Security Review Report
NM-0622 Veraswap

NETHERMIND SECURITY

(Sep 16, 2025)

Contents

1 Executive Summary 2

2 Audited Files 3

3 Summary of Issues 3

4 System Overview 4
4.1 Universal Router's Proposed Addition of `CALL_TARGET` Command (PR #449) 4
4.2 Sweeper Contracts 4
4.3 Bridging and Swapping 4

5 Risk Rating Methodology 6

6 Issues 7
6.0.1 [Medium] Failed bridged message cannot be canceled 7
6.0.2 [Medium] Incorrect refund address 7
6.0.3 [Low] Lack of target restriction in UniversalRouter 8
6.0.4 [Low] transfer(...) used for native currency transfer 8
6.0.5 [Low] Inconsistent application of `msg.value` in `OwnableSignatureExecutor` 8
6.0.6 [Low] Lack of protection during interaction with Position Managers 10
6.0.7 [Info] Potential reentrancy during interaction with the bridge 11
6.0.8 [Info] Transfers of funds are not checked to be successful 11

7 Documentation Evaluation 13

8 Test Suite Evaluation 14
8.1 Compilation Output 14
8.2 Tests Output 14

9 About Nethermind 18

1 Executive Summary

This document presents the security review performed by Nethermind Security for **Veraswap**. Veraswap is an innovative project that aims to provide the best cross-chain swapping experience. Users who swap with Veraswap do not have to worry if the token is on their chain, since swapping and bridging are abstracted from the user and made convenient. To achieve that, Veraswap utilizes `Uniswap Universal Router` commands, `Hyperlane` cross-chain messaging, and `ERC7579` accounts. Veraswap's swapping logic is designed in a modular way, which allows for easy introduction of new features or changing parts of a mechanism to an alternative implementation. Such a design is not only easier to maintain and secure, but also invites contributors who wish to propose improvements to the protocol.

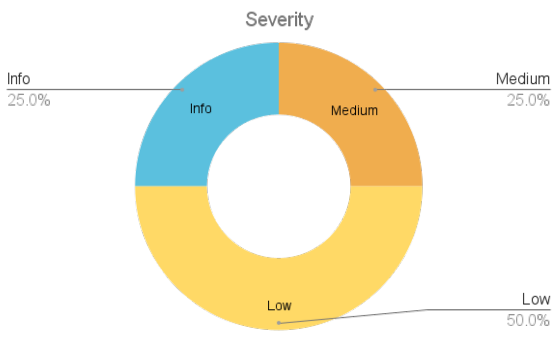
The security review focused on the smart contracts that are the core of the on-chain part of the application. Those contracts are going to be deployed on the chosen blockchains.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases. **Along this document, we report** 9 points of attention, where two are classified as `Medium`, four are classified as `Low` and two are classified as `Informational` or `Best Practice`. The issues are summarized in Fig. 1.

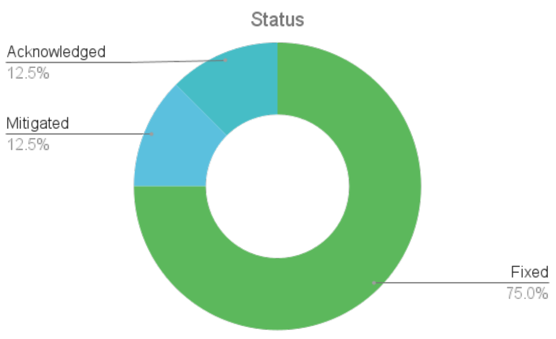
Remarks about the Veraswap tests

Veraswap supplied a substantial amount of tests covering many scenarios. However, those tests seem to be too generic and do not cover in-depth all important interactions and scenarios. The Veraswap protocol interacts with external contracts and bridges, therefore we strongly advise to create a stronger test suite that would cover most important execution paths and include edge cases.

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** (2), **Low** (4), **Undetermined** (0), **Informational** (2), **Best Practices** (0).
Distribution of status: Fixed (6), **Acknowledged** (1), **Mitigated** (1), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Sep 2, 2025
Response from Client	Regular responses during audit engagement
Final Report	Sep 16, 2025
Repository	`owlprotocol/veraswap` `Uniswap/universal-router`
Commit - veraswap	`c98da9037f4b42b0ae92375cc82f647858ee6f06`
Commit - universal-router	`20c847b94eeaf52c5fd12653a31ee80cac24a0dc`
Final Commit - veraswap	`d9327ea3aa4eb87fbe610df61fe7f5f1c5311607`
Final Commit - universal-router	`fb6900abc222a6d9bdae624d5d36dac5d2ee0cc3`
Documentation	Meetings and code comments
Documentation Assessment	Medium
Test Suite Assessment	Low

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	OwnableSignatureExecutor.sol	58	36	62.1%	14	108
2	ERC7579ExecutorRouter.sol	128	64	50.0%	20	211
3	StargateBridgeSweep.sol	83	37	44.6%	30	150
4	HypTokenRouterSweep.sol	34	28	82.4%	9	71
5	PR #449	43	20	46.5%	17	72
	Total	**346**	**185**	**53.5%**	**90**	**621**

3 Summary of Issues

	Finding	Severity	Update
1	Failed bridged message can't be cancelled	Medium	Mitigated
2	Incorrect refund address	Medium	Fixed
3	Lack of target restriction in UniversalRouter	Low	Acknowledged
4	transfer(...) used for native currency transfer	Low	Fixed
5	Inconsistent application of `msg.value` in `OwnableSignatureExecutor`	Low	Fixed
6	Lack of protection during interaction with Position Managers	Low	Fixed
7	Potential reentrancy during interaction with the bridge	Info	Fixed
8	Transfers of funds are not checked to be successful	Info	Fixed

