Smart Contract Audit Report

Client: ZeroDev

Project: Kernel 7579 Plugins and Kernel

Audit Period: May 27, 2024 - June 9, 2024

Auditors: Felix Kim

About the auditor

The journey began in 2019 at HAECHI LABS, where over 100 smart contracts were audited, focusing on Decentralized Finance (DeFi) systems. Responsibilities included high-profile projects such as Shoyu and SuperRare, and the development of an automated auditing tool that enhanced both the efficiency and consistency of the auditing processes.

Continuing contributions to the field are made as a security researcher. Research is dedicated to enhancing security across various platforms and technologies. Notable projects include the development of an advanced fuzzing framework for detecting memory corruption vulnerabilities in Linux device drivers, and designing an emulation environment to detect security vulnerabilities in Android phone bootloaders through sophisticated fuzzing techniques. This depth of experience in practical security applications ensures a thorough and rigorous approach to smart contract audits.

Audited Components

Validators

Validators are a type of module that helps determine if a transaction is valid and should be executed on a smart account. They perform certain checks and validations to ensure the integrity and authenticity of transactions before they are processed by the account.

- **WebAuthn Validator**
  - **File Location:** WebAuthnValidator.sol
  - **Commit Hash:** ae10aa0f
  - **Description:** This contract serves as a validator within the context of ERC-7579, specifically handling WebAuthn protocol-based authentication.

- **MultiChain WebAuthn Validator**
  - **File Location:** MultiChainWebAuthnValidator.sol
  - **Commit Hash:** 817e749
  - **Description:** An extension of the WebAuthnValidator that supports multiple blockchain environments. It validates signatures using a Merkle tree.

- **Weighted Validator**
  - **File Location:** WeightedValidator.sol
  - **Commit Hash:** 91f8fcb
  - **Description:** Implements a weighted validation mechanism, allowing different impacts of validators based on their assigned weights.

Hooks

Hooks are an optional feature that allows smart accounts to execute custom logic and perform additional checks before and after executing transactions. They provide a way to extend and customize the behavior of smart accounts based on specific requirements or use cases. Hooks can be used to implement

various functionalities such as access control, logging, or triggering external actions based on certain conditions.

- **Spending Limit**
  - **File Location:** SpendingLimit.sol
  - **Commit Hash:** 9dc7fcd
  - **Description:** It manages and enforces spending allowances for specific tokens, controlling how much a user can transact.

Kernel

- **Version Comparison**
  - **Comparison:** Kernel Version 3.0 to Future Version
  - **Description:** The kernel is reviewed for changes between versions 3.0 and 3.x with a focus on enhancements. This review was specifically on the changes rather than the entire codebase. The kernel is noted for its compatibility with ERC-4337 and supports modules with ERC-7579 plugins.

Findings

Summary

severity	Description	status
minor	Lack of Implementation in onUninitialize() Method in WeightedValidator	found v.1.0
minor	No Duplicate Address Verification in WeightedValidator and SpendingLimit	found v.1.0
minor	postCheck Function Limited to Checking a Single Token's Balance	found v.1.0

Lack of Implementation in onUninstall() Method in WeightedValidator

```
function onUninstall(bytes calldata) external payable override {
    if (!_isInitialized(msg.sender)) revert NotInitialized(msg.sender);
    // TODO remove everything
}
```

Description

The onUninstall function in the WeightedValidator contract currently checks if msg.sender is initialized and throws an error if it is not. However, there is no code to clear the storage variables associated with msg.sender when it is initialized. This oversight leads to malfunctioning of the _isInitialized function, which relies on these storage variables to determine the initialization state. For instance, the onInstall function should not be callable if msg.sender has been initialized, but due to the current implementation, it remains callable.

Recommendation

Implement logic within the onUninstall function to properly clear all related storage variables.

