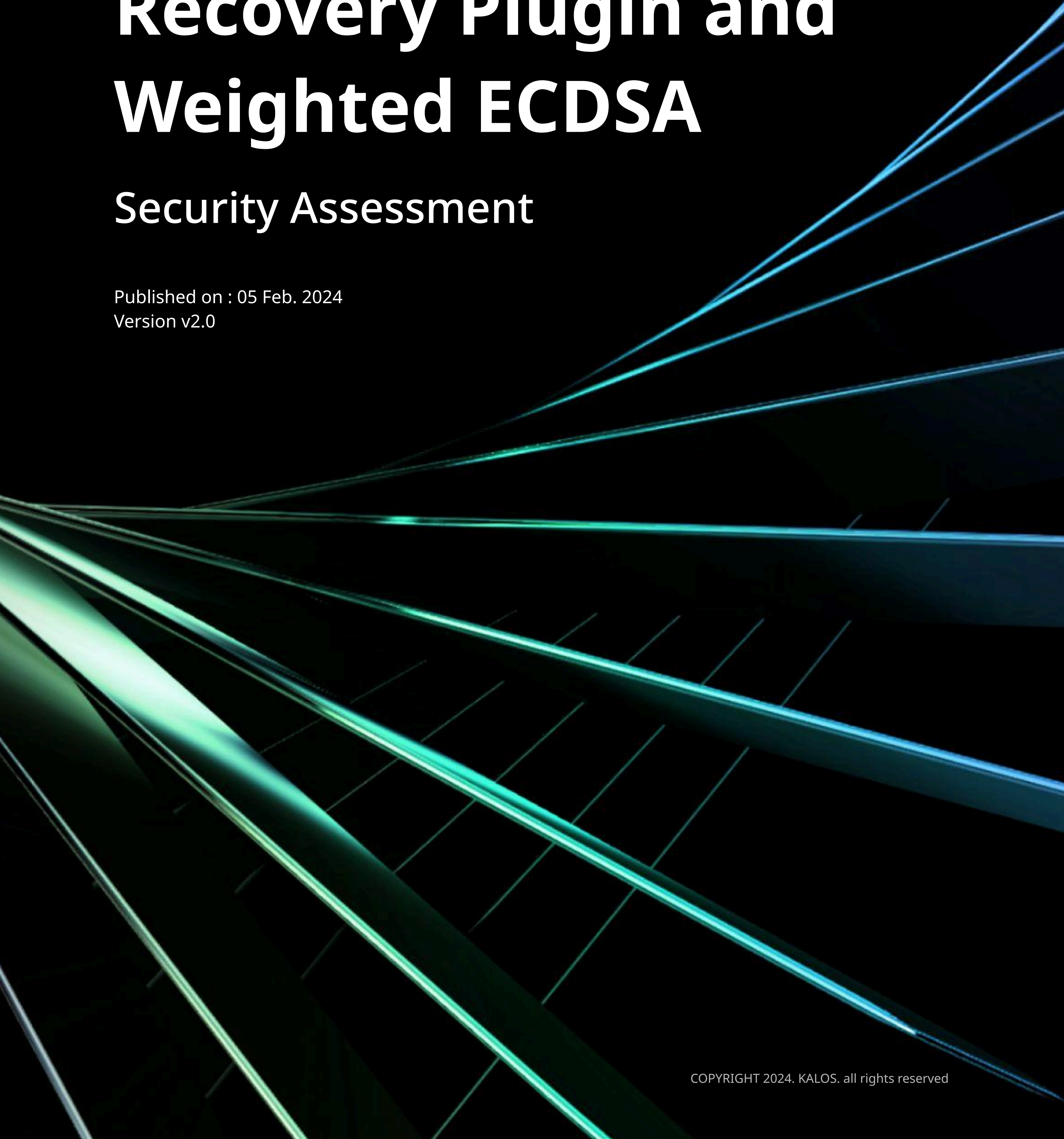
Making Web3 Space Safer for Everyone

ZeroDev Recovery Plugin and Weighted ECDSA

Security Assessment

Published on : 05 Feb. 2024
Version v2.0



Security Report Published by KALOS

v2.0 05 Feb. 2024

Auditor : Jade Han

hojung han

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

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

Pioneering a safer Web3 space since 2018, KALOS proudly won 2nd place in the Paradigm CTF 2023. As a leader in the global blockchain industry, we unite the finest in Web3 security expertise.

Our team consists of top security researchers with expertise in blockchain/smart contracts and experience in bounty hunting. Specializing in the audit of mainnets, DeFi protocols, bridges, and the zkEVM protocol, KALOS has successfully safeguarded billions in crypto assets.

