc.)
- Informational: 5 (Minor suggestions.)

Audit Overview

Scope

Name	Modular Account v2 Security Audit
Target / Version	• Git Repository (`alchemyplatform/modular-account/src/*`): commit `14afcd84859cf468684634eb9a9f8cb787f60b16` • Git Repository (`erc6900/reference-implementation/src/`): commit `e5f55ab1c3ac2f78efa14967e90e95c3e4ace38c` ◦ `helpers/Constants.sol` ◦ `helpers/EmptyCalldataSlice.sol` ◦ `libraries/HookConfigLib.sol` ◦ `libraries/ModuleEntityLib.sol` ◦ `libraries/SparseCalldataSegmentLib.sol` ◦ `libraries/ValidationConfigLib.sol` ◦ `modules/ModuleEIP712.sol` ◦ `modules/ReplaySafeWrapper.sol`
Application Type	• Smart contracts (Modular Account)
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	2	- `ALCHEMY-001` - `ALCHEMY-002`
Medium	0	- N/A
Low	3	- `ALCHEMY-003` - `ALCHEMY-004` - `ALCHEMY-009`
Informational	5	- `ALCHEMY-005` - `ALCHEMY-006` - `ALCHEMY-007` - `ALCHEMY-008` - `ALCHEMY-010`
Note	0	- N/A

Findings

Summary

#	ID	Title	Severity	Status
1	ALCHEMY-001	The `_validateDeferredActionAndSetNonce()` function in `ModularAccountBase` must verify whether the function selector matches `installValidation.selector`	High	Patched
2	ALCHEMY-002	`ModularAccountBase.executeUserOp()` must extract an inner validation function in the deferred validation installation scenario	High	Patched
3	ALCHEMY-003	In `ModularAccountBase._validateUserOp()`, a malicious validation module could perform a DoS attack on other validation modules	Low	Patched
4	ALCHEMY-004	Conflicts in `deferredActionNonceUsed[nonce]` between different validation modules should be prevented	Low	Patched
5	ALCHEMY-005	An attacker could intentionally bypass the deferred validation installation process or include it in another UserOperation to execute it first	Informational	Patched
6	ALCHEMY-006	Execution hooks associated with `installValidation` should not violate the storage rules specified in ERC-7562	Informational	Acknowledged

#	ID	Title	Severity	Status
7	`ALCHEMY-007`	Missing `invalidateDeferredValidationInstallNonce` Selector in `NativeFunctionDelegate.isNativeFunction()`	Informational	Patched
8	`ALCHEMY-008`	Duplicate replay protection in deferred action scenario	Informational	Patched
9	`ALCHEMY-009`	The `AllowlistModule.setAddressAllowlist()` call always disables the ERC20 spend limit functionality for the target address	Low	Patched
10	`ALCHEMY-010`	Minor Suggestions	Informational	Patched

#1 `ALCHEMY-001` The `_validateDeferredActionAndSetNonce()` function in `ModularAccountBase` must verify whether the function selector matches `installValidation.selector`

ID	Summary	Severity
`ALCHEMY-001`	In the deferred validation installation scenario of `ModularAccountBase._validateUserOp()`, only the `installValidation` function should be allowed; however, `_validateDeferredActionAndSetNonce()` does not perform this validation.	High

Description

In the `ModularAccountBase._validateDeferredActionAndSetNonce()` function, `bytes4(encodedData[63:67])` must be verified to be `installValidation.selector`. Currently, other functions (e.g., `execute()`) can also be called in the deferred scenario instead of `installValidation`.

A validation module with authority to execute signature validation and run `execute()` may be abused in these deferred scenarios. Specifically, in this case, because `_validateDeferredActionAndSetNonce()` does not perform `if (target == address(this)) { revert... }` when the selector is `execute()`, it allows direct access to native functions (e.g., `installValidation`) through `execute()`. When accessing a native function through `execute`, the `msg.sender` is the account itself, which bypasses the validation process for native functions and only triggers the execution hooks related to the native function selector. In this case, a validation module with permission for `execute()` can perform other native functions besides `execute()`. In other words, the deferred validation installation process could enable the validation module to perform actions beyond its authorized scope.

Impact

High

In the deferred validation installation scenario, functions other than `installValidation` can be called. Specifically, a validation module with permission to use `execute()` or `executeBatch()`

can access other native functions through `execute()` or `executeBatch()`. In this case, the validation process for native functions can be bypassed, allowing the validation module to perform actions beyond its permitted authority.

