Security Review Report
NM-0272 - Game7

NETHERMIND
SECURITY

(August 16, 2024)

Contents

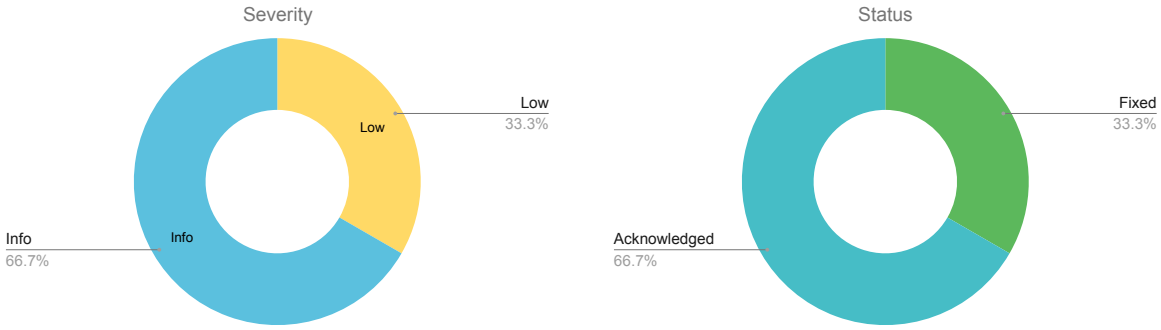
1 Executive Summary

This document outlines the security review conducted by Nethermind Security for the Game7 staker and token contract implementations. The Game7 protocol audit covers a permissionless staking protocol that allows deposits for native, ERC20, ERC721, and ERC1155 tokens where assets are locked for a certain duration of time. Rewards are then handled by other GameFi projects which will observe the state of this contract to determine which users have staked. Each stake is represented as an ERC721 contract, and depending on the pool configuration positions can be transferred to other users. The Game7 team has also included two token implementations in the audit scope, both based on the WETH9 implementation. One is an ERC20 token that will be used for the G7 protocol token, and the other is a wrapped-ether token that will be used to wrap the native asset on the Game7 L3 chain.

The audited code comprises of 471 lines of code written in the Solidity language. The Game7 team has provided comprehensive documentation including design documents and inline comments to explain the behavior of the protocol. Additionaly both the Nethermind and Game7 teams have been in communication to clarify all other questions regarding protocol design.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract. Along with this document, we report 3 points classified as Low.

Along this document, we report 3 points of attention, where 1 is classified as `Low`, and 5 are classified as `Info`.
The issues are summarized in Fig. 1.



(a) distribution of issues according to the severity

(b) distribution of issues according to the status

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

Summary of the Audit

Audit Type	Security Review
Initial Report	August 09, 2024
Final Report	August 16, 2024
Methods	Manual Review, Automated analysis
Repository	`G7DAO/protocol`
Commit Hash	`707b114560aac1afe08b7b40f7db4bf8286885cb`
Final Commit Hash	`24bc858f8a1616fc35c8eebfb0e614e4a7d7d2a2`
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	contracts/token/ERC20.sol	37	6	16.2%	13	56
2	contracts/token/WrappedNativeToken.sol	52	6	11.5%	17	75
3	contracts/interfaces/IERC20Inbox.sol	26	44	169.2%	5	75
4	contracts/interfaces/IERC20.sol	11	57	518.2%	8	76
5	contracts/staking/Staker.sol	345	51	14.8%	67	463
	Total	**471**	**164**	**34.8%**	**110**	**745**

3 Summary of Issues

	Finding	Severity	Update
1	Pool administrators can lock tokens using cooldown and lockup durations	Low	Acknowledged
2	`WrappedNativeToken` approvals can be frontrun	Info	Acknowledged
3	Missing reentrancy protections of `stakeNative` and `initiateUnstake`	Info	Fixed

4 System Overview

The Game7 protocol introduces the `Staker`, `ERC20`, and `WrappedNativeToken` contracts which form the bases of its Web3 components. Each contract is described as follows:

- **`Staker.sol`**: Permissionless staking contract, any users can create a staking pool to be used by others.
- **`ERC20`**: An ERC20 contract based on the WETH9 implementation, to be deployed as the G7 token in the future.
- **`wrappedNativeToken`**: A wrapped-ether implementation also based on WETH9, written in Solidity `0.8`.

