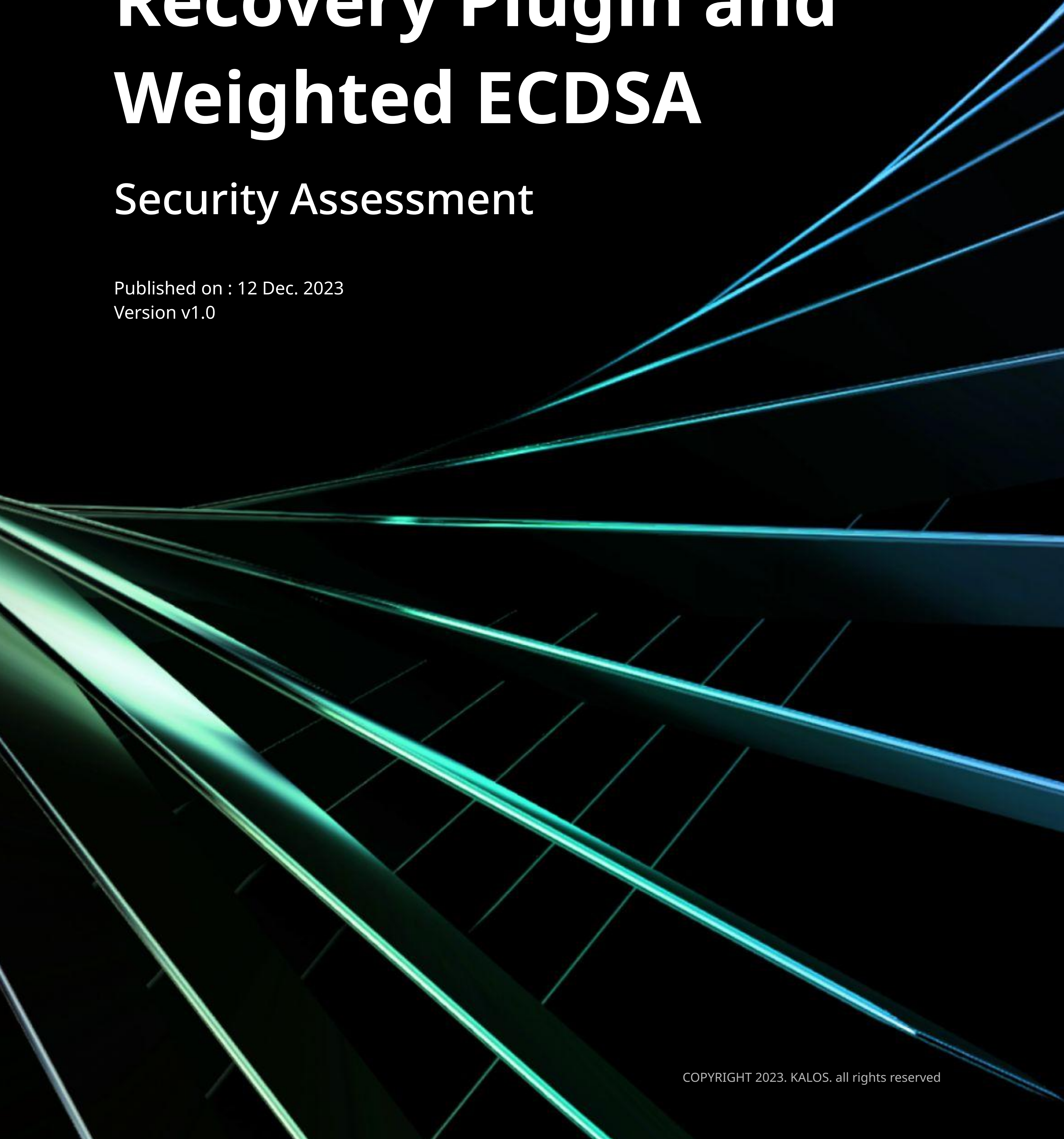
Making Web3 Space Safer for Everyone

KALOS

ZeroDev Recovery Plugin and Weighted ECDSA

Security Assessment

Published on : 12 Dec. 2023
Version v1.0



Security Report Published by KALOS

v1.0 12 Dec. 2023

Auditor : Jade Han

hojung han

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	2	2	-	-
High	3	3	-	-
Medium	-	-	-	-
Low	-	-	-	-
Tips	1	-	1	-

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

KALOS is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

Having secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges, KALOS is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the project developed by the ZeroDev team. KALOS conducted the audit focusing on whether the system created by the ZeroDev team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the project.

