Security Review Report
NM-0302 Worldcoin - Grants4

(Sep 23, 2024)

Contents

1 Executive Summary

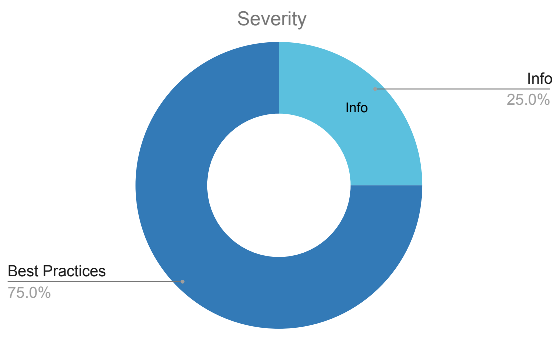
This document outlines the security review conducted by Nethermind Security for the Worldcoin Grants4 contracts. These new contracts will replace the previous ones starting from the 39th grant, distributed in August 2024. While grant reservation is no longer possible under the new system, users can still claim reservations made in the past for grants 21 to 38.

From Grant 39 through Grant 88, users are entitled to claim one `WLD` token drop per month. However, if a user is verified for the first time, they can claim the token drop from the previous month during their first month after joining. The new system also introduces a feature allowing authorized relayers to claim the drops on behalf of users immediately upon verification by the Orb. This enhancement addresses the previous delay, where users had to wait for their identity to be propagated across all relevant L2 networks before claiming their tokens.

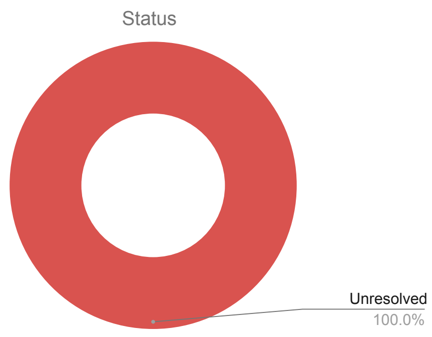
The audited code comprises of 369 lines of code written in the Solidity language, and it was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract.

Along this document, we report four points of attention, where all four are classified as `Best Practice`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.



(a)



(b)

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (0), **Best Practices** (4). **Distribution of status: Fixed** (4), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Aug 30, 2024
Response from Client	Regular responses during audit engagement
Final Report	Sep 23, 2024
Repository	worldcoin-grants-contracts
Commit (Audit)	107a05647490b0d0ec39623b4d400bb7a8f22519
Commit (Final)	e78b7dbc3873c92096c913aec86720f41e340fad
Documentation Assessment	Medium
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	WLDGrantPreGrant4_new.sol	28	11	39.3%	11	50
2	RecurringGrantDrop.sol	132	94	71.2%	58	284
3	WLDGrant.sol	106	26	24.5%	18	150
4	Grants4FirstBatch/Grants4FirstBatch.sol	103	42	40.8%	36	181
	Total	**369**	**173**	**46.9%**	**123**	**665**

3 Summary of Issues

	Finding	Severity	Update
1	Missing zero address checks in `Grants4FirstBatch`	Best Practices	Fixed
2	Third-party code should be imported and installed as an external dependency	Best Practices	Fixed
3	Unused code in `Grants4FirstBatch` contract	Best Practices	Fixed
4	Unused imports	Best Practices	Fixed

4 System Overview

The following section provides an overview of the system architecture and key contracts involved in the current Worldcoin Grants contracts (Grants4) version, which replaces the previous contracts starting from the 39th grant.

4.1 Contracts overview

The Worldcoin Grants4 system is primarily composed of the following contracts:

- **RecurringGrantDrop (new)**: This contract handles the distribution of grant tokens (WLD) for grants with a `grantId` of 39 or higher. The contract allows users to claim monthly WLD drops once the Orb verifies them. The contract supports the new feature where authorized relayers can claim drops on behalf of users immediately after their verification.
- **WLDGrant**: This contract holds the configuration of the Grants contracts. It is used to query the currently active grants and their token amounts. The `checkValidity(...)` function can be used to determine if a particular `grantId` is claimable.
- **Grants4FirstBatch**: Authorized Worldcoin relayers can call the `batch(...)` function from this contract to claim drops on behalf of the users. This action marks the `nullifierHash` as used in the `RecurringGrantDrop` contract to prevent users from claiming the same grant for the second time.
- **RecurringGrantDrop (old)**: Users can still claim their existing reservations by interacting with the old instance of the `RecurringGrantDrop` contract.

The grant claiming process in the Worldcoin Grants4 system differs depending on whether the grant is part of the new grant cycle (Grants 39-88) or involves reservations made in the previous system. The following section outlines the new grant-claiming process.

