QuillAudits

Audit Report
November, 2023

For

BASTION

Table of Content

Table of Content

Executive Summary

Project Name	Bastion Wallet
Project URL	*https://bastionwallet.io/*
Overview	It is a fork of **ZeroDev Kernel** implementation. Github: **https://github.com/zerodevapp/kernel** Zero Dev's Kernel is a minimal and extensible smart contract account where one can plug in new validators and executors. We are using ECDSA and MultiECDSA validators. In executors, we have added BatchAction executor. Additionally, KernelStorage is modified to have setOwner functionality and based on that other changes are done in TempKernel and Kernel.sol
Audit Scope	**https://github.com/bastion-wallet/kernel** Git Branch: main
Contracts in Scope	- src/Kernel.sol - src/executor/BatchActions.sol - src/executor/KillSwitchAction.sol - src/factory/ECDSAKernelFactory.sol - src/factory/EIP1967Proxy.sol - src/factory/KernelFactory.sol - src/factory/MultiECDSAKernelFactory.sol - src/validator/ECDSAValidator.sol - src/validator/ERC165SessionKeyValidator.sol - src/validator/KillSwitchValidator.sol - src/validator/MultiECDSAValidator.sol - src/validator/SessionKeyOwnedValidator.sol - All the corresponding interfaces and imported contracts within these above contracts (except the standard Openzeppelin libraries)

Executive Summary

Commit Hash	79dc2ff67117c8e34f0a1559e35966a259b12b48
Language	Solidity, Yul
Blockchain	Polygon, Arbitrum, Optimism, Scroll, Base, Linea
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	28th August 2023 - 16th October 2023
Updated Code Received	25th October 2023
Review 2	25th October 2023 - 7th November 2023
Fixed In	Branch: Fix/Security-1 454ce19a94638222e3d52632f0138f488e38d7c4

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	0	1	4
Partially Resolved Issues	0	0	0	2
Resolved Issues	0	0	5	5

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Statistic Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. BastionWallet

High Severity Issues

A.1: Missing unit tests

Description

It is highly recommended to have above 95% of code functionality tested before going into production so as to catch bugs that could be introduced from user input as well as return values from function calls.

Remediation

Consider using recent development frameworks like Hardhat or Foundry to write tests that cover all interactions in the codebase and adequately test for all branches in the codebase.

Status

Acknowledged

Medium Severity Issues

No issues were found.

Low Severity Issues

A.2: Solidity version incompatibility

Description

The Kernel smart wallets are expected to be deployed on multiple EVM-based chains, some of which are Optimism, Arbitrum, Scroll, Linea, Polygon and Base) however solidity pramga versions above 0.8.19 may prove incompatible on chains like Arbitrum which do not support the PUSH0 opcode. To ensure the same deterministic bytecode is available across all chains, consider locking the pragma versions to 0.8.19.

Remediation

Lock the solidity versions in use to <=0.8.19.

Status

Resolved

A.3: Array update pattern

Line

335

MultiECDSAKernelFactory - setOwners()

```
// @audit no checks for duplicate accounts | equal voting rights in multi-sig
function setOwners(address[] calldata _owners) external onlyOwner {
    owners = _owners;
}
```

Description

The array is totally replaced without noting that the owner array is completely rewritten with every setOwners() call and not appended.

Remediation

Ensure that the owner is aware of the array update pattern (override not append).

Status

Acknowledged

A.4: Single-step ownership transfer

MultiECDSAKernelFactory: onlyOwner

```
// @audit no checks for duplicate accounts | equal voting rights in multi-sig
function setOwners(address[] calldata _owners) external onlyOwner {
    owners = _owners;
}

function addStake(uint32 unstakeDelaySec) external payable onlyOwner {
    entryPoint.addStake{value: msg.value}(unstakeDelaySec);
}

function unlockStake() external onlyOwner {
    entryPoint.unlockStake();
}

function withdrawStake(address payable withdrawAddress) external onlyOwner {
    entryPoint.withdrawStake(withdrawAddress);
}
```

Description

The MultiECDSAKernelFactory utilizes the onlyOwner modifier inherited from the Ownable library. Functions in this contract with this modifier adjust key parameters in the entrypoint's stake and the owners array. If the owner (from onlyOwner) is compromised, stake can be unlocked and withdrawn, or malicious users can be added to the `owners` array.

Remediation

Use Two-Step Ownable to mitigate the risk of a single owner being compromised or implement a governance mechanism to ensure that changes passed have been vetted by multiple signatories.

Status

Resolved

A.5: Kernel can interact with an EOA instead of a contract

Description

In Kernel, it is expected that the execute() function is called with the target contract `to` being an external contract. The risk here is that an EOA can be delegatecalled or called and returns true as though it were a successful contract call. The execution fails to happen and there would be no reverts to show why.

Remediation

Check for code existence at the address of the target contract.

Comment

In case of ETH transfers, `to` address can be an EOA hence it is marked as resolved.

Status

Resolved

A.6: Unchecked logic address in EIP1967 constructor

Description

The logic address passed in to the EIP1967 proxy is used to deploy new kernel accounts. Although this happens only once, when the proxy is deployed there is no check for address validity. An EOA can be passed into the constructor which will pass the delegatecall without reverting.

Remediation

Ensure the address is checked to be able to process the data passed into it.

Status

Resolved

A.7: Risk of stuck funds in proxy contracts

Description

There is a slight discrepancy between the OpenZeppelin **implementation** and the EIP1967Proxy **implementation**, as msg.value is not checked. The check for msg.value is because when the data length passed is 0, it does not initiate the delegatecall to the logic contract and the value passed can get stuck in the contract.

Remediation

Based on the Proxy **documentation** by OpenZeppelin the msg.value should be zero if data length is zero, this happens by the else check which contains the _checkNotPayable internal function.

Status

Resolved

Informational Issues

A.8: Naming convention

Description

There are multiple mentions of zerodev.kernel in the codebase. Understandably, this is a fork of the zerodev project but these portions of the codebase can be refactored as there are no hardcoded calculations making strict use of the storagePosition formed by hashing the string “zerodev.kernel”.

Remediation

Refactor all appearances of ‘zerodev.kernel’ to ‘indorse.kernel’ or a project specific string.

Status

Acknowledged

A.9: enable() and disable() in validators do not emit events

Description

The ECDSAValidator does not emit an event for disabling validators. Monitoring processes would be easier to perform if logs were generated for account deletion.

Remediation

Emit an OwnerDisabled event.

Status

Partially Resolved

A.10: Missing input validation

Description

There is no validation of the `to` addresses in the executeBatch() as well as the `tokenAddress` array of approveAndTransfer20Batch() of BatchActions.sol. If there is no check, transfers can go to address(0), value and data could also be zero values.

The length of the arrays should also be verified to be equal to avoid array mismatch errors.

Remediation

Perform proper input validation checks.

Status

Resolved

General Recommendations / Gas Optimizations

The following are some gas optimizations that can be performed to lower gas costs per transaction.

A.11: Array operation optimization in BatchActions.sol

Description

In a bid to save gas, calldata is preferred over memory, especially in user-supplied arrays that are looped over. Since the array content is not modified, the function parameters can be declared as calldata instead of memory.

```
function executeBatch(address[] memory to, uint256[] memory value, bytes[] memory data,..) external
{
    ...
}

function approveAndTransfer20Batch(address[] memory tokenAddress, uint256[] memory amount,
address[] memory to) external
{
    ...
}
```

Remediation

Use calldata instead of memory to save gas.

Status

Partially Resolved

A.12: Redundant code

Description

In the validateUserOp function in Kernel.sol, there is a redundancy found in variable assignment and declaration. It is unnecessary to initialize variables to the default value (e.g. uint256 i = 0), or to reassign the same variable to itself (unchanged) in the same function execution.

In the Kernel.sol file, `op` is assigned the same value twice.

```
Kernel.sol
UserOperation memory op = userOp;
...
if (mode == 0x00000000) {
...
op = userOp;
```

Remediation

Remove redundant code.