In detail, we have focused on the following

- Denial of Service
- Access Control of Various Storage Variables
- Access Control of Important Functions
- Freezing of User Assets
- Theft of User Assets
- Unhandled Exceptions
- Compatibility Testing with Bundler

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/zerodevapp/kernel/).

The commit hash of the code used for this Audit is "90fa72ed4386b3a4acb86c021743c5d56fc4f603"

The commit hash of the patched according to our recommendations is "f9461f1af9554540783f671b1762a913060fe081"

Audit Timeline

Date	Event
2023/11/30	Audit Initiation
2023/12/12	Delivery of v1.0 report.

Findings

KALOS found - 2 Critical, 3 High, 0 medium, 0 Low and 1 tips severity issues.

Severity	Issue	Status
Critical	Approval of Arbitrary Proposals Due to Lack of Verification in weightedStorage Threshold	(Resolved)
High	Absence of Handling for Previously Used Guardian Weights Before Updating New Guardian Information	(Resolved)
Critical	Potential Signature Replay Due to Flaw in the Creation Process of callDataAndNonceHash	(Resolved)
High	Temporary Approval of Proposals with Minimal Weight Possible Due to Flaws in Guardian Update Mechanism	(Resolved)
High	Possibility of Secondary Damage Due to BackDoor in the Event of Harm	(Resolved)
TIPS	Lack of Expiry Validation for Approved Proposals	(Acknowledged)

OVERVIEW

Recovery Plugin / Weighted ECDSA

The product under review is the ZeroDev Kernel Wallet. One notable feature of the ZeroDev Kernel Wallet is its ability to activate a RecoveryAction that allows for the modification of configuration information for a specific validator. Furthermore, ZeroDev rigorously enforces a comprehensive validation process when executing the RecoveryAction, using the WeightedECDSAValidator to ensure that the threshold for executing the RecoveryAction is met.

While the official ZeroDev documentation recommends utilizing the RecoveryAction in conjunction with the WeightedECDSAValidator, it is important to note that the RecoveryAction can also be integrated with other validators at a lower level. However, This approach is not recommended.

Scope

```
src
├── executor
|   ├── RecoveryAction.sol
└── validator
    ├── WeightedECDSAValidator.sol
```

*** We have verified whether the codes within the Scope are sufficiently compatible with the 4337 Specification using the Bundler (https://github.com/pimlicolabs/eth-infinitism-bundler).**

FINDINGS

1. Approval of Arbitrary Proposals Due to Lack of Verification in weightedStorage Threshold

ID: ZeroDev-RW-1	Severity: Critical
Type: Logic Error	Difficulty: Low
File: /src/validator/WeightedECDSAValidator.sol	

Issue

The functionality of the approveWithSig function involves performing ECDSA verification with a provided Signature for a Proposal based on specified User Operation callData and nonce in a specific wallet.

If the provided Signature is valid, the Signer's weight is added to the existing weightedApproved for the Proposal. If this sum exceeds the threshold set in weightedStorage, it transitions the Proposal to an approved state, allowing it to pass through validateUserOp.

```
function approveWithSig(bytes32 _callDataAndNonceHash, address _kernel, bytes calldata sigs)
external {
    uint256 sigCount = sigs.length / 65;
    ProposalStorage storage proposal = proposalStatus[_callDataAndNonceHash][_kernel];
    require(proposal.status == ProposalStatus.Ongoing, "Proposal not ongoing");
    for (uint256 i = 0; i < sigCount; i++) {
        address signer = ECDSA.recover(
            _hashTypedData(
                keccak256(abi.encode(keccak256("Approve(bytes32 callDataAndNonceHash)"),
_callDataAndNonceHash))
            ),
            sigs[i * 65:(i + 1) * 65]
        );
        VoteStorage storage vote = voteStatus[_callDataAndNonceHash][signer][_kernel];
        require(vote.status == VoteStatus.NA, "Already voted");
        vote.status = VoteStatus.Approved;
        proposal.weightApproved += guardian[signer][_kernel].weight;
    }
    if (proposal.weightApproved >= weightedStorage[_kernel].threshold) {
        proposal.status = ProposalStatus.Approved;
        proposal.validAfter = ValidAfter.wrap(uint48(block.timestamp +
weightedStorage[_kernel].delay));
    }
}
```

https://github.com/zerodevapp/kernel/blob/90fa72ed4386b3a4acb86c021743c5d56fc4f603/src/validator/WeightedECDSAValidator.sol#L99-L119

However, before the enable function of the WeightedECDSAValidator Contract is executed through a specific wallet, the threshold for that wallet in weightedStorage is always set to 0. Also, any address not approved in advance weights 0. Anyone can change to an `approved` state for any arbitrary proposal in the above situation because the weightedStorage threshold and the proposal's threshold are the same.