Recommendation

It is recommended to verify in the `_validateDeferredActionAndSetNonce()` function whether `bytes4(encodedData[63:67])` matches `installValidation.selector`.

Remediation

Patched

The `_checkIfValidationAppliesSelector()` function has been changed to `_checkIfValidationAppliesCallData()`, introducing additional validation for cases where a self-call is performed through the deferred action.

#2 `ALCHEMY-002` `ModularAccountBase.executeUserOp()` must extract an inner validation function in the deferred validation installation scenario

ID	Summary	Severity
`ALCHEMY-002`	`ModularAccountBase.executeUserOp()` extracts `userOpValidationFunction` from a fixed position (`userOp.signature[:24]`), without accounting for the deferred validation installation case.	High

Description

The `ModularAccountBase.executeUserOp()` function always extracts `userOpValidationFunction` from `userOp.signature[:24]`, without considering deferred scenarios. In a deferred scenario, `userOp.signature[:24]` corresponds to the validation module permitting `installValidation()`, while the inner validation module associated with the user operation is located elsewhere in `userOp.signature`. Thus, in a deferred scenario, the execution hooks related to the inner validation module should be triggered in `executeUserOp()`, but instead, the execution hooks of the validation module permitting `installValidation` are called.

Impact

High

In `executeUserOp()`, during the deferred validation installation scenario, the execution hook of a validation module that permits `installValidation()` is triggered instead of the inner validation. This results in the omission of the execution hook call for the inner validation.

Recommendation

In `executeUserOp()`, if the deferred action flag is true, it is recommended to extract the inner validation module from `userOp.signature[29 + 38: 29 + 63]`.

Remediation

Patched

The address of the inner validation module for `userOp` has been updated to be extracted from `userOp.nonce` instead of `userOp.signature`.

#3 `ALCHEMY-003` In `ModularAccountBase._validateUserOp()`, a malicious validation module could perform a DoS attack on other validation modules

ID	Summary	Severity
`ALCHEMY-003`	A malicious validation module could prevent the execution of a UserOp created by a normal validation module by generating a UserOp using the same nonce value and a higher gas price.	Low

Description

In `ModularAccountBase._validateUserOp()`, the `UserOp.nonce` value is not associated with the `ModuleEntity` value, allowing nonce duplication between different modules. For example, a malicious validation module could generate a `userOp` with the same nonce as another module's valid `userOp` to prevent its execution. If the gas price of this malicious `userOp` is higher, the original `userOp` will be discarded in the bundler's mempool. This way, a malicious `userOp` validation module could conduct a DoS attack to prevent other validation modules from operating correctly.

Impact

Low

A malicious validation module could exploit nonce duplication to prevent the execution of another validation module's UserOperation. In this case, the malicious validation module could perform a DoS attack until it exceeds permitted transaction counts, native token limits, or other constraints.

Recommendation

It is recommended to verify that the key portion of `userOp.nonce` matches the `validationFunction` in `_validateUserOp()`.

Remediation

Patched

The validation module has been updated to be selected based on `userOp.nonce` instead of `userOp.signature`, allowing each validation function to have a unique nonce space.

#4 `ALCHEMY-004` Conflicts in `deferredActionNonceUsed[nonce]` between different validation modules should be prevented

ID	Summary	Severity
`ALCHEMY-004`	If there are multiple validation modules with permission to call `installValidation`, a malicious validation module could use the `deferredActionNonceUsed[nonce]` value first, thereby blocking the deferred actions and UserOperation executions of other validation modules.	Low

Description

Similar to issue [ALCHEMY-003], there may be conflicts in the `deferredActionNonceUsed[nonce]` value between different validation modules with permission to call `installValidation`. A malicious validation module with `installValidation()` permissions could exploit `deferredActionNonceUsed[nonce]` conflicts to cause other UserOperations containing deferred actions to fail during the validation stage.

Impact

Low

A malicious validation module with permission to call `installValidation()` could exploit conflicts in the `deferredActionNonceUsed[nonce]` value to cause other UserOperations containing deferred actions to fail.

Recommendation

It is recommended to verify that `deferredActionNonceUsed[nonce]` matches `userOp.nonce` (assuming the recommendations from issue [ALCHEMY-003] have been applied).

Remediation

Patched

The issue has been resolved as recommended.