Supported by grants from the Ethereum Foundation and the Community Fund, we are dedicated to innovating and enhancing Web3 security, ensuring that our clients' digital assets are securely protected in the highly volatile and ever-evolving Web3 landscape.

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 "eaaac83acd3e7695c742758a54828874ec0dcdee"

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

Audit Timeline

Date	Event
2023/11/30	Audit Initiation
2023/12/12	Delivery of v1.0 report.
2024/02/02	Audit Initiation
2024/02/05	Delivery of v2.0 report.

Findings

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

Severity	Issue	Status
Critical	Voting Power Duplication Vulnerability in validateSignature	(Found)
Critical	Incorrect Verification of paymasterAndData Leading to Reduction of Wallet Deposits in EntryPoint	(Found)
Low	Absence to Update Proposal Status	(Found)

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 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
└── 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/alto)**

FINDINGS

1. Voting Power Replication Vulnerability in validateSignature

ID: ZeroDev-RW2-1	Severity: Critical
Type: Logic Error	Difficulty: Low
File: /src/validator/WeightedECDSAValidator.sol	

Issue

The recent update to the `WeightedECDSAValidator` Contract introduced a revised `validateSignature` function intended for use as the default validator in the `Kernel` Contract.

This function performs signature validation by aggregating the voting power of multiple signers to meet a predefined threshold.

```
function validateSignature(bytes32 hash, bytes calldata signature) external view returns
(ValidationData) {
    WeightedECDSAValidatorStorage storage strg = weightedStorage[msg.sender];
    if (strg.threshold == 0) {
        return SIG_VALIDATION_FAILED;
    }

    uint256 sigCount = signature.length / 65;
    if (sigCount == 0) {
        return SIG_VALIDATION_FAILED;
    }
    uint256 totalWeight = 0;
    address signer;
    for (uint256 i = 0; i < sigCount; i++) {
        signer = ECDSA.recover(hash, signature[i * 65:(i + 1) * 65]);
        totalWeight += guardian[signer][msg.sender].weight;
        if (totalWeight >= strg.threshold) {
            return packValidationData(ValidAfter.wrap(0), ValidUntil.wrap(0));
        }
    }
    return SIG_VALIDATION_FAILED;
}
```

https://github.com/zerodevapp/kernel/blob/eaaac83acd3e7695c742758a54828874ec0dcdee/src/validator/WeightedECDSAValidator.sol#L255-L276

However, an examination of the function's implementation reveals a critical vulnerability that allows the manipulation of the validation process through the repeated submission of a single signer's signature, artificially inflating the voting power and thus undermining the safety of the signature validation mechanism.

Recommendation

To mitigate vulnerabilities, it is recommended to add the signer of a verified signature to a list and later check if the signer of any subsequent signature is included in that list. If the signer is found within the list, implement a mitigation by reverting the transaction.

Patch Comment

-

2. Incorrect Verification of paymasterAndData Leading to Reduction of Wallet Deposits in EntryPoint

ID: ZeroDev-RW2-2	Severity: Critical
Type: Logic Error	Difficulty: Low
File: /src/validator/WeightedECDSAValidator.sol	

Issue

The execution of UserOperation data within the ERC4337 EntryPoint Contract is facilitated through the handleOps function. Within this function, the _copyUserOpToMemory method is employed to reference the paymasterAndData field of UserOperation for extracting the paymaster address. The critical code segment is as follows:

```
function _copyUserOpToMemory(UserOperation calldata userOp, MemoryUserOp memory mUserOp)
internal pure {
    mUserOp.sender = userOp.sender;
    mUserOp.nonce = userOp.nonce;
    mUserOp.callGasLimit = userOp.callGasLimit;
    mUserOp.verificationGasLimit = userOp.verificationGasLimit;
    mUserOp.preVerificationGas = userOp.preVerificationGas;
    mUserOp.maxFeePerGas = userOp.maxFeePerGas;
    mUserOp.maxPriorityFeePerGas = userOp.maxPriorityFeePerGas;
    bytes calldata paymasterAndData = userOp.paymasterAndData;
    if (paymasterAndData.length > 0) {
        require(paymasterAndData.length >= 20, "AA93 invalid paymasterAndData");
        mUserOp.paymaster = address(bytes20(paymasterAndData[: 20]));
    } else {
        mUserOp.paymaster = address(0);
    }
}
```

https://github.com/eth-infinitism/account-abstraction/blob/a10b3380892e5f6a26ae9f025b71d1aa26862b16/contracts/core/EntryPoint.sol#L351-L372

If paymasterAndData.length is zero, the paymaster address is set to address(0), and if not, the first 20 bytes of paymasterAndData are used as the paymaster address. A closer inspection of the code reveals a scenario where if the first 20 bytes of paymasterAndData are set to address(0), it could result in the paymaster address being set to address(0) even if the first condition in the if statement is met.

