ChainLight

ZeroDev Kernel v3 Security Audit

: ZeroDev Kernel v3 (ERC-7579)

Apr 5, 2024

Revision 1.0

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.

© 2024 ChainLight, Theori. All rights reserved

Table of Contents

© 2024 ChainLight, Theori. All rights reserved

© 2024 ChainLight, Theori. All rights reserved

Executive Summary

Starting on March 11th, 2024, ChainLight of Theori audited the smart contract of ZeroDev Kernel v3 for two weeks. In the audit, we primarily considered the issues/impacts listed below.

- Theft of user funds by unprivileged attacker
- Improper access control or a logic error leading to a similar impact in validation modules
- Discrepancies with ERC-7579
- Modules having irrevocable access to kernel

As a result, we identified issues as listed below.

- Total: 12
- High: 2 (Veto logic error of vote-based validation module, etc.)
- Medium: 2
- Low: 3
- Informational: 5

© 2024 ChainLight, Theori. All rights reserved

Audit Overview

Scope

Name	ZeroDev Kernel v3 Security Audit
Target / Version	• Git Repository (`zerodevapp/kernel_v3`): commit `5f086737ad97495d04dcecf8192a3dce391a89ce` , `8646f2c25f2727019d91cbfb43f9c5688acc6fc2`
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

© 2024 ChainLight, Theori. All rights reserved

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.

© 2024 ChainLight, Theori. All rights reserved

Status Categories

Status	Description
Confirm	ChainLight reported the issue to the vendor, and they confirm that they received.
Reported	ChainLight reported the issue to the vendor.
Patched	The vendor resolved the issue.
Acknowledged	The vendor acknowledged the potential risk, but they will resolve it later.
WIP	The vendor is working on the patch.
Won't Fix	The vendor acknowledged the potential risk, but they decided to accept the risk.

© 2024 ChainLight, Theori. All rights reserved

Finding Breakdown by Severity

Category	Count	Findings
Critical	**0**	• N/A
High	**2**	• `ZERODEV-001` • `ZERODEV-010`
Medium	**2**	• `ZERODEV-004` • `ZERODEV-008`
Low	**3**	• `ZERODEV-002` • `ZERODEV-006` • `ZERODEV-007`
Informational	**5**	• `ZERODEV-003` • `ZERODEV-005` • `ZERODEV-009` • `ZERODEV-011` • `ZERODEV-012`
Note	**0**	• N/A

© 2024 ChainLight, Theori. All rights reserved

Findings

Summary

#	ID	Title	Severity	Status
1	ZERODEV-001	`ExecLib._tryExecute()/_executeDelegatecall()` return an inverted success value	High	Patched
2	ZERODEV-002	Validator installation always fails due to the nonce increment in `Kernel.installModule()`	Low	Patched
3	ZERODEV-003	Nonce increment step size should be limited in `ValidationManager._invalidateNonce()`	Informational	Patched
4	ZERODEV-004	`Kernel` should handle collisions of 4-bytes `PermissionId`, and the duplicate `selector`	Medium	Patched
5	ZERODEV-005	`Kernel` should be able to deregister modules	Informational	Patched
6	ZERODEV-006	`VALIDATION_TYPE_PERMISSION` modules cannot be installed after initialization	Low	Patched
7	ZERODEV-007	`isValidSignatureWithSender()`'s behavior is inconsistent between ECDSA validators	Low	Patched
8	ZERODEV-008	`Kernel.isValidSignature()` must check the validity of the signature's validator module address	Medium	Patched

© 2024 ChainLight, Theori. All rights reserved

#	ID	Title	Severity	Status
9	ZERODEV-009	Functions in `ExecLib` should return whether the call has been succeeded	Informational	Acknowledged
10	ZERODEV-010	Logic error of `WeightedECDSAValidator` may allow execution of rejected proposals	High	Patched
11	ZERODEV-011	Gas Optimization	Informational	WIP
12	ZERODEV-012	Minor suggestions	Informational	WIP

© 2024 ChainLight, Theori. All rights reserved

#1 ZERODEV-001

`ExecLib._tryExecute()/_executeDelegatecall()` return an inverted success value

ID	Summary	Severity
ZERODEV-001	In the `ExecLib` contract, `_tryExecute()` / `_executeDelegatecall()` returns an inverted success value from `call` / `delegatecall`.	High

Description

`ExecLib._tryExecute()` assigns a `success` value through `success := iszero(call(gas(), target, value, result, callData.length, codesize(), 0x00))`. Since `call()` returns `true` on success, `false` on failure, and the return value is flipped by `iszero()`, `success` will be `true` on failure and `false` on success. Namely, it returns a value opposite to the success of the call. The same issue exists in `ExecLib._executeDelegatecall()`.