4.2 Grant claiming process

For the **new grants cycle**, starting from Grant 39 and continuing through to Grant 88, users are entitled to claim WLD token drops every month. the new system introduces a decreasing grant amount each month. However, the initial month's token amount starts from a higher base than in the previous system, providing a larger initial drop. The specific amounts are predetermined and stored in the `grantAmountsList` array in the `WLDGrant` contract. Whenever the user or authorized relayer claims a grant, the `RecurringGrantDrop` contract queries the `WLDGrant` contract to determine the exact token amount to be distributed for that particular month.

At any given time, two grants are active: the current month's grant and the previous month's grant. This allows users who missed claiming their tokens in the current month to still claim them in the following month.

For **old reservations** (Grants 21-38), users can still claim their previously reserved tokens by interacting with the old instance of the `RecurringGrantDrop` contract.

4.3 Immediate post-verification grant claims

One of the enhancements in the Worldcoin Grants4 protocol is the introduction of authorized relayers who can claim `WLD` token drops on behalf of users immediately after they are verified by the Orb. This feature is designed to improve the user experience by eliminating the delay that existed in the previous system.

Previously, users had to wait until their identity was propagated across all relevant Layer 2 (L2) networks before claiming their tokens, causing a delay between verification and the ability to claim. This created friction, as users could not claim their grant tokens immediately after being verified by the Orb, even though the Worldcoin backend already had their identity data. The delay occurred because this information had yet to be posted on-chain.

In the new system, authorized Worldcoin relayers can now step in to claim the tokens as soon as a user is verified. The relayer interacts with the `Grants4FirstBatch` contract, using the `batch(...)` function to claim the tokens. In turn, the `Grants4FirstBatch` contract calls the `setNullifierHash(...)` function in the `RecurringGrantDrop` contract to verify whether the nullifier hash for the particular grant claim has already been used. If the nullifier hash has not been used, the `setNullifierHash(...)` function sets it and transfers the WLD tokens to the user, completing the claim. If the nullifier hash has already been set, the function skips the grant claim, effectively preventing duplicate claims.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Best Practices] Missing zero address checks in `Grants4FirstBatch`

File(s): `Grants4FirstBatch.sol`

Description: In the `RecurringGrantDrop` contract, the `constructor(...)` and all the setter functions have `address(0)` checks. This is not the case for the `Grants4FirstBatch` contract. The `constructor(...)` doesn't have `address(0)` checks for its input parameters, nor do the setter functions `setAllowanceModule(...)`, `setWldToken(...)`, `setHolder(...)`, `setRecurringGrantDrop(...)`, `addCaller(...)`.

Recommendation(s): Consider adding zero address checks to maintain consistency throughout the codebase

Status: Fixed.

Update from the client: Addressed in e78b7db.

6.2 [Best Practices] Third-party code should be imported and installed as an external dependency

File(s): `Grants4FirstBatch.sol`

Description: The Grants contracts use third-party libraries and utilities, such as OpenZeppelin's `Ownable2Step` and `SafeERC20`. Unlike other contracts, the `Grants4FirstBatch` contract uses third-party utilities that are directly copied into the project's source files rather than imported and installed as the project's external dependencies.

This approach has implications for the security of the codebase. While libraries receive continuous updates to address any vulnerabilities or issues discovered, copying code internally makes tracking and applying necessary security updates more challenging. Additionally, the absence of the library version and the origin from where the code was copied makes it hard to review that particular code or function.

Below is the list of files where this issue is present:

- `src/Grants4FirstBatch/Context.sol`;
- `src/Grants4FirstBatch/Ownable.sol`;
- `src/Grants4FirstBatch/Ownable2Step.sol`;

Recommendation(s): To mitigate security risks and facilitate the review of the codebase, consider importing third-party code similarly to how it is done in other contracts such as `RecurringGrantDrop`. Alternatively, add comments about the code's origin and the version used.

Status: Fixed.

Update from the client: Addressed in e78b7db. In some places we decided not to use imports (for now) but added docs there.

6.3 [Best Practices] Unused code in `Grants4FirstBatch` contract

File(s): `Grants4FirstBatch.sol`

Description: The `Grants4FirstBatch.sol` file contains an `Enum` contract and a `GnosisSafe` interface. The `execTransactionFromModule(...)` function inside the `GnosisSafe` interface is never called and could be removed to keep the code clean and readable. This change makes the `Enum` contract obsolete as well.

Recommendation(s): Consider removing the unused code from the codebase or moving it to a separate file, from where it will be imported when needed.

Status: Fixed.

Update from the client: Addressed in e78b7db.

6.4 [Best Practices] Unused imports

File(s): `RecurringGrantDrop.sol`

Description: The `RecurringGrantDrop.sol` file imports OpenZeppelin's `ECDSA` library which is unused in the contract.

Recommendation(s): Consider removing unused import statement to keep the codebase clean and readable.

Status: Fixed.

Update from the client: Addressed in e78b7db.

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Worldcoin documentation

The **Worldcoin** team has provided documentation about their protocol in the form of in-line comments within the code.