#5 `ALCHEMY-005` An attacker could intentionally bypass the deferred validation installation process or include it in another UserOperation to execute it first

ID	Summary	Severity
`ALCHEMY-005`	Data related to deferred actions is included in `userOp.signature`, but an attacker could remove this deferred action data from the signature or add it to another `userOp`.	Informational

Description

A malicious bundler or an attacker (i.e. any user) could intentionally bypass the deferred validation installation process in `ModularAccountBase._validateUserOp()`. In the existing `userOp.signature`, the `uoValidation` module value (`userOp.signature[29 + 38 : 29 + 63]`) and `userOpSignature` (`userOp.signature[33 + encodingDataLength + deferredActionSigLength:]`) can be extracted. Subsequently, by placing `uoValidation` at `signature[0:24]` and `userOpSignature` at `signature[25:]`, and setting the `deferred action` flag of `validationFlag` to `false`, the `userOp.signature` can be restructured. In this case, the deferred action of the UserOperation is bypassed. This would enable the malicious bundler to earn additional profit as the `userOp` consumes less gas for verification. Such deferred action bypassing can lead to unexpected results during the execution phase of the userOp.

Additionally, if there is a `userOp A` that contains deferred validation installation data and another `userOp B` using the same `uoValidation`, a malicious bundler or attacker could front-run the deferred validation installation by adding deferred action-related data extracted from `A`'s `userOp.signature` to `B`'s `userOp.signature`. If `userOp B` executes first, `userOp A` would later fail in the deferred validation installation process with a `PreValidationHookDuplicate` error, causing validation failure. Deferred validation installations should be configured to only execute in the original `userOp`.

Impact

Informational

A malicious bundler may be incentivized to remove deferred actions from a `userOperation` to gain higher rewards, motivating them to execute the `userOperation` without the deferred action. If the new validation module intended for installation by the deferred action does not affect the validation process of the `userOp`, the `userOp` will execute normally. However, if the validation module meant to be installed is absent during execution phase of the `userOp`, unexpected results may occur for the user.

If a malicious attacker attaches the deferred action from the original `userOp A` to another `userOp B` and executes it first (assuming `userOp B` has a higher gas price or a lower nonce), then the subsequent execution of `userOp A` will fail. In this case, the user must remove the deferred action from `userOp A` and resubmit it as a new `userOp`.

Recommendation

To prevent the deferred action from being bypassed in `_validateUserOp()`, it is recommended to add logic that prevents tampering with the deferred action flag. Scenarios where the deferred action could be front-run by another `userOp` using the same `uoValidation` can be mitigated by applying the recommendations from issues [ALCHEMY-003] and [ALCHEMY-004].

Remediation

Patched

The deferred action execution flag has been updated to be included in the `userOp.nonce` value. Additionally, the validation module has been modified to be associated with the `userOp.nonce` value, and the `userOp.nonce` value is now included in the typedHash of the deferred action. These changes ensure that attackers cannot arbitrarily skip or front-run the execution of deferred actions.

#6 `ALCHEMY-006` Execution hooks associated with `installValidation` should not violate the storage rules specified in ERC-7562

ID	Summary	Severity
`ALCHEMY-006`	The deferred action in `ModularAccountBase._validateUserOp()` calls the execution hooks associated with `installValidation.selector`. These execution hooks should not violate the storage rules specified in ERC-7562.	Informational

Description

In the deferred validation installation scenario of the `ModularAccountBase._validateUserOp()` function, the `wrapNativeFunction()` modifier triggers execution hooks related to `installValidation()`. If these execution hooks violate the storage rules of ERC-7562, the UserOperation could be discarded by the bundler. Therefore, the execution hooks associated with `installValidation` should only read and write to storage slots that are associated with the account, or perform read-only operations on other storage slots.

Impact

Informational

Recommendation

It is recommended to explicitly document that execution hooks associated with `installValidation()` must adhere to the ERC-7562 storage access rules.

References

https://eips.ethereum.org/EIPS/eip-7562

Remediation

Acknowledged

The team is aware of the issue and has documented it explicitly in the documentation.

#7 `ALCHEMY-007` Missing `invalidateDeferredValidationInstallNonce` Selector in `NativeFunctionDelegate.isNativeFunction()`

ID	Summary	Severity
`ALCHEMY-007`	`ModularAccountView.getExecutionData()` must return the module address value as the account's address for all native functions. However, `invalidateDeferredValidationInstallNonce.selector` returns `false` in `NativeFunctionDelegate.isNativeFunction()`, thus violating the standard.	Informational

Description

According to the ERC-6900 specification, the `getExecutionData()` function of `IModularAccountView` must return the account's address as the module address when the selector is a native function. However, because `NativeFunctionDelegate.isNativeFunction()` returns `false` for `ModularAccountBase.invalidateDeferredValidationInstallNonce.selector`, `ModularAccountView.getExecutionData()` returns an incorrect address instead of the account's address as the module address for `invalidateDeferredValidationInstallNonce`. This violates the ERC-6900 specification.