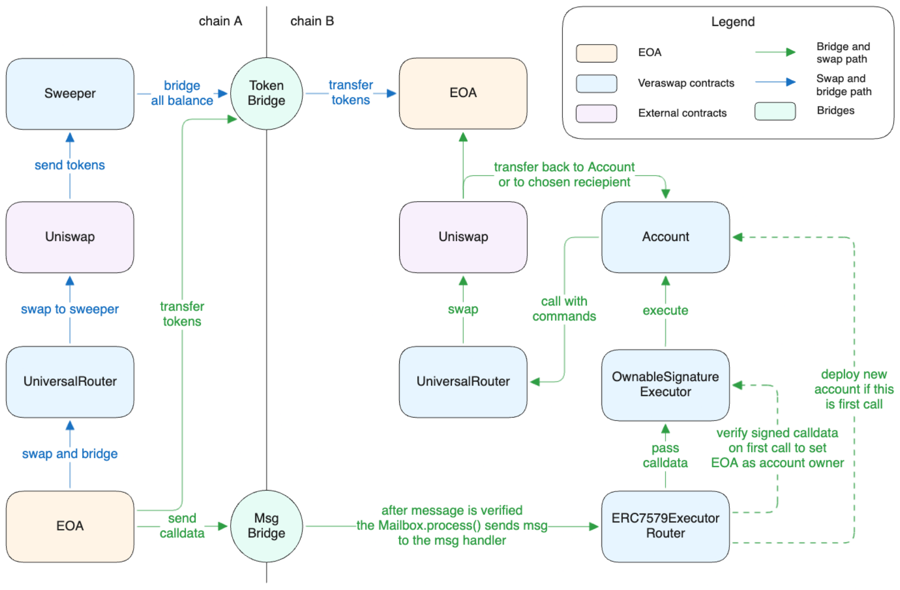


Fig. 2: Most important interaction for swap + bridge and bridge + swap paths paths

4 System Overview

The `Veraswap` protocol abstracts complicated cross-chain interactions to allow users to perform swaps regardless of which chain the token exists on. Veraswap allows users to perform four distinct actions: (1) Swap (2) Bridge (3) Swap and Bridge (4) Bridge and Swap. Action (3) is performed by utilizing the `UniversalRouter`, which is modified by Veraswap team and contains the additional `CALL_TARGET` command to interact with the token bridge contracts. The action (4) requires more complex interaction, since it performs a swap action on the remote chain. Veraswap designed the mechanism so that the user doesn't need to know that tokens are bridged. Below, the bridging of tokens and remote swap execution are described in more detail.

4.1 Universal Router's Proposed Addition of `CALL_TARGET` Command (PR #449)

`Veraswap's` PR #449 proposes extending Uniswap's `UniversalRouter` with a new command `CALL_TARGET` with command ID `0x22` that allows the router to invoke arbitrary external contracts at any point during execution, improving interoperability with protocols not covered by the existing model. The command's inputs are ABI-encoded as `(address target, uint256 value, bytes data)` and are decoded via a lightweight `CalldataCallTargetDecoder.sol` contract also introduced in the proposal. `Dispatcher.sol` is amended so that when it encounters `0x22` it performs a low-level `call` to `target` with the provided `value` and `data`, while explicitly rejecting `target == Permit2` to prevent bypassing the router's curated Permit2 surfaces. As with all Universal Router commands, bit 7 of the command byte independently controls allow-revert semantics, preserving the UniversalRouter's design behaviour.

4.2 Sweeper Contracts

`Veraswap's` two sweeper contracts, `HypTokenRouterSweep.sol` and `StargateBridgeSweep.sol` act as post-swap executors. After a Uniswap trade, the sweeper contract utilizes all balance of the output token and initiates a cross-chain transfer to the designated recipient, using either Hyperlane or Stargate to perform the bridge call.

4.3 Bridging and Swapping

The "Bridge and swap" action is not initiated through the `UniversalRouter`. The `EOA` account performs two calls during this action. The first one is the transfer of the source token to the destination chain. It is performed by `Hyperlane` or `Stargate` standard token bridge. Those tokens are sent to a deterministically calculated address of the remote account. During a second call, the user's `EOA` account calls the `ERC7579ExecutorRouter.callRemote(...)`, which triggers the cross-chain message mechanism. On the destination chain, the `ERC7579ExecutorRouter.handle(...)` handles the received message. If it is the first call to this chain, the account factory deploys the `ERC7579` account, for which the `zerodevapp Kernel` account implementation is used. The `initData`, from the cross-chain message, is used for the account initialization. During the initialization, the `OwnableSignatureExecutor` executor module is installed. Note that `initData` can't be tampered with, since the account's address depends on it, so any change would deploy the account at another address. Next the `ERC7579ExecutorRouter.handle(...)` calls the `OwnableSignatureExecutor.executeOnOwnedAccountWithSignature(...)` with according calldata signed by the `EOA`. This calldata during the first call should trigger the `ERC7579 account` to call `ERC7579ExecutorRouter.setAccountOwners(...)` and register the `EOA` as an owner with the corresponding remote router address and origin domain. With this setup, every next cross-chain call can be done without any additional signatures, but just with a direct call to the `ERC7579 account` from Router through `OwnableSignatureExecutor.executeOnOwnedAccount(...)`. When the bridged tokens arrive at the account, the Executor contract triggers the account to execute the swap. The swap is performed by the `UniversalRouter`. The user can also perform remote calls on the account from a new chain by just signing calldata and executing it through `OwnableSignatureExecutor.executeOnOwnedAccountWithSignature(...)` to register a new remote router and origin domain in the local `ERC7579ExecutorRouter` router.