Impact

High

When `_executeDelegatecall()` or `_tryExecute()` is used, incorrect results will be reported, which may lead to severe side effects. For instance, `executeUserOp()` performs `_doPostHook(hook, context, success, ret);`.

Recommendation

Incorrect usage of `iszero()` should be removed from `ExecLib._tryExecute()` and `ExecLib._executeDelegatecall()`.

Remediation

Patched

© 2024 ChainLight, Theori. All rights reserved

It it patched as recommended.

© 2024 ChainLight, Theori. All rights reserved

#2 ZERODEV-002 Validator installation always fails due to the nonce increment in Kernel.installModule()

ID	Summary	Severity
ZERODEV-002	Validator installation always fails due to the nonce increment in Kernel.installModule() and a nonce check afterward in _installValidation().	Low

Description

Kernel.installModule() sets the config to ValidationConfig({nonce: vs.currentNonce++, hook: IHook(address(bytes20(initData[0:20])))}); when moduleType is 1 (validator). Since the nonce value set in the config and the current vs.currentNonce value will become different due to vs.currentNonce++; it will always lead to a revert in the if(state.currentNonce != config.nonce || ...) revert InvalidNonce() statement in _installValidation().

Impact

Low

Validator modules cannot be installed.

Recommendation

Modify installModule() to increment the vs.currentNonce value after calling _installValidation().

Remediation

Patched

© 2024 ChainLight, Theori. All rights reserved

It is patched as recommended.

© 2024 ChainLight, Theori. All rights reserved

#3 ZERODEV-003 Nonce increment step size should be limited in `ValidationManager._invalidateNonce()`

ID	Summary	Severity
ZERODEV-003	If `ValidationManager._invalidateNonce()` is called with `type(uint32).max`, validators except `rootValidator` cannot be used, and new validator modules cannot be installed anymore.	Informational

Description

If `PermissionManager._invalidateNonce()` is called with the parameter `type(uint32).max`, `validNonceFrom` and `currentNonce` will be set to the maximum value of the `uint32` type. In this instance, overflow occurs during the nonce value increment when installing a new validator module in `installModule()`. Furthermore, in `validateUserOp()`, if the `vType` is not `VALIDATION_TYPE_SUDO`; it checks whether the validator's nonce value is lower than `validNonceFrom`. Therefore, using other validators except `rootValidator` in `validateUserOp()` becomes impossible.

Impact

Informational

`PermissionManager._invalidateNonce()` is a privileged function. However, the possibility of breaking the validation module usage due to an operational error or abuse of a session with limited permission should be limited.

Recommendation

Limit the nonce increment step size in `_invalidateNonce()` to at most `type(uint32).max / 2`. So, at least two accidents are required to trigger the issue. If denial of service attack is also a concern, step size should be limited to one or some reasonably low value.

Remediation

Patched

© 2024 ChainLight, Theori. All rights reserved

It is patched as recommended. (Step size limited to 10.)

© 2024 ChainLight, Theori. All rights reserved

#4 `ZERODEV-004` `Kernel` should handle collisions of 4-bytes `PermissionId`, and the duplicate `selector`

ID	Summary	Severity
`ZERODEV-004`	`PermissionManager._installPermission()` should handle the case where a `PermssionId` with the same 4 bytes already exists when adding a new `Permission` module. `SelectorManager._installSelector()` needs similar mitigation for the duplicate `selector`.	Medium

Description

When a new validation module of permission type is installed, `PermssionManager._installValidation()` calls `_installPermission()` using the first 4 bytes of the corresponding module address as `PermissionId`. A collision occurs if a permission module with a different address but the same first 4 bytes in the address was already registered.

In this case, new policies are added to the configuration for the existing module, and `signer` and `passFlag` are overwritten. Therefore, if an old permission module with ID collision is used in `validateUserOp`, it will malfunction due to the policies not intended for it applied and the signature verified through the overwritten `signer`. Similarly, since the newly added permission module uses the policy from the pre-existing module, it may also malfunction.