Additionally, ERC-6900 specifies that for native functions, the `address module` in the `ExecutionDataView` structure must always be `address(0)` (https://github.com/erc6900/reference-implementation/blob/v0.8.0-rc.5/standard/ERCs/erc-6900.md?plain=1#L228). This conflicts with the comment in `IModularAccountView.getExecutionData()`, which states that for native functions, the address should be the account's address. It is recommended to eliminate this inconsistency.

Impact

Informational

Recommendation

It is recommended to modify `NativeFunctionDelegate.isNativeFunction()` to return `true` when the selector is `ModularAccountBase.invalidateDeferredValidationInstallNonce.selector`.

References

https://github.com/erc6900/reference-implementation/blob/v0.8.0-rc.5/standard/ERCs/erc-6900.md?plain=1#L258

Remediation

Patched

The issue has been resolved as recommended.

#8 `ALCHEMY-008` Duplicate replay protection in deferred action scenario

ID	Summary	Severity
`ALCHEMY-008`	In the deferred action scenario, hash calculations that include the `domainSeparator` occur redundantly.	Informational

Description

In the deferred action scenario, `_validateDeferredActionAndSetNonce()` calculates the `typedDataHash` value with the `domainSeparator` included. However, both `SemiModularAccountBase._exec1271Validation()` and `SingleSignerValidationModule.validateSignature()` also call `replaySafeHash()`, adding the `domainSeparator` again to a hash that already includes it, resulting in redundant hashing with the `domainSeparator`. By calculating only the `structHash` without the `domainSeparator` in `_computeDeferredValidationInstallTypedDataHash()`, this unnecessary duplication can be eliminated.

Impact

Informational

Recommendation

It is recommended to modify the `_computeDeferredValidationInstallTypedDataHash()` to calculate only the `structHash` without computing the entire `typedDataHash`. Instead, the `_replaySafeHash()` for the `structHash` should be performed within the signature validation module or `SemiModularAccountBase._exec1271Validation()`.

Remediation

Patched

The duplicate replay protection has been removed only for the `SemiModularAccount`.

#9 `ALCHEMY-009` The `AllowlistModule.setAddressAllowlist()` call always disables the ERC20 spend limit functionality for the target address

ID	Summary	Severity
`ALCHEMY-009`	The `setAddressAllowlist()` function always updates the settings for the target address with `hasERC20SpendLimit` set to false. As a result, when `setAddressAllowlist()` is called independently, the ERC20 spend limit functionality may be unintentionally disabled.	Low

Description

The `AllowlistModule.setAddressAllowlist()` function creates an `AddressAllowlistEntry` structure with `hasERC20SpendLimit` always set to false, as it does not take `hasERC20SpendLimit` as a separate parameter for configuration. Consequently, if `setAddressAllowlist()` is called directly on the account rather than within `updateAllowlist()`, the ERC20 spend limit functionality for the target token may be unintentionally disabled.

Impact

Low

When `setAddressAllowlist()` is called independently, it always disables the ERC20 spend limit functionality for the target token.

Recommendation

It is recommended to either add a parameter to `setAddressAllowlist()` to set the `hasERC20SpendLimit` value or to set this function as `internal` to prevent it from being called independently.

Remediation

Patched

The issue has been resolved as recommended.

#10 `ALCHEMY-010` Minor Suggestions

ID	Summary	Severity
`ALCHEMY-010`	The description includes multiple suggestions for preventing incorrect settings caused by operational mistakes, mitigating potential issues, improving code maturity and readability, and other minor issues.	Informational

Description

Operational Risk Mitigation / Sanity Check

- In `ModuleManagerInternals._installValidation()`, when `hookConfig.isValidationHook()` is true, it must be verified that both `hasPre` and `hasPost` flags are set to false.
- In `ExecutionInstallDelegate.uninstallExecution()`, there should be a check to ensure that the `module` address is not the zero address.