No Duplicate Address Verification in WeightedValidator and SpendingLimit

```
   function _addGuardians(bytes[] calldata guardianData, address _kernel)
internal {
       uint24 totalWeight = weightedStorage[_kernel].totalWeight;
       for (uint256 i = 0; i < guardianData.length; i++) {
           bytes calldata g = guardianData[i];
           bytes1 guardianType = bytes1(g[0]);
           uint24 weight = uint24(bytes3(g[1:4]));
           totalWeight += weight;
           if (guardianType == 0x01) {
               // k1
               if (g.length != 24) {
                   revert WrongGuardianDataLength();
               }
               guardian[i][_kernel].guardianType = bytes1(0x01);
               guardian[i][_kernel].weight = weight;
               guardian[i][_kernel].encodedPublicKey = g[4:24];
               emit GuardianAddedK1(_kernel, i, address(bytes20(g[4:24])),
weight);
           } else if (guardianType == 0x02) {
               // r1
               if (g.length != 100) {
                   revert WrongGuardianDataLength();
               }
               guardian[i][_kernel].guardianType = bytes1(0x02);
               guardian[i][_kernel].weight = weight;
               guardian[i][_kernel].encodedPublicKey = g[4:100]; // this will
be abi.encodePacked(x,y,authenticatorIdHash)
               emit GuardianAddedR1(
                   _kernel, i, bytes32(g[68:100]), uint256(bytes32(g[4:36])),
uint256(bytes32(g[36:68])), weight
```

```
                );
            } else {
                revert NotSupportedSignatureType();
            }
        }
        weightedStorage[_kernel].totalWeight = totalWeight;
    }
```

WeightedValidator#_addGuardians

```
    function onInstall(bytes calldata data) external payable {
        require(listLength[msg.sender] == 0, "already initialized");
        bytes[] calldata arr = _parseCalldataArrayBytes(data);
        for (uint256 i = 0; i < arr.length; i++) {
            spendingLimit[i][msg.sender] =
                    SpendingLimitData({token: address(bytes20(arr[i][0:20])),
allowance: uint256(bytes32(arr[i][20:52]))});
        }
        listLength[msg.sender] = arr.length;
    }
```

SpendingLimit#onInstall

Description

In the current implementations, the WeightedValidator's _addGuardians function and the SpendingLimit's onInstall function register guardian addresses and token addresses, respectively. Neither of these functions includes a mechanism to check for and prevent the registration of duplicate addresses. This oversight allows for the potential registration of the same address multiple times, which could lead to unintended behavior or vulnerabilities in the management of guardians or token spending limits.

Recommendation

It is recommended to incorporate a routine to verify whether an address is already registered before allowing it to be added again in both the WeightedValidator and SpendingLimit contracts.

[Minor] postCheck Function Limited to Checking a Single Token's Balance

```
    function postCheck(bytes calldata context, bool, bytes calldata) external payable override {
        uint256 length = listLength[msg.sender];
        uint256[] memory preBalance = abi.decode(context, (uint256[]));
        for (uint256 i = 0; i < length; i++) {
            SpendingLimitData storage data = spendingLimit[i][msg.sender];
            uint256 balance;
            if (data.token == address(0)) {
                balance = msg.sender.balance;
            } else {
                balance = ERC20(data.token).balanceOf(msg.sender);
            }
            // if balance increased, skip the allowance check
            if (balance > preBalance[i]) {
                return;
            }
            uint256 used = preBalance[i] - balance;
            if (data.allowance < used) {
                revert ExceedsAllowance();
            }
            data.allowance -= used;
        }
    }
```

SpendingLimit#postCheck

Description

The postCheck function is executed after specific operations are carried out, intended to verify the balances of tokens registered in SpendingLimitData against their allowances. However, the current implementation of the

postCheck logic only checks the balance of one token when its balance has changed. This means that it does not adequately handle scenarios where multiple token balances are altered simultaneously.

Recommendation

If there is a need to verify multiple tokens following a transaction, it is recommended to revise the logic to ensure all relevant token balances are checked.

Disclaimer

This audit report provides an independent evaluation of the smart contracts listed herein as requested by ZeroDev. The findings, interpretations, and conclusions expressed in this report are those of the auditors based on the information and source code made available to us at the time of the audit. This report does not constitute investment advice, legal advice, or any other type of advice.

While every effort has been made to ensure accuracy and thoroughness, the nature of software development and the risks associated with blockchain technology mean that absolute security cannot be guaranteed. This report should not be seen as an endorsement of any kind of the security practices involved. ZeroDev team is advised to continue conducting regular security assessments and updates to the audited contracts.