8 Test Suite Evaluation

```
forge test
[] Compiling...
[] Compiling 38 files with Solc 0.8.19
[] Solc 0.8.19 finished in 1.10s
Compiler run successful!

Ran 3 tests for src/test/LaunchGrant.t.sol:MonthlyGrantTest
[PASS] testBiWeeklySwitch() (gas: 23639)
[PASS] testConsecutiveSpecialWeeks() (gas: 13644)
[PASS] testInitialLaunch2Weeks() (gas: 17927)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 964.88µs (1.05ms CPU time)

Ran 17 tests for src/test/WLDGrant.t.sol:WLDGrantTest
[PASS] testFuzz_getAmount_RevertsIfOutsideBounds(uint256) (runs: 256, : 9140, ~: 9141)
[PASS] test_checkValidity_August2024_grant39() (gas: 10715)
[PASS] test_checkValidity_February2026_grant56() (gas: 11172)
[PASS] test_checkValidity_Feburary2026_grant57() (gas: 11206)
[PASS] test_checkValidity_January2026_grant55() (gas: 11149)
[PASS] test_checkValidity_January2026_grant56() (gas: 11197)
[PASS] test_checkValidity_September2024_grant39() (gas: 11216)
[PASS] test_checkValidity_September2024_grant40() (gas: 11173)
[PASS] test_checkValidity_revertIfGrantIdLessThan21() (gas: 8641)
[PASS] test_checkValidity_revertIfGrantIdLessThan38ButGrant4LaunchHappened() (gas: 8961)
[PASS] test_getAmount_grant30() (gas: 8638)
[PASS] test_getAmount_grant38() (gas: 8646)
[PASS] test_getAmount_grant39() (gas: 7848)
[PASS] test_getAmount_grant40() (gas: 7870)
[PASS] test_getAmount_grant75() (gas: 7858)
[PASS] test_getAmount_grant88() (gas: 7848)
[PASS] test_getAmount_grant89() (gas: 8659)
Suite result: ok. 17 passed; 0 failed; 0 skipped; finished in 4.92ms (4.73ms CPU time)

Ran 6 tests for src/test/WLDGrantPreGrant4_new.t.sol:WLDGrantTest
[PASS] testFuzz_checkReservationValidityReverts(uint256) (runs: 256, : 11029, ~: 11029)
[PASS] testFuzz_checkValidityReverts(uint256) (runs: 256, : 9954, ~: 9955)
[PASS] test_checkReservation_grant37() (gas: 11367)
[PASS] test_checkReservation_grant38() (gas: 11249)
[PASS] test_checkValidity_grant38() (gas: 10110)
[PASS] test_checkValidity_revertsGreaterThanGrant38() (gas: 10793)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 13.96ms (24.30ms CPU time)

Ran 18 tests for src/test/RecurringGrantDrop.t.sol:RecurringGrantDropTest
[PASS] testFuzz_CannotAddNullifierHashBlockerIfNotManager(address) (runs: 256, : 14040, ~: 14040)
[PASS] test_CanAddNullifierHashBlockerIfManager() (gas: 35982)
[PASS] test_CanClaim39(uint256,uint256) (runs: 256, : 124272, ~: 124272)
[PASS] test_CanClaim39_2ndMonth(uint256,uint256) (runs: 256, : 124772, ~: 124772)
[PASS] test_CannotClaimAlreadyClaimedGrant_39(uint256,uint256) (runs: 256, : 128167, ~: 128167)
[PASS] test_CannotClaimAlreadyClaimedGrant_40(uint256,uint256) (runs: 256, : 128627, ~: 128627)
[PASS] test_CannotClaimFuture_22(uint256,uint256) (runs: 256, : 50117, ~: 50117)
[PASS] test_CannotClaimFuture_55(uint256,uint256) (runs: 256, : 53074, ~: 53074)
[PASS] test_CannotClaimPastGrant_21(uint256,uint256) (runs: 256, : 50199, ~: 50199)
[PASS] test_CannotClaimPastGrant_39(uint256,uint256) (runs: 256, : 53128, ~: 53128)
[PASS] test_CannotDoubleClaim(uint256,uint256) (runs: 256, : 130330, ~: 130330)
[PASS] test_CannotUpdateGrantIfNotManager(address) (runs: 256, : 2576782, ~: 2576782)
[PASS] test_UpdateGrant() (gas: 2579739)
[PASS] test_cannotClaimBelow39_21(uint256,uint256) (runs: 256, : 41242, ~: 41242)
[PASS] test_cannotClaimBelow39_38(uint256,uint256) (runs: 256, : 41297, ~: 41297)
[PASS] test_setNullifierHash_canBeCalledByAllowedBlocker() (gas: 60648)
[PASS] test_setNullifierHash_revertsIfCalledTwice() (gas: 62503)
[PASS] test_setNullifierHash_revertsIfNotAllowed(address) (runs: 256, : 11321, ~: 11321)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 45.10ms (269.92ms CPU time)

Ran 4 test suites in 163.69ms (64.94ms CPU time): 44 tests passed, 0 failed, 0 skipped (44 total tests)
```

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer