Security Review Report
NM-0160 WORLDCOIN

Allowance Module

NETHERMIND
SECURITY

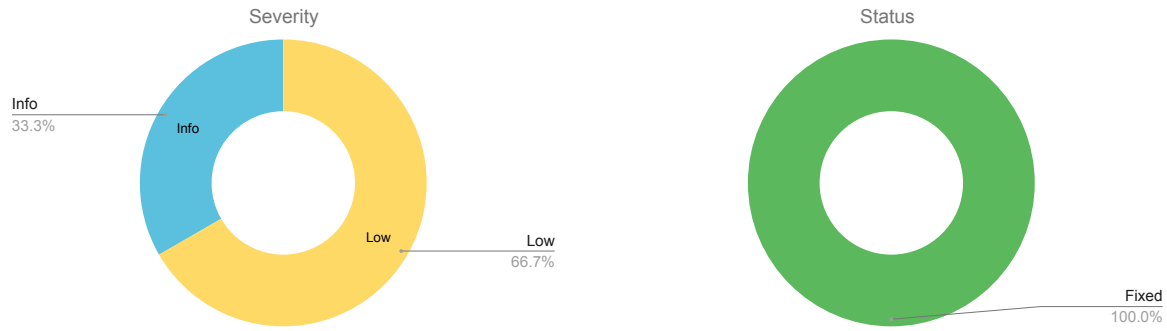
(Dec 01, 2023)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind on the Allowance Module containing 213 lines of code. The AllowanceModule contract is a fork from the repo safe-global/safe-modules with some modifications applied by the Worldcoin team. The contract is used to spend allowances set via Safe transactions. It also implements full control over allowances that are given to certain wallets. More specifically, the contract supports dynamic adjustments to allowances and periodic resets, providing a versatile solution for implementing complex allowance systems within the Safe environment.

Along the audit of this contract, we report 3 (**three**) points of attention, where two are classified as `Low` and one is classified as `Informational`. The issues are summarized in Fig. 1. **This document is organized as follows.** Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 discusses the risk rating methodology adopted for this audit. Section 5 presents the system overview. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 concludes the document.



(a) distribution of issues

(b) status of issues

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (2), **Undetermined** (0), **Informational** (1), **Best Practices** (0). **Distribution of status: Fixed** (3), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Oct. 23, 2023
Response from Client	Oct. 30, 2023
Final Report	Dec. 5, 2023
Methods	Manual Review, Automated Analysis
Repository	helper-contracts
Commit Hash (Audit)	abf345b152e484c0717975b41752b3657f4db393
Documentation	README.md
Documentation Assessment	Medium
Test Suite Assessment	Not Provided (one single file audit)

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	AllowanceModule.sol	213	54	25.4%	26	293
	Total	**213**	**54**	**25.4%**	**26**	**293**

3 Summary of Issues

	Finding	Severity	Update
1	Potential division by zero	Low	Fixed
2	The `executeAlllowanceTransfer(...)` function allows any user to do `zero` transfer on behalf of the `Safe`	Low	Fixed
3	Unnecessary overflow checks	Info	Fixed

4 Risk Rating Methodology

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

Likelihood is a measure of 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 the finding were to be exploited by an attacker. 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 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 formally pass to the client;

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.

5 System Overview

The **AllowanceModule** is a contract that extends the capabilities of Gnosis Safe, a multi-signature wallet. This module introduces a comprehensive system for managing token allowances, allowing the owner to delegate specific spending permissions to trusted parties. The contract employs a double-linked list to manage delegates, for addition and removal operations.

Detailed records of allowances are maintained for each token and delegate combination. These allowances include information on the permitted amount, amount spent, reset time, and initialization status. This level of granularity provides the owner with a nuanced control mechanism over fund utilization. Owners can set, reset, or delete allowances for individual delegates and tokens. The contract AllowanceModule has a number of important mappings and structs:

- `Mappings`: The contract maintains several mappings that keep track of allowances, tokens, and delegates related to a given Safe address. These mappings are integral to the system's structure.
- `Structs`: The contract defines two structures: `Delegate` and `Allowance`. They are used to store information about delegates and allowances, respectively.

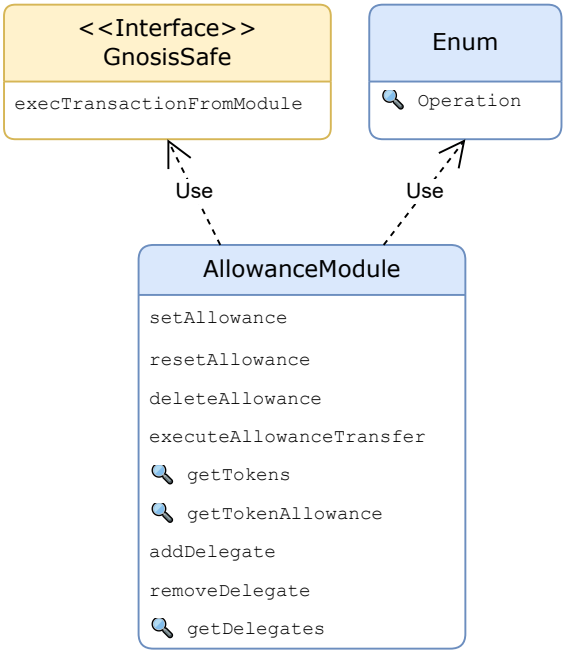


Fig. 2: Worldcoin - Structural Diagram of the `AllowanceModule` contract

Fig. 2 presents a structure diagram of the contract. As illustrated in it, the contract implements the following functions:

- `setAllowance(...)`: This function allows to update the allowance for a specific token. This operation can only be executed via a Safe transaction. It takes in parameters such as the delegate, token, allowance amount, and reset time.
- `executeAllowanceTransfer(...)`: It allows for the execution of a transfer based on an allowance. It takes parameters such as the Safe address, delegate, token, receiver, and amount. The function checks if the delegate is allowed to execute the transfer, deducts the pertinent amount from the allowance, and then executes the transfer.
- `resetAllowance(...)`: This function resets the allowance for a particular delegate and token addresses.
- `getAllowance(...)`: This function queries the allowance for a particular delegate and token. It returns the allowance amount and the reset time. The function applies resets in regular intervals.
- `addDelegate(...)` and `removeDelegate(...)`: These functions allow for the adding and the removal of a delegate.