4.1 Staking Contract

The staking contract supports deposits of native, ERC20, ERC721 and ERC1155 assets. Any user can create a staking pool and set configuration values for the token lock durations, asset type, and position transferability. Once a pool has been created any user is able to deposit into this pool. Each stake position is represented as an ERC721 asset which may be transferred, depending on the pool transferability options set by the pool creator. Pool creators are able to change the configuration after it has been created, and ownership of a pool can be transferred to the zero address to prevent future config changes.

Asset lock durations are broken down into two sections; "lockup" and "cooldown". When a stake begins the lockup period begins to count down, and once the lockup time has passed then the user must initiate an unstake, where the cooldown period will begin. Once the cooldown period has passed the user can finish the unstake process and recieve their assets.

4.2 Token Contracts

The token implementations are heavily inspired by the WETH9 token implementation, but are written in Solidity `0.8` which features overflow protection. The ERC20 contract is to be used for the G7DAO token at some point in the future, and the wrapped ether token will be used to wrap the native asset on the Game7 chain.

5 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 measures how likely an attacker will uncover and exploit the finding. 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 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.

6 Issues

6.1 [Low] Pool administrators can lock tokens using cooldown and lockup durations

File(s): Staker.sol

Description: Every pool has an administrator who is able to modify some aspects of the pool through the function `updatePoolConfiguration`. Specifically, the administrator is able to change the `transferability, lockupSeconds, and cooldownSeconds`. For the lockup and cooldown durations there are no upper limits on how long a user must wait, meaning it is possible for an administrator to effectively lock user tokens by setting the duration to an extremely long duration.

Also, if the `cooldownSeconds` is set to a value such that when added with a position's stake timestamp it exceeds $2^{256} - 1$, then it is possible to overflow when calling `initiateUnstake`, making it impossible to unstake tokens, as shown below:

```
function updatePoolConfiguration(
       uint256 poolID,
       bool changeTransferability,
       bool transferable,
       bool changeLockup,
       uint256 lockupSeconds,
       bool changeCooldown,
       uint256 cooldownSeconds
   ) external {
       StakingPool storage pool = Pools[poolID];
       if (msg.sender != pool.administrator) {
           revert NonAdministrator();
       }
       if (changeTransferability) {
           pool.transferable = transferable;
       }
       if (changeLockup) {
           pool.lockupSeconds = lockupSeconds;
       }
       if (changeCooldown) {
           pool.cooldownSeconds = cooldownSeconds;
       }
       emit StakingPoolConfigured(
           poolID,
           pool.administrator,
           pool.transferable,
           pool.lockupSeconds,
           pool.cooldownSeconds
       );
   }
```

Recommendation(s): Consider adding input validation when setting the `lockupSeconds` and `cooldownSeconds` to prevent these value from being unreasonably high. This validation should be added to both the `constructor` and `updatePoolConfigurations`.

Status: Acknowledged

Update from the client: We don't think this is a protocol-level concern. Applications which use the 'Staker' may want different types of protections on these values. For example, we are planning to have game contracts which default to max lockup but modify it atomically for players based on game state.

6.2 [Info] Missing Reentrancy protections of `stakeNative` and `initiateUnstake`

File(s):Staker.sol

Description: The functions stakeERC20, stakeERC721, stakeERC1155 and unstake use the modifier nonReentrant to protect against reentrancy where a given asset address may be malicious or give execution to a malicious contract. The functions stakeNative and initiateUnstake do not have such protections as they do not directly interact with an external contract. However, it is possible for these functions to be accessed within the non-reentrancy protected functions, for example unstake when certain assets are returned to the caller.

```
function unstake(uint256 positionTokenID) external nonReentrant {
    // ...

    if (pool.tokenType == NATIVE_TOKEN_TYPE) {
        payable(msg.sender).transfer(...);                  // Does call to recipient
    } else if (pool.tokenType == ERC20_TOKEN_TYPE) {
        IERC20(pool.tokenAddress).safeTransfer(...);
    } else if (pool.tokenType == ERC721_TOKEN_TYPE) {
        IERC721(pool.tokenAddress).safeTransferFrom(...);   // Does call to recipient
    } else if (pool.tokenType == ERC1155_TOKEN_TYPE) {
        IERC1155(pool.tokenAddress).safeTransferFrom(...);  // Does call to recipient
    }
}
```

