CANTINA

# Ethereum Pectra: Nethermind

## Competition

July 21, 2025

# Contents

# 1 Introduction

## 1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

## 1.2 Disclaimer

A competition provides a broad evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While competitions endeavor to identify and disclose all potential security issues, they cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities, therefore, any changes made to the code would require an additional security review. Please be advised that competitions are not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

## 1.3 Risk assessment

| Severity | Description |
|---|---|
| **High** | *Must* fix as soon as possible (if already deployed) and can be triggered by any user without significant constraints, generating outsized returns to the exploiter. For example: loss of user funds (significant amount of funds being stolen or lost) or breaking core functionality (failure in fundamental protocol operations). |
| **Medium** | Global losses <10% or losses to only a subset of users, requiring significant constraints (capital, planning, other users...) to be exploited. For example: temporary disruption or denial of service (DoS), minor fund loss or exposure or breaking non-core functionality |
| **Low** | Losses will be annoying but easily recoverable, requiring unusual scenarios or admin actions to be exploited. |
| **Gas Optimization** | Suggestions around gas saving practices. |
| **Informational** | Suggestions around best practices or readability. |

### 1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above matrix. High severity findings represent the most critical issues that must be addressed immediately, as they either have high impact and high likelihood of occurrence, or medium impact with high likelihood.

Medium severity findings represent issues that, while not immediately critical, still pose significant risks and should be addressed promptly. These typically involve scenarios with medium impact and medium likelihood, or high impact with low likelihood.

Low severity findings represent issues that, while not posing immediate threats, could potentially cause problems in specific scenarios. These typically involve medium impact with low likelihood, or low impact with medium likelihood.

Lastly, some findings might represent improvements that don't directly impact security but could enhance the codebase's quality, readability, or efficiency (Gas and Informational findings).

## 2 Security Review Summary

Ethereum is a worldwide system, an open-source platform to write computer code that stores and automates digital databases using smart contracts, without relying upon a central intermediary, solving trust with cryptographic techniques.

From Feb 21st to Mar 27th Cantina hosted a competition based on the Ethereum Pectra upgrade. The present report focuses in the nethermind implementation. The participants identified a total of **6** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 4

# 3 Findings

## 3.1 Low Risk

### 3.1.1 EIP-7702: Permanent DoS of an EOA that has performed a SetCodeTx due to improper handling in mempool

*Submitted by Haxatron*

**Severity:** Low Risk

**Context:** *(No context files were provided by the reviewer)*

**Description:** The new EIP-7702 transaction is handled differently in the mempool, with strict limit on how many pending transactions an EOA can have. However, how Nethermind handles this transaction is problematic.

- DelegatedAccountFilter.cs#L29-L40:

```
if (pendingDelegations.HasPending(tx.SenderAddress!))
{
    //Check if the sender has a self-sponsored SetCode transaction with same nonce.
    //If he does then this is a replacement tx and should be accepted
    if (!standardPool.BucketAny(tx.SenderAddress!,
        t => t.Nonce == tx.Nonce
        && t.HasAuthorizationList
        && t.AuthorizationList.Any(tuple => tuple.Authority == tx.SenderAddress)))
    {
        return AcceptTxResult.PendingDelegation;
    }
}
```

In particular in the above code, if the account has a pending authorization (which means a EIP-7702 transaction that is inside the mempool but has not been confirmed), which is stored in `pendingDelegations`, then the account is only allowed to submit EIP-7702 transactions of the same nonce and has the same authorization as the sender. Here is the problem associated with this:

1. Firstly, anyone can submit a valid authorization tuple for an account that is different from the transaction sender. This means that it is possible to submit even an old valid authorization tuple, which will be added to a `pendingDelegation` cache.
2. If this occurs, the victim account will be stored in `pendingDelegations`.
3. Effectively, the victim EOA can be permanently DOS from submitting transactions until the attacker's transaction that contains the victim authorization is removed from the mempool, the attacker can set extremely low base fee which effectively DOS it forever.