6 Issues

6.1 [Low] Potential division by zero

File(s): AllowanceModule.sol

Description: The function `setAllowance(...)` in the contract `AllowanceModule` is used to update the allowance for a particular token. The function is presented below:

```
  function setAllowance(address delegate, address token, uint96 allowanceAmount, uint16 resetTimeMin, uint32
  ↪ resetBaseMin)
      public
  {
  ...
      uint32 currentMin = uint32(block.timestamp / 60);
      if (resetBaseMin > 0) {
          require(resetBaseMin <= currentMin, "resetBaseMin <= currentMin");
          ////////////////////////////////////////////////////
          // @audit resetTimeMin need to be greater than zero
          //        before updating allowance.lastResetMin
          ////////////////////////////////////////////////////
          allowance.lastResetMin = currentMin - ((currentMin - resetBaseMin) % resetTimeMin);
      } else if (allowance.lastResetMin == 0) {
          allowance.lastResetMin = currentMin;
      }
...
}
```

When the parameter `resetBaseMin` is greater than zero, and the following condition is satisfied, the `allowance.lastResetMin` is updated.

```
require(resetBaseMin <= currentMin, "resetBaseMin <= currentMin");
```

However, if `resetTimeMin == 0`, the transaction reverts. While this does not reveal a security issue, the scenario where the division by zero will cause the transaction to revert with an unexpected error message. Therefore, to avoid unexpected behaviors, consider returning ...

Recommendation(s): Apply a check for `resetTimeMin > 0` to avoid unexpected behaviors before updating `allowance.lastResetMin`.

Status: Fixed

Update from the client: Implemented

6.2 [Low] The `executeAlllowanceTransfer(...)` function allows any user to do `zero` transfer on behalf of the `Safe`

File(s): `AllowanceModule.sol`

Description: The `executeAllowanceTransfer(...)` function is designed to execute `transfers` from `Safe` contracts; these `transfers` can be executed by users with the required allowances. However, the current implementation allows the bypassing of checks when the transfer amount is set to `zero`

```
    function executeAllowanceTransfer(GnosisSafe safe, address token, address payable to, uint96 amount) public {
        // Get current state
        address delegate = msg.sender;
        Allowance memory allowance = getAllowance(address(safe), delegate, token);

        // Update state
        uint96 newSpent = allowance.spent + amount;
        // Check new spent amount and overflow
        // @audit - This requirement can be bypassed by any user using an amount equal to zero
        require(
            newSpent > allowance.spent && newSpent <= allowance.amount,
            "newSpent > allowance.spent && newSpent <= allowance.amount"
        );
        allowance.spent = newSpent;
        updateAllowance(address(safe), delegate, token, allowance);

        // Transfer token
        transfer(safe, token, to, amount);
        emit ExecuteAllowanceTransfer(address(safe), delegate, token, to, amount);
    }
```

The vulnerability arises because a `zero` transfer amount does not trigger the allowance checks, enabling any user, regardless of their allowance status, to execute calls on behalf of the `Safe` contracts. This could be potentially exploited by malicious actors to execute calls on behalf of the `Safe` contracts.

Recommendation(s): Disallow the transfer of `zero` amounts to avoid the mentioned case.

Status: Fixed

Update from the client: Implemented

6.3 [Info] Unnecessary overflow checks

File(s): AllowanceModule.sol

Description: The `executeAllowanceTransfer(...)` function includes a redundant overflow check.

```
    function executeAllowanceTransfer(...) public {
        ...
        // Update state
        uint96 newSpent = allowance.spent + amount;
        // Check new spent amount and overflow
        // @audit - Redundant overflow check
        require(
            newSpent > allowance.spent && newSpent <= allowance.amount,
            "newSpent > allowance.spent && newSpent <= allowance.amount"
        );
        ...
    }
```

Beginning with version `0.8.0`, the Solidity compiler automatically adds overflow checks for arithmetic operations. Consequently, the manual check in this function is redundant.

Recommendation(s): Consider removing the unnecessary operation.

Status: Fixed

Update from the client: Implemented

Update from Nethermind: The unnecessary overflow check was removed but this change added a new issue. As presented below, the condition that checks whether the new spent amount (`newSpent`) does not exceed the allowed amount (`allowance.amount`) has also been removed. In other words, the function allows to transfer more than the allowed amount.

```
-    // Check new spent amount and overflow
-    require(
-        newSpent > allowance.spent && newSpent <= allowance.amount,
-        "newSpent > allowance.spent && newSpent <= allowance.amount"
-    );
```

Update from the client: Fixed

7 Documentation Evaluation

Software documentation refers to the written or visual information describing software's functionality, architecture, design, and implementation. 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 the Allowance Module documentation

The Worldcoin team provided a README file describing how to build, run tests, etc. However, there is not any documentation to explain how the module should work and test implementation. During the audit process, the client remained in constant communication with the audit team for clarifications to tackle the lack of documentation.

8 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

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report 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 sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.