Status

Resolved

A.13: Function visibility specifiers can be external instead of public

Description

A number of externally facing functions in the codebase are declared public but are not called internally or cross-contract. If there is no chance to reuse these functions in an internal contract call (because external functions cannot be called within the same contract), as external functions are cheaper than public functions it would be beneficial to adjust the function visibility.

In KillSwitchValidator.sol, ECDSAValidator.sol, ERC165SessionKeyValidator, MultiECDSAValidator, SessionKeyOwnedValidator, the functions (i) getAccountAddress() and (ii) validateSignature() can be declared external instead of public.

Remediation

Update the externally facing functions to external visibility instead of public.

Status

Resolved

A.14: Error handling

Description

Custom errors can be used to reduce gas costs for returning long strings. They are usually cheaper than string reverts.

Remediation

Use custom errors where possible.

Status

Acknowledged

A.15: Logic in validateSignature() can be reused within validateUserOp()

Description

The function validateSignature() tries to recover data using the ECDSA scheme for hashed data and user-inputted data as well. The same happens in validateUserOp within the SessionKeyValidator, MultiECDSAValidator and ECDSAValidator. The repeated code can be reduced.

Remediation

The repeated code can be made integral by calling validateSignature() within validateUserOp().

Status

Acknowledged

A.16: Unused interface

Description

IKernel is an empty interface declared and not used within the contract.

Remediation

If IKernel is intended to be unused, remove its declaration from the codebase.

Status

Resolved

A.17: Convert repeated code into a modifier

Description

The check (msg.sender == entrypoint) is used multiple times and can be converted into a modifier.

Remediation

Convert repeated lines of code to a modifier.

Status

Resolved

A.18: Repeated addresses can cost extra gas when iterating over owners array

Line

335

MultiECDSAKernelFactory - setOwners()

```
// @audit no checks for duplicate accounts | equal voting rights in multi-sig
function setOwners(address[] calldata _owners) external onlyOwner {
    owners = _owners;
}
```

Description

When setOwners in MultiECDSAKernelFactory is called, it passes an array of addresses but does not check whether the addresses passed are unique. Non-unique addresses can unnecessarily increase the length of the array thereby increasing the number of times the loop has to run.

Remediation

Ensure proper input validation.

Status

Acknowledged

Functional Tests

Some of the tests performed are mentioned below:

- [PASS] test extreme values for validation data
- [PASS] test validUntil and validAfter in multiple scenarios
- [PASS] test_callcode() (gas: 247855)
- [PASS] test_disable_mode() (gas: 175713)
- [PASS] test_initialize_twice() (gas: 20646)
- [PASS] test_set_default_validator() (gas: 350172)
- [PASS] test_set_execution() (gas: 400458)
- [PASS] test_validate_signature() (gas: 190118)
- [PASS] test_mode_2() (gas: 533084)
- [PASS] test_mode_2_1() (gas: 509430)
- [PASS] test_mode_2_erc165() (gas: 2469958)
- [PASS] test_revert_when_mode_disabled() (gas: 196871)
- [PASS] test_sudo() (gas: 221198)
- [PASS] testIntersect(uint48,uint48,uint48,uint48) (runs: 256, μ: 2378, ~: 2377)
- [PASS] test_callcode() (gas: 247855)
- [PASS] test_disable_mode() (gas: 173633)
- [PASS] test_erc721_receive() (gas: 1274094)
- [PASS] test_initialize() (gas: 162299)
- [PASS] test_initialize_twice() (gas: 20668)
- [PASS] test_set_default_validator() (gas: 348092)
- [PASS] test_set_execution() (gas: 398378)
- [PASS] test_validate_signature() (gas: 186500)
- [PASS] test_mode_2() (gas: 600217)

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Bastion Wallet. We performed our audit according to the procedure described above.

Some issues of High, Low and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Bastion Wallet smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Bastion Wallet smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Bastion Wallet to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2023

For

BASTION

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com