Security Research Labs

# HydraDX Feature Security Assurance

Hacking assessment report

**V1.0, 03 June 2024**


| | | |
|---|---|---|
| Audited By: | Bruno Produit | bruno@srlabs.de |
| | Gabriel Arnautu | gabriel@srlabs.de |
| | Marc Heuse | marc@srlabs.de |



**Abstract.** This work describes the result of the thorough and independent security assurance audit performed by Security Research Labs for the HydraDX custom's EVM implementation layer which allows Metamask users to pay for gas using any supported asset on the HydraDX platform. Security Research Labs is a consulting firm that has been providing specialized audit services in the Polkadot ecosystem since 2019, including for the Substrate and Polkadot projects.

During this audit, HydraDX team provided access to relevant code and supported the research team effectively. The code in scope was verified to assure that the business logic of the product is resilient to hacking and abuse.

The research team identified several issues ranging from medium to info-level severity.

In addition to mitigating the open issues, Security Research Labs recommends further enhancing the existing documentation and tests around the HydraDX system.

## Content

# 1 Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the agreed-on timeframe and scope as detailed in Table 1. Please note that this report does not guarantee that all existing security vulnerabilities were discovered in the codebase exhaustively and that following all evolution suggestions described in Chapter 5 may not ensure all future code to be bug free.

## 2 Motivation and scope

Blockchains evolve in a trustless and decentralized environment, which by its own nature could lead to security issues. Ensuring availability and integrity is a priority for HydraDX as it aims to provide liquidity pools for a significant number of assets.

The current audit was a spotlight review covering a very specific feature of HydraDX: the custom implementation of their EVM layer which allows Metamask users to pay for gas using any supported asset on HydraDX.

The in-scope components/files are reflected in Table 1. The audit covered six Rust files which concern the above mentioned feature, using the **e3821e0** commit from the HydraDX-node GitHub repository [1].

| Repository | Component(s) |
|---|---|
| HydraDX-node | precompiles-multicurrency [2] |
| | evm-fee [3] |
| | evm-permit [4] |
| | dynamic-evm-fee [5] |
| | evm-accounts [6] |
| | transaction-multi-payment [7] |

Table 1. In-scope HydraDX components

## 3 Findings summary

We identified two issues - summarized in Table 2 - during our analysis of the in- scope files in the HydraDX codebase. In summary, one medium severity and one info severity issues were found.

| Issue | Severity | Status |
|---|---|---|
| Potential spamming attack via unsigned `dispatch_permit` extrinsic | Medium | Open |
| The mapping between Ethereum and Substrate addresses could create unwanted behavior | Info | Open |

Table 2 Issue summary

## 4 Detailed findings

### 4.1 Potential spamming attack via unsigned *dispatch_permit* extrinsic

| | |
|---|---|
| **Attack scenario** | **An attacker spams the network with unsigned transactions.** |
| **Location** | pallet-transaction-multi-payment |
| **Attack impact** | Slowing down the chain. |
| **Severity** | Medium |
| **Status** | **Open** |

The dispatch_permit extrinsic in the HydraDX node allows for unsigned transactions that perform heavy computational validation (e.g.: **validate_permit**, **decode_all_with_depth_limit**). This extrinsic is part of the **transaction-multi-payment** pallet and utilizes the validate_unsigned function to ensure the transaction's validity. The computationally intensive nature of this validation process can be exploited by submitting numerous unsigned extrinsics, potentially leading to a denial-of-service (DoS) attack on the network.

Moreover, the **validate_unsigned** logic duplicates the same code as the actual extrinsic, **dispatch_permit**, which adds even more computation time and resources for the entire call.

As the **dispatch_permit** is an unsigned extrinsic, there are no fees to be paid, increasing the attack surface.

The following risks can be attributed to this vulnerability:

1. **Spam vector**: An attacker can use a node to submit a large number of unsigned extrinsics, which will be gossiped across the network. This can lead to:
    - High computational load on collators/validators
    - Potential denial-of-service (DoS) attack, delaying or dropping legitimate transactions
    - Network congestion and performance degradation
2. **Resource consumption**: The heavy computation required for validating unsigned extrinsics can consume significant node resources, impacting overall network stability and performance.