**Recommendation:** If the EOA has a pending delegation, then allow any transaction of the next nonce, this match the GETH behaviour.

```
- if (pendingDelegations.HasPending(tx.SenderAddress!))
-     {
-         //Check if the sender has a self-sponsored SetCode transaction with same nonce.
-         //If he does then this is a replacement tx and should be accepted
-         if (!standardPool.BucketAny(tx.SenderAddress!,
-             t => t.Nonce == tx.Nonce
-             && t.HasAuthorizationList
-             && t.AuthorizationList.Any(tuple => tuple.Authority == tx.SenderAddress)))
-         {
-             return AcceptTxResult.PendingDelegation;
-         }
-     }

- if (!codeInfoRepository.TryGetDelegation(worldState, tx.SenderAddress!, out _))
+ if (!codeInfoRepository.TryGetDelegation(worldState, tx.SenderAddress!, out _) &&
↪  !pendingDelegations.HasPending(tx.SenderAddress!))
+     return AcceptTxResult.Accepted;
- //If the account is delegated we only accept the next transaction nonce
+ //If the account is delegated or has pending delegation we only accept the next transaction nonce
  if (state.SenderAccount.Nonce != tx.Nonce)
  {
      return AcceptTxResult.FutureNonceForDelegatedAccount;
  }
```

### 3.1.2 EIP-7702: Dangling delegation in the `pendingDelegation` cache in edge case

*Submitted by Haxatron*

**Severity:** Low Risk

**Context:** *(No context files were provided by the reviewer)*

**Description:** The pending delegation count of an account is stored in a `pendingDelegation` cache. This helps enforce restrictions on the EOA when they later submit a transaction. How it is tracked is as follows:

- When `AddCore` returns `AcceptTxResult.Accepted`, then the `pendingDelegation` cache is incremented via `AddPendingDelegations` (TxPool.cs#L454-L466):

```
else
{
    accepted = AddCore(tx, ref state, startBroadcast);
    if (accepted)
    {
        AddPendingDelegations(tx);
        // Clear proper snapshot
        if (tx.SupportsBlobs)
            _blobTransactionSnapshot = null;
        else
            _transactionSnapshot = null;
    }
}
```

- However, there is an edge case where the transaction is not added to the mempool but `AcceptTxResult.Accepted` is returned. This occurs when the transaction pool is full of better transactions and persistent broadcast is enabled and the transaction is broadcasted (TxPool.cs#L508):

```
private AcceptTxResult AddCore(Transaction tx, ref TxFilteringState state, bool isPersistentBroadcast)
{
    bool eip1559Enabled = _specProvider.GetCurrentHeadSpec().IsEip1559Enabled;
    UInt256 effectiveGasPrice = tx.CalculateEffectiveGasPrice(eip1559Enabled, _headInfo.CurrentBaseFee);
    TxDistinctSortedPool relevantPool = (tx.SupportsBlobs ? _blobTransactions : _transactions);
    ...
    if (tx.Hash == removed?.Hash)
    {
        // it means it was added and immediately evicted - pool was full of better txs
        if (!isPersistentBroadcast || tx.SupportsBlobs || !_broadcaster.Broadcast(tx, true))
        {
            // we are adding only to persistent broadcast - not good enough for standard pool,
            // but can be good enough for TxBroadcaster pool - for local txs only
            Metrics.PendingTransactionsPassedFiltersButCannotCompeteOnFees++;
            return AcceptTxResult.FeeTooLowToCompete;
        }
        else
        {
            return AcceptTxResult.Accepted;
        }
    }
```

In the scenario above, `AddCore` returns `AcceptTxResult.Accepted` because while the transaction was not added to the local mempool, it was successfully broadcasted so the `else` branch is followed.

- Since it is not in the mempool, when attempting to remove the transaction it will not be removed and the `RemovePendingDelegations` will not be called (TxPool.cs#L712-L722):

```
bool hasBeenRemoved = _transactions.TryRemove(hash, out Transaction? transaction)
                      || _blobTransactions.TryRemove(hash, out transaction);

if (transaction is null || !hasBeenRemoved)
{
    return false;
}

if (hasBeenRemoved)
{
    RemovedPending?.Invoke(this, new TxEventArgs(transaction));

    RemovePendingDelegations(transaction);
}
```

In essence, there will be a dangling pending delegation that will never be deleted from the cache.