Additionally, due to the `PermssionManager._uninstallValidation()` not deleting the policy for modules of permission type; stale policies will be used when a new module with the same `PermssionId` as the uninstalled module is added.

Besides the above, `SelectorManager._installSelector()` has a problem different functions having the same selector cannot be differentiated.

Impact

Medium

- Both validation modules may become unusable when a `PermissionId` collision occurs.
- A previously installed handler for a selector can be replaced unintended.

© 2024 ChainLight, Theori. All rights reserved

Recommendation

Apply three items below:

1. Revert in `PermissionManager._installPermission()` if a validation module for the `PermissionId` already exists.
2. Delete all existing policies before adding new in `PermissionManager._installPermission()`, or clear the policy if the `ValidationType` is `VALIDATION_TYPE_PERMISSION` in `PermssionManager._uninstallValidation()`.
3. Revert in `SelectorManager._installSelector()` if the same selector already exists. To allow the selector to be remapped to a different target, add a parameter to explicitly allow overwriting or leave an instruction to call install after uninstall.

Remediation

Patched

`policyData` is cleared in case of a duplicate `PermissionId`. Handling of a duplicate `selector` is kept as is.

© 2024 ChainLight, Theori. All rights reserved

#5 ZERODEV-005 Kernel should be able to deregister modules

ID	Summary	Severity
ZERODEV-005	Kernel does not have a feature to deregister a module once installed.	Informational

Description

`Kernel.uninstallModule()` calls `IMoudle(module).onUninstall()` to uninstall a module. However, even if the callback is implemented well, it will only delete `Kernel` related data stored in the module and will not affect the module configuration in `Kernel`. Thus, once installed, a module is permanently attached to `Kernel`, although the actual usage of some types of modules can be disabled by a method specific to that type.

Impact

Informational

A universal method to uninstall/disable a module is crucial to effectively mitigating the risk of a vulnerable/compromised module.

Recommendation

All modules except `rootValidator` should be blockable by their address, and once blocked, they must be filtered in all `Kernel` functions.

Remediation

Patched

The uninstall feature has been revised to correctly remove related entries from `Kernel`, and a safe call wrapper prevents modules from reverting the uninstall transaction.

© 2024 ChainLight, Theori. All rights reserved

#6 ZERODEV-006 VALIDATION_TYPE_PERMISSION modules cannot be installed after initialization

ID	Summary	Severity
ZERODEV-006	`Kernel.installModule()` cannot install a validation module of the permission type since `ValidatorLib.validatorToIdentifier()` does not support `VALIDATION_TYPE_PERMISSION`.	Low

Description

When installing a validation module (`moduleType` 1), `Kernel.installModule()` uses `ValidatorLib.validatorToIdentifier()` to create `ValidationId` to be passed to `ValidationManager._installValidation()`. Since `validatorToIdentifier()` always returns `ValidationId` with `ValidationType` of `VALIDATION_TYPE_VALIDATOR`, `VALIDATION_TYPE_PERMISSION` can not be represented. Therefore, `ValidationManager._installPermission()` is not reachable, and validation modules of permission type are not installable. (Only installable through `Kernel.initialize()`.)

Impact

Low

It is a functionality issue without security implications but affects common use cases.

Recommendation

Support encoding of different `ValidationType` such as `VALIDATION_TYPE_PERMISSION` in `ValidatorLib.validatorToIdentifier` so that the permission module can be properly installed through `Kernel.installModule()`.

Remediation

Patched

© 2024 ChainLight, Theori. All rights reserved

A function that directly calls `_installValidation()` is introduced.

© 2024 ChainLight, Theori. All rights reserved

#7 ZERODEV-007 isValidSignatureWithSender()'s behavior is inconsistent between ECDSA validators

ID	Summary	Severity
ZERODEV-007	`isValidSignatureWithSender()`'s behavior in determining the expected signers of signatures is inconsistent between `ECDSAValidator` and `WeightedECDSAValidator`.	Low

Description

`ECDSAValidator.isValidSignatureWithSender()` checks if the signature is from the `msg.sender`'s `owner`, but the `WeightedECDSAValidator.isValidSignatureWithSender()` checks if the signatures are from the `sender` address's `guardians`. One will not function correctly since their behavior is inconsistent while the interface is the same.

Impact

Low

`Kernel.isValidSignature()`'s return value may be incorrect when ECDSA validators are used. However, the more critical usage in `ValidationManager` is not affected since they pass `Kernel`'s address as `sender`.