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

6.0.1 [Medium] Failed bridged message cannot be canceled

File(s): ERC7579ExecutorRouter.sol

Description: The Hyperlane allows for replaying failed messages by any relayer by calling `process()`. However, there is no option to cancel the message. A user who uses Veraswap may perform the bridge and swap action, which bridges the tokens to the destination chain and then bridges the calldata, which is used by the ERC7579 account to perform the swap. If the swap fails (e.g., due to slippage), the user may want to bridge tokens back to the origin chain by submitting another cross-chain call. However, the first swap call, which failed, is still present in the `Mailbox` and can be executed in the future. If the user were to use Veraswap to perform another swap using the same source token, the first failed swap may be executed by anyone who calls `Mailbox.process()` with the failed message. As a result, the user's funds would be used for the first swap and not the newest swap.

Recommendation(s): Since Hyperlane does not allow for cancellation of verified messages that failed on execution, solving the described issue requires handling it on the Veraswap side. One possible scenario would involve stopping the use of the account after bridging back the tokens if the swap fails and the user does not want to execute it. Then on the next call, the new account should be deployed. Another solution could be implementing the registry in the `ERC7579ExecutorRouter`, which would allow the user to invalidate chosen calldata, so that when it is processed in the `Mailbox`, the `ERC7579ExecutorRouter` would not perform any call to the executor.

Status: Mitigated

Update from the client: Reduced execution deadline to 10 minutes. Messages cannot be cancelled, but expire by themselves once their deadline is passed. PR

6.0.2 [Medium] Incorrect refund address

File(s): StargateBridgeSweep.sol, HypTokenRouterSweep.sol

Description: Sweepers do not specify the sender EOA address as the fee surplus or swap leftover as the recipient. In the `StargateBridgeSweep`, the `_refundAddress` is specified as the bridge transfer recipient:

```
function bridgeAllETH(address payable to, BridgeParams calldata bridgeParams) external {
    ...
    pool.sendToken{value: balance}({_sendParam: sendParam, _fee: fee, _refundAddress: bridgeParams.recipient});

    // Refund user with any excess native tokens
    uint256 leftoverBalance = address(this).balance;
    if (leftoverBalance > 0) {  payable(bridgeParams.recipient).transfer(leftoverBalance);
    }
}
```

The `bridgeParams.recipient` can be any address to which the user wants to bridge funds, which does not have to be the user's controlled EOA, but the fee surplus and the ETH leftover will be sent to this address.

In the `HypTokenRouterSweep`, there are two functions, where only one allows for defining the EOA sender in the hookMetadata as the fee surplus recipient:

```
// @audit: this function would return the fee surplus to the sweeper
function transferRemote(address router, uint32 destination, bytes32 recipient, uint256 bridgePayment)
// @audit: this function allows for defining the EOA as the refund recipient in the hookMetadata
function transferRemote(
    address router,
    uint32 destination,
    bytes32 recipient,
    uint256 bridgePayment,
    bytes calldata hookMetadata,
    address hook
)
```

Recommendation(s): Consider always defining the EOA as the fee surplus and the ETH leftover as a recipient. This can be done by setting the user's EOA as the `_refundAddress` in the `StargateBridgeSweep` and defining the EOA address in the hookMetadata with the corresponding hook in the `HypTokenRouterSweep`. Moreover, ensure that the calls made directly from the EOA account during Bridge & Swap also define the EOA as the refund address.

Status: Fixed

Update from the client: Resolved for Stargate, acknowledged for Hyperlane, although not launch-critical, as refund is limited to excess fees

`HypTokenRouterSweep.sol`: As mentioned, refundAddress can be encoded in the hookMetadata. The Veraswap frontend does not do this yet, but this is a marginal issue considering the refund amount. Can be fixed client-side. No Solidity changes necessary.

StargateBridgeSweep.sol: Added a refund address parameter bridgeAllETH and bridgeAllToken, commit.

6.0.3 [Low] Lack of target restriction in UniversalRouter

File(s): Dispatcher.sol

Description: The CALL_TARGET command allows for arbitrary calls to any address, excluding the PERMIT2. This allows malicious users to use the UniversalRouter contract to perform transactions that are against the law. This may include interactions with blacklisted protocols, sending funds to criminals, or using the UniversalRouter as part of the exploit. Since the UniversalRouter is a core part of the Veraswap protocol, such transactions may involve legal or reputation issues. Note that no such scenario has been registered yet; therefore, the probability of such an event is considered minor.

Recommendation(s): Consider introducing a mapping of allowed target addresses to minimize the legal and reputation risk.

Status: Acknowledged

Update from the client: This is an accepted risk and is not much different from having the UniversalRouter interact with a malicious Uniswap pool. The entire purpose of the CALL_TARGET command is to enable open modularity, and as such, any allowlist would defeat its purpose. The Veraswap UI only uses the CALL_TARGET command to interact with our audited contracts (eg, HyperlaneSweeper, StargateSweeper).