Code Maturity

- In the comments of `ExecutionLib.doPreHooks()`, change “post hood entity Id” to replace `hood` with `hook`.
- The `performCreate2` function, which is not used in the current codebase, is noted in the comments of the `NativeTokenLimitModule` contract and `ModularAccountBase.wrapNativeFunction()` modifier. It is recommended to remove it.
- It is recommended to change the call from `ExecutionLib.invokePreExecHook()` to `invokePreExecHook()` in `ExecutionLib.doPreHooks()`.

Missing / Confusing Events

- In `AllowlistModule.updateLimits()`, an event should be emitted for state changes. While the `addressAllowlist` state is handled with event processing, the `erc20SpendLimits` state is not. Therefore, it is recommended to emit an event for `erc20SpendLimits` or emit an event based on `updateLimits`.

- In `TimeRangeModule.setTimeRange()`, it is recommended to add event emission for state changes.
- It is recommended to add event for state changes in the `updateLimits()` and `updateSpecialPaymaster()` functions of the `NativeTokenLimitModule`.
- The `InitializerDisabled` event in the `SemiModularAccountBase` contract is unused. It is recommended to remove it.
- The `NotInitializing` event in the `AccountStorageInitializable` contract is unused and should be removed.

Other

- In `ModuleManagerInternals._uninstallValidation()`, comments regarding hook ordering are included, but it is difficult for the account sender to verify this. It is thus recommended to explicitly announce hook ordering compliance for `execHooks` and `validationHooks` at the SDK level or in the documentation.
- In `ModuleManagerInternals._uninstallValidation()`, since `onUninstallSuccess` is a boolean variable, it is challenging to determine precisely which hook's `uninstall()` failed. It is recommended to use bitmaps to accurately indicate which module's installation failed.
- In `AccountStorage.toSetValue()`, since `SetValue` is `bytes31`, it is recommended to convert it to `bytes31` rather than `bytes30` before performing `wrap()`.
- In `SemiModularAccountBytecode._retrieveFallbackSignerUnchecked()`, when using `abi.encodePacked(signerAddress)` as the args for `createDeterministicClone()`, it parses correctly. However, if `abi.encode(signerAddress)` is used, the address value does not parse accurately. It is necessary to clarify the description of args.
- The `uint8` type of the `AccountStorage.initialized` field in `AccountStorage` undergoes implicit type conversion to `uint64` in the `AccountStorageInitializable.initializer()` function. Although this process does not lead to data loss or errors, it is recommended to change the type of the `initialized` field to `uint64` for type consistency.
- In `ExecutionInstallDelegate._setExecutionFunction()`, the argument `selector` is unnecessarily cast again to `bytes4` as in `executionStorage[bytes4(selector)];`. It is recommended that this redundant casting be removed.
- Contrary to the content in the documentation (Notion) for the Semi-Modular Account, the `SemiModularAccountBase` contract does not implement `setFallbackSignerData()`.

Therefore, if you plan to use a corresponding `updateFallbackSignerData()` function, it is recommended to update the documentation accordingly.

- To prevent the transmission of garbage data in `ExecutionLib`, initializing the last word to 0 is unnecessary when the data length is a multiple of 32. For instance, in `invokeRuntimeCallBufferPreValidationHook()`, if `authorizationSegment.length` is 32, executing `mstore(authorizationAbsOffset + 0x40, 0)` is redundant, as the buffer's last word is located at `authorizationAbsOffset + 0x20`. It is therefore recommended to adjust the code to perform `mstore(add(authorizationAbsOffset, add(roundedDownAuthorizationLength, 0x20)), 0)` only when `authorizationSegment.length != roundedDownAuthorizationLength`.
- It is recommended to update the `_INSTALL_VALIDATION_TYPEHASH` value in the comments of `ModularAccountBase._computeDeferredValidationInstallTypedDataHash` to `_DEFERRED_ACTION_TYPEHASH`.
- The `SignerTransferred()` event in `SingleSignerValidationModule` uses three indexed topics but is declared as an anonymous event. It is recommended to remove the `anonymous` keyword unless there is a specific reason for it to remain.
- In `ExecutionInstallDelegate._removeExecHooks()`, the function does not trigger a revert even if `tryRemove()` returns false. If this behavior is unintended, it is recommended to modify the function to perform a revert when `tryRemove()` returns false.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

Most suggestions are applied as recommended.

Revision History

Version	Date	Description
1.0	Nov 5, 2024	Initial version
1.1	Nov 28, 2024	Revised remediation status

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.

ChainLight