Recommendation

We recommend modifying the validateUserOp, approve, and approveWithSig functions to include a check that reverts the transaction if the threshold for weightedStorage of a specific wallet is set to zero. Additionally, to prevent users from inadvertently setting the threshold of weightedStorage to zero, it is advisable to incorporate a validation mechanism within the enable function to verify whether the user input for the threshold is zero. This implementation will enhance the overall security and integrity of the system.

Patch Comment

We have confirmed that the issue mentioned in this patch (https://github.com/zerodevapp/kernel/pull/53/commits/1566b952804ae0b47d9cdf42afb5c871eb517f96) has been resolved.

2. Absence of Handling for Previously Used Guardian Weights Before Updating New Guardian Information

ID: ZeroDev-RW-2
Severity: High
Type: Warning
Difficulty: Low
File: /src/validator/WeightedECDSAValidator.sol

Issue

The enable function in WeightedECDSAValidator to replace the existing guardian with a new one does not reset the previously used existing guardian's weight in a specific proposal.

```
function enable(bytes calldata _data) external payable override {
    (address[] memory _guardians, uint24[] memory _weights, uint24 _threshold, uint48
_delay) =
        abi.decode(_data, (address[], uint24[], uint24, uint48));
    require(_guardians.length == _weights.length, "Length mismatch");
    require(weightedStorage[msg.sender].totalWeight == 0, "Already enabled");
    weightedStorage[msg.sender].firstGuardian = msg.sender;
    for (uint256 i = 0; i < _guardians.length; i++) {
        require(_guardians[i] != address(0), "Guardian cannot be 0");
        require(_weights[i] != 0, "Weight cannot be 0");
        require(guardian[_guardians[i]][msg.sender].weight == 0, "Guardian already
enabled");
        guardian[_guardians[i]][msg.sender] =
            GuardianStorage({weight: _weights[i], nextGuardian:
weightedStorage[msg.sender].firstGuardian});

        weightedStorage[msg.sender].firstGuardian = _guardians[i];
        weightedStorage[msg.sender].totalWeight += _weights[i];
    }
    weightedStorage[msg.sender].delay = _delay;
    weightedStorage[msg.sender].threshold = _threshold;
}

function disable(bytes calldata) external payable override {
    require(weightedStorage[msg.sender].totalWeight != 0, "Not enabled");
    address currentGuardian = weightedStorage[msg.sender].firstGuardian;
    while (currentGuardian != msg.sender) {
        address nextGuardian = guardian[currentGuardian][msg.sender].nextGuardian;
        delete guardian[currentGuardian][msg.sender];
        currentGuardian = nextGuardian;
    }
    delete weightedStorage[msg.sender];
}
```

https://github.com/zerodevapp/kernel/blob/90fa72ed4386b3a4acb86c021743c5d56fc4f603/src/validator/WeightedECDSAValidator.sol#L55-L83

Recommendation

We recommend adding a member variable named version of type uint16 to the WeightedECDSAValidatorStorage structure. Similarly, it is advisable to include a version member variable of type uint16 in the ProposalStorage structure. Furthermore, increment the version of weightedStorage by one each time the enable function is executed. Subsequently, when a specific proposal is approved through the approve or approveWithSig functions, it is recommended to record the current version as the initial approval in its version. Additionally, we propose modifying the code in the approve, approveWithSig, and validateUserOp functions to revert the transaction if the current version of weightedStorage does not match the version set in the proposal or if the version of weightedStorage is 0.

Patch Comment

We have confirmed that the issue mentioned in this patch (https://github.com/zerodevapp/kernel/pull/53/commits/25b51798e339f524d5756754e317b984759ec936) has been resolved.

Although this patch does not follow the guidelines mentioned in the Recommendation, it is much more sophisticated and concise.