The changes to the UniversalRouter are released under an open-source license and deployed on-chain as public good infrastructure on which we have no control over post-deployment. Developers are responsible for using the CALL_TARGET command responsibly in their projects.

6.0.4 [Low] transfer(...) used for native currency transfer

File(s): StargateBridgeSweep.sol

Description: The StargateBridgeSweep contract in the bridgeAllETH(...) transfers any leftover ETH to the specified recipient. However, the ETH transfer is performed with the .transfer(...) function. This function limits the passed gas to 2300 weis. While it was designed to protect against reentrancy, it is not a recommended way of transferring ETH, since the receiver address may need more gas to handle such a transfer (e.g., wallets) or the gas cost for specific opcodes may be changed in the future. Please refer to this article for more in-depth information. Note that while .transfer(...) was designed to protect against re-entrancy, those protections should be applied by the CEI pattern or the non-reentrant modifiers.

Recommendation(s): Consider using .call(...) for native currency transfers.

Status: Fixed

Update from the client: Changed to using CurrencyLibrary in the base Sweep.sol contract that implements shared behaviour between StargateBridgeSweep.sol and HyperlaneSweep.sol. The refund logic is now implemented in the internal _sweep() function of Sweep.sol. Native token refund logic has also been added to the bridgeAllToken function if the user sends too many native tokens to the sweeper.

Commit (diff and reference to the _sweep function)

6.0.5 [Low] Inconsistent application of msg.value in OwnableSignatureExecutor

File(s): OwnableSignatureExecutor

Description: The executeBatchOnOwnedAccountWithSignature (...) function in the OwnableSignatureExecutor contract executes the batch of transactions on the owned account using msg.value as:

```
function executeBatchOnOwnedAccountWithSignature(
        SignatureExecutionLib.SignatureExecution calldata signatureExecution,
        bytes calldata signature
    ) external payable {
        // ...
        // execute the batch of transactions on the owned account
        IERC7579Account(signatureExecution.account).executeFromExecutor{value: msg.value}( //@audit signatureExecution
         ↪ value, why msg.value directly.
            ModeLib.encodeSimpleBatch(),
            signatureExecution.callData
        );
    }
```

While executeOnOwnedAccountWithSignature (...) function is implemented as:

```
    function executeOnOwnedAccountWithSignature(
        SignatureExecutionLib.SignatureExecution calldata signatureExecution,
        bytes calldata signature
    ) external payable {
        // ....
        IERC7579Account(signatureExecution.account).executeFromExecutor{value: signatureExecution.value}(
            ModeLib.encodeSimpleSingle(),
            signatureExecution.callData
        );
    }
```

The `executeOnOwnedAccountWithSignature (...)` function uses the signed `msg.value` as `value: signatureExecution.value` while the `executeBatchOnOwnedAccountWithSignature (...)` function uses the caller's passed `msg.value` as `value: msg.value`.

This inconsistency leads to a varied amount of native token, `msg.value` passed to the user's account. As a result, if the user wanted a specific amount of native token to be added to their account, this is not possible.

Additionally, given that the `executeBatchOnOwnedAccountWithSignature (...)` function is an external function, anyone can call this function directly with the right `signatureExecution` and `signature` data, but with an altered `msg.value`, which is different from the user-intended signed `msg.value`. This will successfully execute the target's calls without the intended `msg.value`, but it will consume the account's nonce, effectively taking away the ability of a user to forward a given native token amount to their account.

Recommendation(s): Consider using the `signatureExecution.value` as is done in `executeOnOwnedAccountWithSignature(...)` function.

Status: Fixed

Update from the client: We have implemented the fix and added comments regarding the purpose of the `signatureExecution.value` to clarify that it does not impact the value of the calls themselves but rather serves as a constraint on having a deposit to the smart account. The `signatureExecution.value` will be 0 for most use cases. Commit

6.0.6 [Low] Lack of protection during interaction with Position Managers

File(s): Dispatcher.sol

Description: The CALL_TARGET command allows for arbitrary calls to any address. This includes managing positions in Uniswap V3 and V4. The commands that manage positions in Universal Router apply restrictions in those scenarios:

```
function dispatch(bytes1 commandType, bytes calldata inputs) internal returns (bool success, bytes memory output) {
    uint256 command = uint8(commandType & Commands.COMMAND_TYPE_MASK);
    success = true;
    // 0x00 <= command < 0x21
    if (command < Commands.EXECUTE_SUB_PLAN) {
        // 0x00 <= command < 0x10
        ...
        } else {
            // 0x10 <= command < 0x21
            if (command == Commands.V4_SWAP) {
                // pass the calldata provided to V4SwapRouter._executeActions (defined in BaseActionsRouter)
                _executeActions(inputs);
               // This contract MUST be approved to spend the token since its going to be doing the call on the position
                ↪  manager
            } else if (command == Commands.V3_POSITION_MANAGER_PERMIT) {
                _checkV3PermitCall(inputs);
                (success, output) = address(V3_POSITION_MANAGER).call(inputs);
            } else if (command == Commands.V3_POSITION_MANAGER_CALL) {
               //@audit: check the caller and restrict actions
                _checkV3PositionManagerCall(inputs, msgSender());
                (success, output) = address(V3_POSITION_MANAGER).call(inputs);
            } else if (command == Commands.V4_INITIALIZE_POOL) {
                PoolKey calldata poolKey;
                uint160 sqrtPriceX96;
                assembly {
                    poolKey := inputs.offset
                    sqrtPriceX96 := calldataload(add(inputs.offset, 0xa0))
                }
                (success, output) =
                    address(poolManager).call(abi.encodeCall(IPoolManager.initialize, (poolKey, sqrtPriceX96)));
            } else if (command == Commands.V4_POSITION_MANAGER_CALL) {
                // should only call modifyLiquidities() to mint
               //@audit: restrict actions
                _checkV4PositionManagerCall(inputs);
                (success, output) = address(V4_POSITION_MANAGER).call{value: address(this).balance}(inputs);
            } else {
                // placeholder area for commands 0x15-0x20
                revert InvalidCommandType(command);
            }
        }
    ...
}
```

Those checks protect the users who were tricked into approving their position NFT to the Universal Router, which would allow malicious actors to perform token burns, decrease of liquidity, or collection of rewards. The CALL_TARGET does not perform those checks, leaving this scenario to be potentially exploited.