As there are no reasons why a user would want to initiate an unstake or deposit native assets within an existing call to the contract, adding reentrancy protection to stakeNative and initiateUnstake will not affects users but will help reduce unexpected interactions with the protocol that may lead to incorrect behavior.

Recommendation(s): Consider adding reentrancy protection to the functions stakeNative and initiateUnstake

Status: Fixed

Update from the client: This has been fixed as of this PR: #31. Current commit hash: `4bed97c378310754d340c17b76dc8e6adc1cbfe1`.

6.3 [Info] `wrappedNativeToken` approvals can be frontrun

File(s): `WrappedNativeToken.sol`

Description: The WrappedNativeToken contract was built based on the WETH9 contract. It is a known issue that WETH9::approve(...) function is susceptible to frontrunning. In this implementation, the way that an approver address can reduce the allowance of a spender address is by calling the approve function again with a lower allowance value.

The spender could frontrun the approve(...) call in order to spend the current allowance before getting approved again for a new allowance. Doing so, the spender is able to spend allowance 1 + allowance 2, but this has always been the case with WETH9.

Recommendation(s): This allowance behavior is known, the EIP20 proposal as well as the Openzeppelin ERC20 implementation do not handle approval frontrunning as the probability of malicious activity is extremely low. Given that the token implementation is following `WETH9`, leaving the `WrappedNativeToken` logic as is will keep it closer to the source implementation. The token approval behavior can be left as is, or the functions `increaseAllowance` and `decreaseAllowance` can be added to help manage approvals.

Status: Acknowledged

Update from the client: Thank you for bringing this to our attention. Our security scanning tool had brought it to our attention before, as well. We would prefer to keep our implementation - it is simple and close to the WETH9 source. We consider the probability of an exploit here to be very low, especially given that many use cases for the wrapped token are as an atomic intermediary between the native token and ERC20 functionality on smart contracts.

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 Game7 Protocol documentation

The **Game7 Protocol** team has provided documentation about their protocol through their github documentation which includes the execution flow of the Staker contract
Additionally, the **Game7 Protocol** team was available to address any questions or concerns from the Nethermind Security team.

8 Test Suite Evaluation

8.1 Compilation Output

```
> npx hardhat compile
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
 ↪  "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for
 ↪  non-open-source code. Please see https://spdx.org for more information.
--> contracts/mock/tokens.sol


Warning: Source file does not specify required compiler version! Consider adding "pragma solidity ^0.8.24;"
--> contracts/mock/tokens.sol


Generating typings for: 39 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 118 typings!
Compiled 35 Solidity files successfully (evm target: paris).
```