**Recommendation:** Must not add the pending delegation when `AcceptTxResult.Accepted` is received in the `else` branch:

```
if (tx.Hash == removed?.Hash)
{
    // it means it was added and immediately evicted - pool was full of better txs
    if (!isPersistentBroadcast || tx.SupportsBlobs || !_broadcaster.Broadcast(tx, true))
    {
        // we are adding only to persistent broadcast - not good enough for standard pool,
        // but can be good enough for TxBroadcaster pool - for local txs only
        Metrics.PendingTransactionsPassedFiltersButCannotCompeteOnFees++;
        return AcceptTxResult.FeeTooLowToCompete;
    }
    else
    {
        return AcceptTxResult.Accepted;
    }
}
```

## 3.2 Informational

### 3.2.1 Duplicated nonce check in `IsValidForExecution` for EIP-7702 support

*Submitted by zigtur, also found by flacko, Haxatron and alexfilippov314*

**Severity:** Informational

**Context:** TransactionProcessor.cs#L278-L294

**Description:** The `IsValidForExecution` function executes twice the same check on the `authorizationTuple.Nonce`:

```
bool IsValidForExecution(
    AuthorizationTuple authorizationTuple,
    StackAccessTracker accessTracker,
    [NotNullWhen(false)] out string? error)
{
    // ...

    if (authorizationTuple.Nonce == ulong.MaxValue)
    {
        error = $"Nonce ({authorizationTuple.Nonce}) must be less than 2**64 - 1.";
        return false;
    } // @POC: check 1

    if (authorizationTuple.AuthoritySignature.ChainId is not null &&
    ↪  authorizationTuple.AuthoritySignature.ChainId != authorizationTuple.ChainId)
    {
        error = "Bad signature.";
        return false;
    }

    if (authorizationTuple.Nonce == ulong.MaxValue)
    {
        error = $"Nonce ({authorizationTuple.Nonce}) must be less than 2**64 - 1.";
        return false;
    } // @POC: check2
```

**Recommendation:** Consider applying the following patch to remove one of these two checks:

```
diff --git a/src/Nethermind/Nethermind.Evm/TransactionProcessing/TransactionProcessor.cs
↪  b/src/Nethermind/Nethermind.Evm/TransactionProcessing/TransactionProcessor.cs
index e428f0f4cf..b1b09d2eb6 100644
--- a/src/Nethermind/Nethermind.Evm/TransactionProcessing/TransactionProcessor.cs
+++ b/src/Nethermind/Nethermind.Evm/TransactionProcessing/TransactionProcessor.cs
@@ -287,11 +287,6 @@ namespace Nethermind.Evm.TransactionProcessing
                     return false;
                 }

-                if (authorizationTuple.Nonce == ulong.MaxValue)
-                {
-                    error = $"Nonce ({authorizationTuple.Nonce}) must be less than 2**64 - 1.";
-                    return false;
-                }
                 accessTracker.WarmUp(authorizationTuple.Authority);

                 if (WorldState.HasCode(authorizationTuple.Authority) &&
                 ↪  !_codeInfoRepository.TryGetDelegation(WorldState, authorizationTuple.Authority, out _))
```

### 3.2.2 EIP-7702: Incorrect metrics reported for EIP-7702 transaction

*Submitted by Haxatron*

**Severity:** Informational

**Context:** *(No context files were provided by the reviewer)*

**Description:** Currently the metric `Metrics.Eip7702TransactionsInBlock` has a description that tracks the `Ratio of 7702-type transactions in the block.` (Metrics.cs#L126):

```
[GaugeMetric]
[Description("Ratio of 7702-type transactions in the block.")]
public static long Eip7702TransactionsInBlock { get; set; }
```

However, the absolute number of EIP-7702 transactions is being tracked instead of the ratio (TxPool.cs#L375):

```
Metrics.Eip7702TransactionsInBlock = eip7702Txs;
```

**Recommendation:** Either fix the metric description or change the variable to track the ratio:

```
- Metrics.Eip7702TransactionsInBlock = eip7702Txs;
+ Metrics.Eip7702TransactionsRatio = (float)eip7702Txs / transactionsInBlock;
```

### 3.2.3 The `debug` RPC methods don't warm up delegation target

*Submitted by alexfilippov314*

**Severity:** Informational

**Context:** *(No context files were provided by the reviewer)*

**Description:** EIP-7702 introduces a new transaction type that allows setting code for an EOA. According to the EIP, if the `to` address has a delegation designation, the delegation target should be warmed up:

> Additionally, if a transaction's destination has a delegation designation, add the target of the delegation to accessed_addresses.

The issue arises because Nethermind does not warm up the delegation target in the `debug_traceCall` and `debug_traceTransaction` RPC methods. This behavior is demonstrated in the POC section.