Recommendation(s): Consider excluding addresses for which the Universal Router has additional checks when using the CALL_TARGET command.

Status: Fixed

Update from the client: Fully excluded all hard-coded addresses that have dedicated commands for interactions. The attack surface is now significantly reduced. CALL_TARGET can now only be used with addresses that are "out of scope" of the original Universal Router. We've also added a detailed comment explaining where the immutable constants are inherited from.

```
// Call target cannot be one of the following addresses to reduce the attack surface and enforce proper use of commands
// when interacting with protocol specific addresses
// - address(self): to enforce use of EXECUTE_SUB_PLAN command for self re-entrancy
// - PaymentsImmutables.sol: to avoid arbitrary token transfers
// - UniswapImmutables.sol: to have clear protocol interactions with v2,v3 factories
// - MigratorImmutables.sol: to have clear protocol interactions with v3,v4 position managers
// - ImmutableState.sol (v4-periphery): to have clear protocol interactions with v4 pool actions
if (
    target == address(self) || target == address(WETH9) || target == address(PERMIT2)
        || target == address(UNISWAP_V2_FACTORY) || target == address(UNISWAP_V3_FACTORY)
        || target == address(V3_POSITION_MANAGER) || target == address(V4_POSITION_MANAGER)
        || target == address(poolManager)
) {
    revert CallTargetInvalid(target);
}
```

Commit

6.0.7 [Info] Potential reentrancy during interaction with the bridge

File(s): `StargateBridgeSweep.sol` & `HypTokenRouterSweep.sol`

Description: The Veraswap protocol interacts with bridges to transfer tokens and to execute swaps on remote chains. During interaction with the bridge, the fee is sent in native currency. If the fee sent is too large, then the bridge returns the surplus to the specified recipient. Such a recipient can be a contract that can execute any code during a callback with native currency.

Recommendation(s): Since the bridge interactions are not made mid-execution, we didn't identify any immediate risk. However, this behaviour should be documented and shared with the users. Additionally, the interactions with bridges should be performed as the last action of the Veraswap multicalls.

Status: Fixed

Update from the client: In our current architecture, reentrancy from bridge interactions poses no issue, though this could be a concern in the future. The UniversalRouter is unaffected as it already has reentrancy protection. The sweeper contracts are also unaffected, assuming only one token at a time is sent to the sweeper. This is the case in Veraswap, where users are only making single token pair swaps.

In a more complex scenario, the sweeper contracts could be affected. For example, a user might send the sweeper two tokens to bridge, and then call `transferRemote` on each of them. In this scenario, a `transferRemote` call for the first token, followed by a native token refund from the bridging protocol to a malicious recipient, could cause a reentrancy calling `transferRemote` on the second token and stealing the tokens the user had meant to bridge to themselves.

While not affecting Veraswap's current implementation, we fixed the reentrancy risk to the sweeper contracts by adding a TSTORE-based reentrancy lock similar to UniversalRouter to future-proof the sweeper modules. This was fixed below, where all sweepers now inherit from `Sweep.sol`, which inherits from `Lock.sol` that implements a TSTORE-based lock. All relevant functions are then protected using the `isNotLocked` modifier.

Commit

6.0.8 [Info] Transfers of funds are not checked to be successful

File(s): `Dispatcher.sol`

Description: The `UniversalRouter` contract's dispatcher logic handles a `CALL_TARGET` command, which is designed to execute arbitrary calls to external contracts. This is achieved using a low-level `call(...)`. Low-level calls like `.call` do not propagate reverts from the callee. Instead, they return a boolean value indicating whether the call succeeded or failed.

```
// ...
function dispatch(...) internal {
    // ...
    } else if (command == Commands.CALL_TARGET) {
        (address target, uint256 value, bytes calldata data) = CalldataCallTargetDecoder.decodeCallTarget(
                    inputs
                );
          if (target == address(PERMIT2)) revert CallTargetPermit2();

        // @audit The `success` boolean is not checked.
        (success, output) = payable(target).call{value: value}(data);
        // ...
    }
    // ...
}
```

The `UniversalRouter` command structure includes an `allowRevert` flag to control whether a command's failure reverts the entire transaction. However, it allows for failure during the tokens or native currency transfers. This differs from other commands, which utilize the `Payments` contract, which performs funds transfers with `safeTransferETH(...)` and `safeTransfer(...)` that require always successful transfers.

Recommendation(s): Consider properly documenting described behaviour.