3. Potential Signature Replay Due to Flaw in the Creation Process of callDataAndNonceHash

ID: ZeroDev-RW-3
Severity: Critical
Type: Logic Error
Difficulty: High
File: /src/validator/WeightedECDSAValidator.sol

Issue

A specific Proposal can be identified using the wallet address and callDataAndNonceHash as follows:

```
function validateUserOp(UserOperation calldata userOp, bytes32, uint256)
    external
    payable
    returns (ValidationData validationData)
{
    bytes32 callDataAndNonceHash = keccak256(abi.encode(userOp.callData, userOp.nonce));

    ProposalStorage storage proposal = proposalStatus[callDataAndNonceHash][msg.sender];
    WeightedECDSAValidatorStorage storage strg = weightedStorage[msg.sender];
    ...
}
```

https://github.com/zerodevapp/kernel/blob/90fa72ed4386b3a4acb86c021743c5d56fc4f603/src/validator/WeightedECDSAValidator.sol#L130-L176

The callDataAndNonceHash is generated by concatenating the callData and nonce of the UserOperation and then applying the keccak256 encryption. The reason the method of generating callDataAndNonceHash is flawed can be illustrated by assuming the existence of Guardian1 (0xaaaa). Suppose Guardian1 is involved with Wallet A (0xbbbb) and Wallet B (0xcccc). In this scenario, due to the way callDataAndNonceHash is generated, the Signature used for a Proposal that has already changed its status to Executed in Wallet A can be reused in Wallet B. This is an abnormal situation, and it is evident that allowing a Signature used in Wallet A to be reused in Wallet B, even when Guardian1 is involved with both Wallets A and B, is fundamentally incorrect.

Recommendation

When composing callDataAndNonceHash, it should incorporate not only the User Operation's callData and nonce but also the sender that is wallet address.

Patch Comment

We have confirmed that the issue mentioned in this patch (https://github.com/zerodevapp/kernel/pull/53/commits/9bff92f9d7db0508131af15488edc4ad16bc577b) has been resolved.

4. Temporary Approval of Proposals with Minimal Weight Possible Due to Flaws in Guardian Update Mechanism

ID: ZeroDev-RW-4 Severity: High
Type: Logic Error Difficulty: High
File: /src/validator/WeightedECDSAValidator.sol

Issue

In the initial setup, to modify information about a Guardian, it is necessary to call the disable function and then invoke the enable function anew.

```
function enable(bytes calldata _data) external payable override {
    (address[] memory _guardians, uint24[] memory _weights, uint24 _threshold, uint48
_delay) =
        abi.decode(_data, (address[], uint24[], uint24, uint48));
    require(_guardians.length == _weights.length, "Length mismatch");
    require(weightedStorage[msg.sender].totalWeight == 0, "Already enabled");
    weightedStorage[msg.sender].firstGuardian = msg.sender;
    for (uint256 i = 0; i < _guardians.length; i++) {
        require(_guardians[i] != address(0), "Guardian cannot be 0");
        require(_weights[i] != 0, "Weight cannot be 0");
        require(guardian[_guardians[i]][msg.sender].weight == 0, "Guardian already
enabled");
        guardian[_guardians[i]][msg.sender] =
            GuardianStorage({weight: _weights[i], nextGuardian:
weightedStorage[msg.sender].firstGuardian});

        weightedStorage[msg.sender].firstGuardian = _guardians[i];
        weightedStorage[msg.sender].totalWeight += _weights[i];
    }
    weightedStorage[msg.sender].delay = _delay;
    weightedStorage[msg.sender].threshold = _threshold;
}

function disable(bytes calldata) external payable override {
    require(weightedStorage[msg.sender].totalWeight != 0, "Not enabled");
    address currentGuardian = weightedStorage[msg.sender].firstGuardian;
    while (currentGuardian != msg.sender) {
        address nextGuardian = guardian[currentGuardian][msg.sender].nextGuardian;
        delete guardian[currentGuardian][msg.sender];
        currentGuardian = nextGuardian;
    }
    delete weightedStorage[msg.sender];
}
```

https://github.com/zerodevapp/kernel/blob/90fa72ed4386b3a4acb86c021743c5d56fc4f603/src/validator/WeightedECDSAValidator.sol#L55-L83

This requirement exists due to the absence of a function designed to delete and create Guardian information atomically. Although the Kernel's executeBatch function is an good option, it is not mentioned in the official documentation. This situation poses a security risk: if a user executes the disable and enable functions in two separate transactions, an attacker could insert an arbitrary UserOperation transaction following the disable function transaction. This could allow the transaction to pass through the validateUserOp function without a signature.

Recommendation

We recommended adding a new function that allows for the atomic updating of Guardian information within a single transaction, separate from the existing enable and disable functions. This enhancement will streamline the process and significantly reduce the security vulnerabilities of splitting the update into two transactions.

If the recommendations for ZeroDev-RW-1 are implemented, it will also prevent the vulnerability mentioned in ZeroDev-RW-4. However, for an enhanced user experience, we have formulated additional recommendations.

Patch Comment

We have confirmed that the issue mentioned in this patch (https://github.com/zerodevapp/kernel/pull/53/commits/f9461f1af9554540783f671b1762a913060fe081) has been resolved.