This could lead to the execution of UserOperation using the wallet owner's funds instead of the paymaster's, as demonstrated in the following code:

```
function _validateAccountPrepayment(uint256 opIndex, UserOperation calldata op, UserOpInfo
memory opInfo, uint256 requiredPrefund)
    internal returns (uint256 gasUsedByValidateAccountPrepayment, uint256 validationData) {
    unchecked {
        ...
        if (paymaster == address(0)) {
            uint256 bal = balanceOf(sender);
            missingAccountFunds = bal > requiredPrefund ? 0 : requiredPrefund - bal;
        }
        ...
        if (paymaster == address(0)) {
            DepositInfo storage senderInfo = deposits[sender];
            uint256 deposit = senderInfo.deposit;
            if (requiredPrefund > deposit) {
                revert FailedOp(opIndex, "AA21 didn't pay prefund");
            }
            senderInfo.deposit = uint112(deposit - requiredPrefund);
        }
        ...
    }
}
```

https://github.com/eth-infinitism/account-abstraction/blob/a10b3380892e5f6a26ae9f025b71d1aa26862b16/contracts/core/EntryPoint.sol#L441-L482

In response, the WeightedECDSAValidator code includes checks to prevent the omission of the paymaster by malicious validator by validating the userOpHash signature again if paymasterAndData is empty. However, filling paymasterAndData with 20 bytes set to address(0) could bypass the mitigation strategy that checks for the length of paymasterAndData.

```
} else if (proposal.status == ProposalStatus.Approved || passed) {
        if(userOp.paymasterAndData.length == 0) {
            address signer = ECDSA.recover(ECDSA.toEthSignedMessageHash(userOpHash),
userOp.signature);
            if (guardian[signer][msg.sender].weight != 0) {
                proposal.status = ProposalStatus.Executed;
                return packValidationData(proposal.validAfter, ValidUntil.wrap(0));
            }
        } else {
            return packValidationData(proposal.validAfter, ValidUntil.wrap(0));
        }
    }
    return SIG_VALIDATION_FAILED;
}
```

https://github.com/zerodevapp/kernel/blob/eaaac83acd3e7695c742758a54828874ec0dcdee/src/validator/WeightedECDSAValidator.sol#L223-L235

Recommendation

To mitigate this issue, it is essential to update the code to perform validation of the userOpHash whenever paymasterAndData's length is zero or the first 20 bytes of paymasterAndData match address(0). This adjustment ensures a more robust verification process, preventing potential wallet fund misuse by addressing the scenario outlined above.

Patch Comment

-

3. Absence to Update Proposal Status

ID: ZeroDev-RW2-3 Severity: Low
Type: Logic Error Difficulty: Low
File: /src/validator/WeightedECDSAValidator.sol

Issue

The EntryPoint Contract incorporates a Nonce Manager and utilizes the callDataAndNonceHash variable for hashing operations, taking into account the nonce from UserOperation as per the recommendations from the initial audit of the WeightedECDSAValidator. This approach effectively eliminates the risk of replay attacks. However, the current implementation is not considered best practice, and there is uncertainty regarding potential side effects that may arise in the future. It is advised to address this issue to prevent any unintended consequences.

```
function validateUserOp(UserOperation calldata userOp, bytes32 userOpHash, uint256)
   external
   payable
   returns (ValidationData validationData)
{
   bytes32 callDataAndNonceHash = keccak256(abi.encode(userOp.sender, userOp.callData,
userOp.nonce));
   ProposalStorage storage proposal = proposalStatus[callDataAndNonceHash][msg.sender];
   WeightedECDSAValidatorStorage storage strg = weightedStorage[msg.sender];
   ...
   } else if (proposal.status == ProposalStatus.Approved || passed) {
       if(userOp.paymasterAndData.length == 0) {
           address signer = ECDSA.recover(ECDSA.toEthSignedMessageHash(userOpHash),
userOp.signature);
           if (guardian[signer][msg.sender].weight != 0) {
               proposal.status = ProposalStatus.Executed;
               return packValidationData(proposal.validAfter, ValidUntil.wrap(0));
           }
       } else {
           return packValidationData(proposal.validAfter, ValidUntil.wrap(0));
       }
   }
   return SIG_VALIDATION_FAILED;
}
```

https://github.com/zerodevapp/kernel/blob/eaaac83acd3e7695c742758a54828874ec0dcdee/src/validator/WeightedECDSAValidator.sol#L168-L235

Recommendation

It is recommended to enhance the security of the EntryPoint Contract by updating the proposal status to Executed when the proposal status is Approved, and the UserOperation has an adequately set paymaster.

Patch Comment

-

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