Status: Fixed

Update from the client: Added comments detailing the `CALL_TARGET` command's behaviour, its similarity with other commands that return success, and its difference from commands that always revert, such as those that handle token transfers.

```
    // Call may fail without a revert if Commands.FLAG_ALLOW_REVERT is set
   // This behavior is similar to other commands that use .call (eg. V3_POSITION_MANAGER_CALL, V4_POSITION_MANAGER_CALL)
    // This behavior is different other commands that transfer ETH or tokens (eg. TRANSFER)
    (success, output) = payable(target).call{value: value}(data);
```

Commit

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 the Veraswap documentation

Veraswap supplied comprehensive documentation prior to the audit, including a detailed Notion document and GitHub repository. The team also conducted a thorough code walkthrough during the kick-off meeting. Furthermore, Veraswap promptly addressed all questions and concerns raised by the Nethermind Security team, offering valuable insights and a clear understanding of the project's technical details

8 Test Suite Evaluation

8.1 Compilation Output

```
pnpm build
  > @owlprotocol/veraswap@2.0.1 build
  > turbo run build
  turbo 2.4.4
  • Packages in scope: @owlprotocol/bloomfolio-app, @owlprotocol/veraswap-app, @owlprotocol/veraswap-sdk,
   ↪ @owlprotocol/veraswap-test-duster
  • Running build in 4 packages
  • Remote caching disabled
     Tasks:    4 successful, 4 total
    Cached:    4 cached, 4 total
      Time:    3.792s >>> FULL TURBO
```