Recommendation

Their behavior should be matched to the intended one.

Remediation

Patched

© 2024 ChainLight, Theori. All rights reserved

`WeightedECDSAValidator.isValidSignatureWithSender()` has been changed to use `msg.sender` instead of `sender`.

© 2024 ChainLight, Theori. All rights reserved

#8 ZERODEV-008 Kernel.isValidSignature() must check the validity of the signature's validator module address

ID	Summary	Severity
ZERODEV-008	In `Kernel.isValidSignature()`, when `vType == VALIDATION_TYPE_VALIDATOR`, the kernel uses `isValidSignatureWithSender()` of the validator module without verifying its address.	Medium

Description

`Kernel.isValidSignature()` calls `ValidatorLib.decodeSignature(signature)` to decode the `vId` (i.e., `vType`) and the raw signature. If `vType` is `VALIDATION_TYPE_VALIDATOR`, the next 20 bytes of the `vId` are the address of a validator module. However, there is no check whether the validator module is installed in the kernel. Thus, an attacker can set their malicious contract that returns `ERC1271_MAGICVALUE` (i.e., a signature is verified). When `isValidSignature()` is called with attacker-crafted signature data, it can always report that the signature is valid. Since this may have unexpected side effects, validating if the user-provided validator module is installed in the kernel is required.

```
// https://github.com/zerodevapp/kernel_v3/blob/19ba63ceb4364833d3d81438ffd1f1b898797413/src/Kernel.sol#L287C9-L290C17
if (ValidatorLib.getType(vId) == VALIDATION_TYPE_VALIDATOR) {
    IValidator validator = ValidatorLib.getValidator(vId);
    return validator.isValidSignatureWithSender(msg.sender, _toWrappedHash(hash), sig);
} else {
```

Impact

Medium

The impact varies depending on the implementation of the caller of `isValidSignature()` and the degree of control the attacker has on the parameter. However, the consequences of recognizing an arbitrary signature as valid may be critical.

© 2024 ChainLight, Theori. All rights reserved

Recommendation

In `Kernel.isValidSignature()`, if `vType` is `VALIDATION_TYPE_VALIDATOR`, it should check whether the user-provided validator module is installed in the kernel by checking the result of `this.isModuleInstalled(vType, validator, additionalContext)`.

Remediation

Patched

It is patched as recommended.

© 2024 ChainLight, Theori. All rights reserved

#9 ZERODEV-009 Functions in ExecLib should return whether the call has been succeeded

ID	Summary	Severity
ZERODEV-009	`Kernel.executeFromExecutor()` cannot report to `_doPostHook()` whether the call was successful since `ExecLib._execute` and other functions in `ExecLib` do not report it.	Informational

Description

In `ExecLib._execute(ExecMode execMode, bytes calldata executionCalldata)`, if `execType` is `EXECTYPE_TRY`, `ExecLib._tryExecute()` is called and only the `TryExecuteUnsuccessful` event is emitted without returning the outcome of the call. And then, `true` is passed to `_doPostHook()`, regardless of the call's outcome. (`_doPostHook(hook, context, true, abi.encode(returnData))`)

Impact

Informational

It has no impact as of now since `HookManager._doPostHook()` ignores the `success` parameter. However, it may lead to side effects in the next versions with the change of `_doPostHook()`.

Recommendation

- Functions in `ExecLib` should return the call's outcome whenever possible.
- `Kernel.executeFromExecutor()` should pass the call's outcome to `_doPostHook()`.
- `_doPostHook()` should pass the `success` parameter to the `hook.postCheck()`.

Remediation

Acknowledged

© 2024 ChainLight, Theori. All rights reserved

Modification of `ExecLib` is deferred since the hook interface will not change soon.

© 2024 ChainLight, Theori. All rights reserved

#10 `ZERODEV-010` Logic error of `WeightedECDSAValidator` may allow execution of rejected proposals

ID	Summary	Severity
`ZERODEV-010`	A rejected proposal that previously received enough approval votes can still be executed due to a logic error in `WeightedECDSAValidator`.	High

Description

When a proposal is vetoed, its state is transitioned to `Rejected`; however, if the proposal had enough votes to be approved before the veto, `getApproval()` would still return `true` for `passed`, which means the proposal received enough votes to be approved. Since `validateUserOp()` does not have a special case for the `Rejected` state and allows the execution of the proposal if `passed` from `getApproval()` is `true`, a veto has no impact on an already approved proposal.