5. Possibility of Secondary Damage Due to BackDoor in the Event of Harm

ID: ZeroDev-RW-5 Severity: High
Type: Logic Error Difficulty: High
File: /src/validator/WeightedECDSAValidator.sol

Issue

A logic error in the enable function inserts the wallet address at the end of the Guardian List. When the disable function initializes this Guardian List, it checks whether the list's end contains a wallet address. This process is defined in the following Solidity code:

```
function enable(bytes calldata _data) external payable override {
    (address[] memory _guardians, uint24[] memory _weights, uint24 _threshold, uint48
_delay) =
        abi.decode(_data, (address[], uint24[], uint24, uint48));
    require(_guardians.length == _weights.length, "Length mismatch");
    require(weightedStorage[msg.sender].totalWeight == 0, "Already enabled");
    weightedStorage[msg.sender].firstGuardian = msg.sender;
    for (uint256 i = 0; i < _guardians.length; i++) {
        require(_guardians[i] != address(0), "Guardian cannot be 0");
        require(_weights[i] != 0, "Weight cannot be 0");
        require(guardian[_guardians[i]][msg.sender].weight == 0, "Guardian already
enabled");
        guardian[_guardians[i]][msg.sender] =
            GuardianStorage({weight: _weights[i], nextGuardian:
weightedStorage[msg.sender].firstGuardian});

        weightedStorage[msg.sender].firstGuardian = _guardians[i];
        weightedStorage[msg.sender].totalWeight += _weights[i];
    }
    weightedStorage[msg.sender].delay = _delay;
    weightedStorage[msg.sender].threshold = _threshold;
}
...
function disable(bytes calldata) external payable override {
    require(weightedStorage[msg.sender].totalWeight != 0, "Not enabled");
    address currentGuardian = weightedStorage[msg.sender].firstGuardian;
    while (currentGuardian != msg.sender) {
        address nextGuardian = guardian[currentGuardian][msg.sender].nextGuardian;
        delete guardian[currentGuardian][msg.sender];
        currentGuardian = nextGuardian;
    }
    delete weightedStorage[msg.sender];
}
```

https://github.com/zerodevapp/kernel/blob/90fa72ed4386b3a4acb86c021743c5d56fc4f603/src/validator/WeightedECDSAValidator.sol#L55-L72

Consider a scenario where a user inputs guardian value in the enable function, including wallet address. When the disable function is executed to reset the Guardian information, it recognizes the wallet address in the middle of the Guardian list and stops the loop. Consequently, any Guardian positioned after the Wallet address in the Guardian List is not deleted. If attackers can temporarily seize control of a specific wallet due to user negligence, this flaw provides an opportunity to install a backdoor.

Recommendation

We recommend modifying the enable function to ensure that msg.sender is not included in the user input guardian values. The transaction should be reverted if msg.sender is found among these values. This will effectively prevent the insertion of wallet addresses into the Guardian List, thus significantly reducing the risk of secondary damage through an installed backdoor.

Patch Comment

We have confirmed that the issue mentioned in this patch (https://github.com/zerodevapp/kernel/pull/53/commits/5d7ebc0db61302dc05c44c9934f91da70ef01b5a) has been resolved.

6. Lack of Expiry Validation for Approved Proposals

ID: ZeroDev-RW-6 Severity: Tips
Type: - Difficulty: -
File: /src/validator/WeightedECDSAValidator.sol

Issue

The process of handling a proposal currently only considers its validAfter status. This mechanism is primarily used to determine when a particular proposal, once approved, becomes executable.

```
struct ProposalStorage {
    ProposalStatus status;
    ValidAfter validAfter;
    uint24 weightApproved;
    uint48 firstApprovedAt;
}
```

https://github.com/zerodevapp/kernel/blob/90fa72ed4386b3a4acb86c021743c5d56fc4f603/src/validator/WeightedECDSAValidator.sol#L55-L72

However, once a proposal reaches the approved status, the system only checks its validity from a start time perspective, neglecting to verify when the proposal should no longer be valid.

Recommendation

For a safer environment, it is recommended to introduce a mechanism that checks the expiry date of a specific proposal. This addition would ensure that proposals are verified for when they become effective and when they should cease to be valid. Implementing such a feature would significantly enhance the security and integrity of the proposal-handling process.

Patch Comment

The ZeroDev Team has decided not to proceed with a patch for the issue above.
Opting not to implement the patch does not result in any additional security threats.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure codes, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	**Medium**	**High**
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document