8.2 Tests Output

```
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test: > @owlprotocol/veraswap-sdk@3.0.0 test
@owlprotocol/veraswap-sdk:test: > pnpm run test:ci
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test: > @owlprotocol/veraswap-sdk@3.0.0 test:ci
@owlprotocol/veraswap-sdk:test: > vitest --run
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test:  RUN  v1.6.1
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test: Loading NODE envvars
@owlprotocol/veraswap-sdk:test: Loading .env
stdout |

@owlprotocol/veraswap-sdk:test: Loading NODE envvars
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test: stdout | dotenvConfig
@owlprotocol/veraswap-sdk:test: Loading .env
@owlprotocol/veraswap-sdk:test:
  src/uniswap/quote/getUniswapRoute.test.ts (13)
@owlprotocol/veraswap-sdk:test:   src/uniswap/quote/MetaQuoter.test.ts (24)
@owlprotocol/veraswap-sdk:test:  ↓ src/index.test.ts (3) [skipped]
@owlprotocol/veraswap-sdk:test:   src/calls/getTransferRemoteWithKernelCalls.test.ts (4)
@owlprotocol/veraswap-sdk:test:   src/smartaccount/ERC7579ExecutorRouter.test.ts (2)
@owlprotocol/veraswap-sdk:test:  ↓ src/calls/getBridgeSwapWithKernelCalls.test.ts (1) [skipped]
@owlprotocol/veraswap-sdk:test:  ↓ src/basket/getBasketMint.test.ts (2) [skipped]
@owlprotocol/veraswap-sdk:test:   src/uniswap/quote/MixedRoutes.test.ts (2)
@owlprotocol/veraswap-sdk:test:   src/smartaccount/OwnableExecutor.test.ts (5)
@owlprotocol/veraswap-sdk:test:  ↓ src/basket/BasketFixedUnits.test.ts (3) [skipped]
@owlprotocol/veraswap-sdk:test:   src/calls/Permit2/getPermit2ApproveCalls.test.ts (3)
@owlprotocol/veraswap-sdk:test:  ↓ src/uniswap/getUniswapV4RouteMultichain.test.ts (8) [skipped]
@owlprotocol/veraswap-sdk:test:  ↓ src/uniswap/getRouteMultichain.test.ts (5) [skipped]
@owlprotocol/veraswap-sdk:test:   src/calls/getTransferRemoteWithFunderCalls.test.ts (1)
@owlprotocol/veraswap-sdk:test:   src/calls/Permit2/getPermit2PermitCalls.test.ts (2)
@owlprotocol/veraswap-sdk:test:  ↓ src/query/quote.test.tsx (2) [skipped]
@owlprotocol/veraswap-sdk:test:   src/basket/getBatchSwaps.test.ts (1)
@owlprotocol/veraswap-sdk:test:   src/calls/getOwnableExecutorAddOwnerCalls.test.ts (2)
@owlprotocol/veraswap-sdk:test:  ↓ src/calls/getTransferRemoteWithApproveCalls.test.ts (1) [skipped]
@owlprotocol/veraswap-sdk:test:   src/calls/ERC20/getERC20ApproveCalls.test.ts (2)
```

```
@owlprotocol/veraswap-sdk:test:   src/calls/Permit2/getPermit2TransferFromCalls.test.ts (1)
@owlprotocol/veraswap-sdk:test:  ↓ src/uniswap/getUniswapV4Route.test.ts (4) [skipped]
@owlprotocol/veraswap-sdk:test:   src/basket/getBasketSwaps.test.ts (1)
@owlprotocol/veraswap-sdk:test:  ↓ src/swap/uniswap.test.ts (1) [skipped]
@owlprotocol/veraswap-sdk:test:   src/calls/ERC20/getERC20TransferFromCalls.test.ts (1)
@owlprotocol/veraswap-sdk:test:  ↓ src/query/tokenData.test.tsx (2) [skipped]
@owlprotocol/veraswap-sdk:test:  ↓ src/basket/getBasketMintQuote.test.ts (2) [skipped]
@owlprotocol/veraswap-sdk:test:   src/query/stargateTokenQuote.test.ts (3) 3143ms
@owlprotocol/veraswap-sdk:test:   src/query/stargateETHQuote.test.ts (3) 2305ms
@owlprotocol/veraswap-sdk:test:  ↓ src/utils/getTransactionType.test.ts (0) [skipped]
@owlprotocol/veraswap-sdk:test:  ↓ src/constants/tokens.test.ts (1) [skipped]
@owlprotocol/veraswap-sdk:test:   src/basket/getBasketQuotes.test.ts (1)
@owlprotocol/veraswap-sdk:test:   src/test/constants.test.ts (1)
@owlprotocol/veraswap-sdk:test:   src/calls/getBalance.test.ts (1)
@owlprotocol/veraswap-sdk:test:   src/smartaccount/LibClone.test.ts (3)
@owlprotocol/veraswap-sdk:test:   src/constants/uniswap.test.ts (1)
@owlprotocol/veraswap-sdk:test:   src/constants/hyperlane.test.ts (1)
@owlprotocol/veraswap-sdk:test:   src/constants/kernel.test.ts (1)
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test:  Test Files  24 passed | 14 skipped (38)
@owlprotocol/veraswap-sdk:test:       Tests  75 passed | 39 skipped (114)
@owlprotocol/veraswap-sdk:test:    Start at  20:05:36
@owlprotocol/veraswap-sdk:test:    Duration  17.43s (transform 900ms, setup 1ms, collect 4.67s, tests 6.11s, environment
 ↪  0ms, prepare 81ms)
@owlprotocol/veraswap-sdk:test:
@owlprotocol/veraswap-sdk:test:


 Tasks:    5 successful, 5 total
Cached:    4 cached, 5 total
  Time:    27.548s

forge test
[] Compiling...
No files changed, compilation skipped

Ran 4 tests for test/V3QuoterTest.s.sol:V3QuoterTest
[PASS] testExactInput() (gas: 404622)
[PASS] testExactInputSingle() (gas: 230618)
[PASS] testExactOutput() (gas: 400248)
[PASS] testExactOutputSingle() (gas: 229660)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 11.32ms (1.65ms CPU time)

Ran 4 tests for test/V3MetaQuoterTest.s.sol:V3QuoterTest
[PASS] testExactInput() (gas: 435987)
[PASS] testExactInputSingle() (gas: 246524)
[PASS] testExactOutput() (gas: 431678)
[PASS] testExactOutputSingle() (gas: 245695)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 11.88ms (2.22ms CPU time)

Ran 1 test for test/V2QuoterTest.s.sol:V2QuoterTest
[PASS] testExactInputSingle_A_L2() (gas: 102746)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 12.55ms (101.96µs CPU time)

Ran 2 tests for test/MetaQuoter/MetaQuoterV3Test.s.sol:MetaQuoterV3Test
[PASS] testExactInputSingle_A_L3() (gas: 5465027)
[PASS] testExactOutputSingle_A_L3() (gas: 5463012)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 12.90ms (1.27ms CPU time)

Ran 6 tests for test/CommandsBuilder/CommandsBuilderV2Test.s.sol:CommandsBuilderV2Test
[PASS] test_V2_A_B() (gas: 3490686)
[PASS] test_V2_A_ETH() (gas: 3515926)
[PASS] test_V2_A_L2_B() (gas: 6904672)
