KGeN
Audit Report

Thu Feb 05 2026

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

KGeN Audit Report

1 Executive Summary

1.1 Project Information

Description	KGEN is an ERC-2771–based meta-transaction forwarder
Type	Game
Auditors	Alex,Bear Two
Timeline	Sun Feb 01 2026 - Thu Feb 05 2026
Languages	Solidity
Platform	BSC
Methods	Architecture Review, Unit Testing, Manual Review

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
WSS	wallet-service-signer.ts	b04db66198d8e1aa06c367c5e767f5f6cce31459
EAS	evm-adapter.service.ts	cfc86dc2ac372c5b34d64063c1c8f2f9fc179963
KGF	KGenForwarder.sol	dbe14bb95de18bf559e7fddd2bfeb58b0fbf6941

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	3	0
Centralization	1	1	0
Critical	0	0	0
Major	1	1	0
Medium	0	0	0
Minor	0	0	0
Informational	1	1	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by KGeN to identify any potential issues and vulnerabilities in the source code of the KGeN smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
KGF-1	Underlying `executeBatch` Can Bypass Permission Checks	Major	Fixed
KGF-2	Adding or Removing First Checks If Exists	Informational	Fixed
KGF-4	Centralization Risk	Centralization	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the KGeN Smart Contract :

Admin

- Admin can add trusted relayers through the `addTrustedRelayer()` function.
- Admin can remove trusted relayers through the `removeTrustedRelayer()` function.
- Admin can transfer ownership through the `transferOwnership()` function.
- Admin can accept ownership through the `acceptOwnership()` function.

Relayer

- Relayer can execute meta-transactions through the `execute()` function.
- Relayer can execute batches of meta-transactions through the `executeBatch()` function.

4 Findings

KGF-1 Underlying `executeBatch` Can Bypass Permission Checks

Severity: Major

Status: Fixed

Code Location:

KGenForwarder.sol#1

Descriptions:

The `KGenForwarder` contract implements access control on the `execute()` function by restricting calls to trusted relayers only. However, the contract fails to override the inherited `executeBatch()` function from `ERC2771Forwarder`, allowing any external actor to bypass the trusted relayer restriction and execute meta-transactions without authorization.

Suggestion:

It is recommended that override both `execute()` and `executeBatch()` functions with consistent access control.

Resolution:

The team adopted our advice and fixed this issue by overriding both `execute()` and `executeBatch()` functions with consistent access control.

KGF-2 Adding or Removing First Checks If Exists

Severity: Informational

Status: Fixed

Code Location:

KGenForwarder.sol#4224-4235

Descriptions:

The `addTrustedRelayer()` and `removeTrustedRelayer()` functions in `KGenForwarder` do not validate the current state of the relayer before modifying it. This allows redundant operations that emit misleading events and waste gas.

Suggestion:

It is recommended to add state validation checks.

Resolution:

The team adopted our advice and fixed this issue by adding state validation checks.

KGF-4 Centralization Risk

Severity: Centralization

Status: Fixed

Code Location:

KGenForwarder.sol#4224-4235

Descriptions:

The `KGenForwarder` contract has a centralization risk where the contract owner has unilateral and unrestricted control over the trusted relayer whitelist through `addTrustedRelayer()` and `removeTrustedRelayer()` functions.

Suggestion:

It is recommended to deploy with a multi-signature wallet as the owner to prevent single-key compromise.

Resolution:

The team fixed this issue by having a multisig wallet (0xF0E5D87C65483D11973CEaAA1f5353919aC1B0D0) set as a contract owner.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.