Impact

High

Although the attacker must compromise a sufficient number of guardians for this issue to be practical, it is particularly dangerous because it gives a false sense of security. (e.g., the attack has been stopped because the malicious proposal has been vetoed.)

Recommendation

For `Rejected` proposals, `WeightedECDSAValidator.getApproval()` should return `(0, false)`.

Remediation

Patched

© 2024 ChainLight, Theori. All rights reserved

`passed` will be false for `Rejected` proposals. However, `approvals` is kept as is.

© 2024 ChainLight, Theori. All rights reserved

#11 ZERODEV-011 Gas Optimization

ID	Summary	Severity
ZERODEV-011	Some suggestions for the general gas optimization.	Informational

Description

Instead of directly referencing the array length in the condition expression of the for-loop statement, caching it to a variable might save some gas. Note that the array length must not change in the for-loop statement. (If the expected number of elements in the input array is less than 3, it is more expensive.)

- `_guardians.length` in `WeightedECDSAValidator.onInstall()`
- `_guardians.length` in `WeightedECDSAValidator.renew()`
- `validators.length` in `PermissionManager._installValidations()`
- `policyData.length` in `PermissionManager._checkUserOpPolicy()`

Impact

Informational

Recommendation

Consider applying the suggestions in the description section above.

Remediation

WIP

The ZeroDev team is working on the issue.

© 2024 ChainLight, Theori. All rights reserved

#12 ZERODEV-012 Minor suggestions

ID	Summary	Severity
ZERODEV-012	The description includes multiple minor issues, and suggestions for preventing incorrect settings caused by operational mistakes, mitigating potential issues, improving code maturity and readability.	Informational

Description

Code Readability / Typo

1. In `Kernel` and `ValidationManager`, `0xffffffff` and `0x1626ba7e` can be replaced by constant variables `ERC1271_INVALID` and `ERC1271_MAGICVALUE`.
2. `WeightedECDSAValidator` should be refactored since it has many duplicate code fragments. (e.g., loop in `onInstall()` and `renew()`, kernel/proposal status checks in `approve()` and `approveWithSig()`, and so forth.)
3. `hook.onInstall` in `ValidationManager._uninstallValidation()`'s comment should be `hook.onUninstall`.

Other

1. `Kernel.executeFromExecutor()` provides `msg.data` to the hook contract through `_doPreHook()`, but unnecessary data, such as function selector and trailing dummy data (if it exists), is also sent. Only `execMode` and `executionCalldata` should be packed and provided using `abi.encode()`.
2. `ECDSAValidator.onInstall()` should revert if `owner == address(0)`.
3. `Kernel.supportsModule()` should return `true` only for supported types 1, 2, 3, and 6.
4. `Kernel.supportsExecutionMode()` should return `true` only for `ExecMode` from supported `CallType` and `ExecType` combinations.
5. `WeightedECDSAValidator._isInitialized()` should check the `threshold` rather than `totalWeight` to be consistent with other functions.
6. Like the `ExecutorManager.executorConfig` view function, external view functions for `_fallBackConfig()` and `_selectorConfig()` should be added to `SelectorManager`.
7. `Kernel` must implement ERC-165 according to the ERC-7579.
8. According to the ERC-7579 specification, `uninstallModule()` and `installModule()` of `Kernel` should emit an event to indicate the activated module type and address.

© 2024 ChainLight, Theori. All rights reserved

9. `Kernel` leaves hook information in storage (`executionHook`), increasing gas costs. Therefore, it should be deleted at the end of `executionUserOp()`. (Transient storage can be used for EIP-1153 chains.)
10. In `ValidationManager._installPermission()`, the local variable `bytes32 aa` should be deleted as it is unused.
11. The module uninstall feature is incomplete across the codebase. An uninstall feature that properly cleans relevant states should be implemented. However, fixing the [ZERODEV-005] issue can mitigate the security implications of not having an uninstall feature.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

WIP

The ZeroDev team is working on the issue.

© 2024 ChainLight, Theori. All rights reserved

Revision History

Version	Date	Description
1.0	Apr 5, 2024	Initial version

© 2024 ChainLight, Theori. All rights reserved

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.

© 2024 ChainLight, Theori. All rights reserved