[PASS] test_V2_A_WETH() (gas: 3488562)
[PASS] test_V2_ETH_A() (gas: 3507044)
[PASS] test_V2_WETH_A() (gas: 3495298)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 14.10ms (1.69ms CPU time)
```

```
Ran 6 tests for test/CommandsBuilder/CommandsBuilderV4Test.s.sol:CommandsBuilderV4Test
[PASS] test_V4_A_B() (gas: 596306)
[PASS] test_V4_A_ETH() (gas: 520588)
[PASS] test_V4_A_L4_B() (gas: 1049907)
[PASS] test_V4_A_WETH() (gas: 554413)
[PASS] test_V4_ETH_A() (gas: 524695)
[PASS] test_V4_WETH_A() (gas: 570392)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 16.51ms (4.37ms CPU time)

Ran 1 test for test/RouterFeeTest.s.sol:RouterFeeTest
[PASS] test_permit2_transfer_from() (gas: 59777)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.68ms (40.08µs CPU time)

Ran 8 tests for test/CommandsBuilder/CommandsBuilderV3V4Test.s.sol:CommandsBuilderTest
[PASS] test_V3V4_A_B_ETH() (gas: 5798059)
[PASS] test_V3V4_A_ETH_B() (gas: 5829562)
[PASS] test_V3V4_A_L4_B() (gas: 5873587)
[PASS] test_V3V4_ETH_B_A() (gas: 5890451)
[PASS] test_V4V3_A_B_ETH() (gas: 5891355)
[PASS] test_V4V3_A_ETH_B() (gas: 5826599)
[PASS] test_V4V3_A_L4_B() (gas: 5872356)
[PASS] test_V4V3_ETH_B_A() (gas: 5794967)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 16.87ms (8.70ms CPU time)

Ran 4 tests for test/V4MetaQuoterTest.s.sol:V4MetaQuoterTest
[PASS] testExactInput() (gas: 398768)
[PASS] testExactInputSingle() (gas: 225006)
[PASS] testExactOutput() (gas: 396267)
[PASS] testExactOutputSingle() (gas: 225213)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 17.04ms (2.86ms CPU time)

Ran 4 tests for test/V4QuoterTest.s.sol:V4QuoterTest
[PASS] testExactInput() (gas: 309121)
[PASS] testExactInputSingle() (gas: 182214)
[PASS] testExactOutput() (gas: 308657)
[PASS] testExactOutputSingle() (gas: 182992)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 5.43ms (1.56ms CPU time)

Ran 3 tests for test/LibCloneTest.s.sol:LibCloneTest
[PASS] testInitCodeERC1967() (gas: 3285)
[PASS] testInitCodeHashERC1967() (gas: 3549)
[PASS] testPredictDeterministicAddressERC1967() (gas: 3605)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 789.04µs (707.21µs CPU time)

Ran 6 tests for test/CommandsBuilder/CommandsBuilderV3Test.s.sol:CommandsBuilderV3Test
[PASS] test_V3_A_B() (gas: 5310984)
[PASS] test_V3_A_ETH() (gas: 5335993)
[PASS] test_V3_A_L3_B() (gas: 10536982)
[PASS] test_V3_A_WETH() (gas: 5308794)
[PASS] test_V3_ETH_A() (gas: 5325927)
[PASS] test_V3_WETH_A() (gas: 5309470)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 18.41ms (5.23ms CPU time)

Ran 8 tests for test/CommandsBuilder/CommandsBuilderV2V4Test.s.sol:CommandsBuilderTest
[PASS] test_V2V4_A_B_ETH() (gas: 3979633)
[PASS] test_V2V4_A_ETH_B() (gas: 4009639)
[PASS] test_V2V4_A_L4_B() (gas: 4055444)
[PASS] test_V2V4_ETH_B_A() (gas: 4070494)
[PASS] test_V4V2_A_B_ETH() (gas: 4077412)
[PASS] test_V4V2_A_ETH_B() (gas: 4004709)
[PASS] test_V4V2_A_L4_B() (gas: 4052474)
[PASS] test_V4V2_ETH_B_A() (gas: 3980793)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 18.96ms (8.35ms CPU time)

Ran 2 tests for test/MetaQuoter/MetaQuoterV2Test.s.sol:MetaQuoterV2Test
[PASS] testExactInputSingle_A_L2() (gas: 3576852)
[PASS] testExactOutputSingle_A_L2() (gas: 3575865)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 5.30ms (801.13µs CPU time)
```

```
Ran 8 tests for test/MetaQuoter/MetaQuoterV2V4Test.s.sol:MetaQuoterV2V4Test
[PASS] test_V2V4_A_B_ETH() (gas: 4188196)
[PASS] test_V2V4_A_ETH_B() (gas: 4221550)
[PASS] test_V2V4_A_L4_B() (gas: 4267823)
[PASS] test_V2V4_ETH_B_A() (gas: 4278441)
[PASS] test_V4V2_A_B_ETH() (gas: 4290703)
[PASS] test_V4V2_A_ETH_B() (gas: 4214013)
[PASS] test_V4V2_A_L4_B() (gas: 4257585)
[PASS] test_V4V2_ETH_B_A() (gas: 4194703)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 5.49ms (10.63ms CPU time)

Ran 8 tests for test/MetaQuoter/MetaQuoterV3V4Test.s.sol:MetaQuoterV3V4Test
[PASS] test_V3V4_A_B_ETH() (gas: 6074815)
[PASS] test_V3V4_A_ETH_B() (gas: 6109641)
[PASS] test_V3V4_A_L4_B() (gas: 6154166)
[PASS] test_V3V4_ETH_B_A() (gas: 6166611)
[PASS] test_V4V3_A_B_ETH() (gas: 6166936)
[PASS] test_V4V3_A_ETH_B() (gas: 6104169)
[PASS] test_V4V3_A_L4_B() (gas: 6154382)
[PASS] test_V4V3_ETH_B_A() (gas: 6071171)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 10.34ms (14.02ms CPU time)

Ran 8 tests for test/MetaQuoter/MetaQuoterV2V3Test.s.sol:MetaQuoterV2V3Test
[PASS] test_V2V3_A_B_ETH() (gas: 9029072)
[PASS] test_V2V3_A_ETH_B() (gas: 9014116)
[PASS] test_V2V3_A_L3_B() (gas: 9004625)
[PASS] test_V2V3_ETH_B_A() (gas: 9023039)
[PASS] test_V3V2_A_B_ETH() (gas: 9039209)
[PASS] test_V3V2_A_ETH_B() (gas: 9012448)
[PASS] test_V3V2_A_L3_B() (gas: 9014548)
[PASS] test_V3V2_ETH_B_A() (gas: 9034881)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 5.64ms (11.83ms CPU time)

Ran 10 tests for test/MetaQuoter/MetaQuoterV4Test.s.sol:MetaQuoterV4Test
[PASS] testExactInputSingle_A_ETH() (gas: 637918)
[PASS] testExactInputSingle_A_L4() (gas: 717717)
[PASS] testExactInputSingle_ETH_A() (gas: 646230)
[PASS] testExactInput_A_ETH_L4() (gas: 1199680)
[PASS] testExactInput_A_L4_B() (gas: 1299361)
[PASS] testExactInput_A_L4_ETH() (gas: 1219944)
[PASS] testExactInput_ETH_L4_A() (gas: 1220450)
[PASS] testExactOutputSingle_A_ETH() (gas: 638153)
[PASS] testExactOutputSingle_A_L4() (gas: 714626)
[PASS] testExactOutputSingle_ETH_A() (gas: 643555)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 22.57ms (6.80ms CPU time)

Ran 18 test suites in 152.80ms (208.78ms CPU time): 93 tests passed, 0 failed, 0 skipped (93 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