Changing this extrinsic to be an **ensure_signed** extrinsic will remove the spamming vector by ensuring the payment of the computation from the caller's account.

## 4.2 The mapping between Ethereum and Substrate addresses could create unwanted behavior

| | |
|---|---|
| **Attack scenario** | **Legitimate user transfers tokens to an EVM address without a bound Substrate account.** |
| **Location** | pallet-evm-accounts |
| **Attack impact** | The user could lose their funds as the destination address cannot be claimed by a Substrate account. |
| **Severity** | Info |
| **Status** | Open |

HydraDX defines three types of addresses:

1. **Truncated address**: A substrate address created from an EVM address by prefixing it with "ETH\0" and appending with eight 0 bytes
2. **Full Substrate address**: Original 32 bytes long native address (not a truncated address)
3. **EVM address**: First 20 bytes of a Substrate address

This method of mapping EVM addresses to Substrate addresses raises a number of concerns:

- taking the raw bytes of an address is against best practices (i.e.: the first 20 bytes of a Substrate address are being used to represent an EVM address)
- the entropy of truncated addresses is reduced by 12 bytes as the address always begins with "ETH\0" and ends with eight zero bytes
- transferring funds to an EVM address which was not bound to a Substrate address, results in the funds being transferred to a truncated address, which cannot be claimed by any user

Legitimate users might lose funds by submitting a transfer to an EVM address that is not bound to a Substrate address. This could decrease the trust of the users in the HydraDX network. By reducing the addresses entropy, the chances of collision attacks increase significantly.

Our mitigation suggestion is to deprecate the usage of truncated addresses and make sure that every EVM address is bound to a Substrate address.

## 5 Evolution suggestions

**Improve unit and integration tests:** Various components of the HydraDX code are not covered by any tests. Extensive unit and integration tests should be developed, covering the largest number of use cases.

**Engage in an economic audit.** Although SRLabs has some knowledge of economic attacks, our primary goal during engagements is to find logic vulnerabilities through code assurance. Therefore, an economic audit for the HydraDX node is recommended to ensure the safety of the platform and its users.

**Regular code review and continuous fuzz testing.** Regular code reviews are recommended to avoid introducing new logic or arithmetic bugs, while continuous fuzz testing can identify potential vulnerabilities early in the development process. Ideally, HydraDX should continuously fuzz their code on each release.

## 6 Bibliography

[1] [Online]. Available: https://github.com/galacticcouncil/HydraDX-node/tree/e3821e078bdb72a0416f8aebca21ba4a7a599f64.

[2] [Online]. Available: https://github.com/galacticcouncil/HydraDX-node/blob/e3821e078bdb72a0416f8aebca21ba4a7a599f64/runtime/hydradx/src/evm/precompiles/multicurrency.rs.

[3] [Online]. Available: https://github.com/galacticcouncil/HydraDX-node/blob/e3821e078bdb72a0416f8aebca21ba4a7a599f64/runtime/hydradx/src/evm/evm_fee.rs.

[4] [Online]. Available: https://github.com/galacticcouncil/HydraDX-node/blob/e3821e078bdb72a0416f8aebca21ba4a7a599f64/runtime/hydradx/src/evm/permit.rs.

[5] [Online]. Available: https://github.com/galacticcouncil/HydraDX-node/blob/e3821e078bdb72a0416f8aebca21ba4a7a599f64/pallets/dynamic-evm-fee/src/lib.rs.

[6] [Online]. Available: https://github.com/galacticcouncil/HydraDX-node/blob/e3821e078bdb72a0416f8aebca21ba4a7a599f64/pallets/evm-accounts/src/lib.rs.

[7] [Online]. Available: https://github.com/galacticcouncil/HydraDX-node/blob/e3821e078bdb72a0416f8aebca21ba4a7a599f64/pallets/transaction-multi-payment/src/lib.rs#L372-L529.