8.2 Tests Output

8.2.1 Staker and ERC20 tests

```
> npx hardhat test

ERC20
    contracts/token/ERC20.sol:ERC20
       total supply: returns the total token value
       has a name
       has a symbol
       has 18 decimals
      balanceOf
         returns zero when the requested account has no tokens
         returns the total token value when the requested account has some tokens
      transfer
        when the recipient is not the zero address
           reverts when the sender does not have enough balance
          when the sender transfers all balance
             transfers the requested value
             emits a transfer event
          when the sender transfers zero tokens
             transfers the requested value
             emits a transfer event
      transfer from
        when the token owner is not the zero address
          when the recipient is not the zero address
            when the spender has enough allowance
               reverts when the token owner does not have enough balance
              when the token owner has enough balance
                 transfers the requested value
                 decreases the spender allowance
                 emits a transfer event
                 does not emit an approval event
            when the spender does not have enough allowance
               reverts when the token owner has enough balance
               reverts when the token owner does not have enough balance
      approve
        when the spender is not the zero address
          when the sender has enough balance
             emits an approval event
             approves the requested value when there was no approved value before
             approves the requested value and replaces the previous one when the spender had an approved value
          when the sender does not have enough balance
             emits an approval event
             approves the requested value when there was no approved value before
             approves the requested value and replaces the previous one when the spender had an approved value
      transfer
        when the recipient is not the zero address
           reverts when the sender does not have enough balance
          when the sender transfers all balance
             transfers the requested value
             emits a transfer event
          when the sender transfers zero tokens
             transfers the requested value
             emits a transfer event
      approve
         reverts when the spender has insufficient allowance
        when the spender is not the zero address
          when the sender has enough balance
             emits an approval event
             approves the requested value when there was no approved value before
             approves the requested value and replaces the previous one when the spender had an approved value
          when the sender does not have enough balance
             emits an approval event
             approves the requested value when there was no approved value before
             approves the requested value and replaces the previous one when the spender had an approved value
```

```
WrappedNativeToken
  WrappedNativeToken
     should not deploy with value
     total supply: returns the total token value
     has a name
     has a symbol
     has 18 decimals
    balanceOf
       returns zero when the requested account has no tokens
       returns the total token value when the requested account has some tokens
    transfer
      when the recipient is not the zero address
         reverts when the sender does not have enough balance
        when the sender transfers all balance
           transfers the requested value
           emits a transfer event
        when the sender transfers zero tokens
           transfers the requested value
           emits a transfer event
    transfer from
      when the token owner is not the zero address
        when the recipient is not the zero address
          when the spender has enough allowance
             reverts when the token owner does not have enough balance
            when the token owner has enough balance
               transfers the requested value
               decreases the spender allowance
               emits a transfer event
               does not emit an approval event
          when the spender does not have enough allowance
             reverts when the token owner has enough balance
             reverts when the token owner does not have enough balance
    approve
      when the spender is not the zero address
        when the sender has enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
        when the sender does not have enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
    transfer
      when the recipient is not the zero address
         reverts when the sender does not have enough balance
        when the sender transfers all balance
           transfers the requested value
           emits a transfer event
        when the sender transfers zero tokens
           transfers the requested value
           emits a transfer event
    approve
       reverts when the spender has insufficient allowance
      when the spender is not the zero address
        when the sender has enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
        when the sender does not have enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
    deposit
       reverts when the value is zero
       should deposit
      withdraw
         reverts when the value is zero
         reverts when the value is greater than the balance
         should withdraw
```

```
WrappedNativeToken
  WrappedNativeToken
     should not deploy with value
     total supply: returns the total token value
     has a name
     has a symbol
     has 18 decimals
    balanceOf
       returns zero when the requested account has no tokens
       returns the total token value when the requested account has some tokens
    transfer
      when the recipient is not the zero address
         reverts when the sender does not have enough balance
        when the sender transfers all balance
           transfers the requested value
           emits a transfer event
        when the sender transfers zero tokens
           transfers the requested value
           emits a transfer event
    transfer from
      when the token owner is not the zero address
        when the recipient is not the zero address
          when the spender has enough allowance
             reverts when the token owner does not have enough balance
            when the token owner has enough balance
               transfers the requested value
               decreases the spender allowance
               emits a transfer event
               does not emit an approval event
          when the spender does not have enough allowance
             reverts when the token owner has enough balance
             reverts when the token owner does not have enough balance
    approve
      when the spender is not the zero address
        when the sender has enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
        when the sender does not have enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
    transfer
      when the recipient is not the zero address
         reverts when the sender does not have enough balance
        when the sender transfers all balance
           transfers the requested value
           emits a transfer event
        when the sender transfers zero tokens
           transfers the requested value
           emits a transfer event
    approve
       reverts when the spender has insufficient allowance
      when the spender is not the zero address
        when the sender has enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
        when the sender does not have enough balance
           emits an approval event
           approves the requested value when there was no approved value before
           approves the requested value and replaces the previous one when the spender had an approved value
    deposit
       reverts when the value is zero
       should deposit
      withdraw
         reverts when the value is zero
         reverts when the value is greater than the balance
         should withdraw
```

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 procehttps://www.overleaf.com/project/65c0e737f41a29601bda5c48ss, 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.