The root cause lies in the implementation of `OverridableCodeInfoRepository`, which is used during debug calls and does not return the `delegationAddress` in its `GetCachedCodeInfo` functions:

```
public CodeInfo GetCachedCodeInfo(IWorldState worldState, Address codeSource, IReleaseSpec vmSpec, out
↪  Address? delegationAddress)
{
    return GetCachedCodeInfo(worldState, codeSource, true, vmSpec, out delegationAddress);
}
public CodeInfo GetCachedCodeInfo(IWorldState worldState, Address codeSource, bool followDelegation,
↪  IReleaseSpec vmSpec, out Address? delegationAddress)
{
    delegationAddress = null;
    return _codeOverwrites.TryGetValue(codeSource, out CodeInfo result)
        ? result
        : codeInfoRepository.GetCachedCodeInfo(worldState, codeSource, vmSpec);
}
```

Due to this, TransactionProcessor.cs#L566 in `TransactionProcessorBase.BuildExecutionEnvironment` is not executed, leading to a gas miscalculation later caused by a cold delegation target:

```
if (delegationAddress is not null)
    accessTracker.WarmUp(delegationAddress);
```

Since this issue occurs only during `debug` RPC methods, its impact is limited to external tools like block explorers or wallets, which may display incorrect gas values if they rely on this method.

**Proof of Concept:**

1. Build a local `nethermind` image. I've used the following commit: `29cccb1fc1028d3efcdf04edfbb91e85f217c932` (7 Mar).

   ```
   git clone https://github.com/NethermindEth/nethermind
   cd nethermind
   docker buildx build --platform linux/amd64 -t local-nethermind:latest .
   ```

2. Set up the network using `Kurtosis`:

   - `network_params.yaml`:

```
    participants:
    - el_type: geth
        cl_type: prysm
        count: 1
    - el_type: nethermind
        el_image: local-nethermind:latest
        cl_type: prysm
        count: 1
    - el_type: reth
        el_image: reth-local:latest
        cl_type: prysm
        count: 1

    network_params:
    network_id: "585858"
    electra_fork_epoch: 1
    genesis_gaslimit: 30000000
    seconds_per_slot: 12
```

- `Makefile`:

```
    build-geth: ## build geth
        cd /home/allfi/src/audits/cantina/pectra/execution/go-ethereum && docker build -t
        ↪  local-ef-geth:latest .

    start-e2e: ## start the e2e
        kurtosis run github.com/ethpandaops/ethereum-package --args-file ./network_params.yaml
        ↪  --image-download always --enclave e2e

    kill-e2e: ## stop e2e tests
        kurtosis enclave stop e2e
        kurtosis enclave rm e2e

    ## Phony targets
    .PHONY: build-geth start-e2e kill-e2e
```

3. Run the network:

```
make start-e2e
```

4. Use the following command with `geth` and `nethermind` RPC:

```
cast rpc debug_traceCall --raw '[
{
    "from": "0x069B02919Cbc2671bF1dB26745DfEDB8d3c7D146",
    "to": "0xFB2C8Ff1f4c60Abf5dcC3940a6C1F393e6d764dd",
    "data": "0x9e5faafc",
    "authorizationList": [
    {"chainId":"0x8f082","address":"0xed1636f035eb18dc509e8d8976e4a7ae3ebc7572","nonce":"0x0","yParity"
    ↪  :"0x1","r":"0xc006ae640e5e098379e9e9e150c3356a7bbd488b7cbec248f586a29ebf2ef92f","s":"0x2f0db907
    ↪  2425ee2a4eaac0ad2f3895c7d5476dfa84a08ef93137dc408874aa34"}
    ]
},
"latest",
{
    "stateOverrides": {
    "0xed1636f035eb18dc509e8d8976e4a7ae3ebc7572": {
        "code": "0x6080604052600436106101e575f3560e01c80639e5faafc14602157601f565b5b005b348015602b575f5f
        ↪  fd5b5060326034565b005b5f73fb2c8ff1f4c60abf5dcc3940a6c1f393e6d764dd90508073ffffffffffffffff
        ↪  ffffffffffffffffffffffff16604051606d9060d8565b5f604051808303815f865af19150503d805f811460a457
        ↪  6040519150601f19603f3d011682016040523d82523d5f602084013e60a9565b606091505b50505050565b5f819
        ↪  05092915050565b50565b5f60c55f8360af565b915060ce8260b9565b5f82019050919050565b5f60e08260bc56
        ↪  5b9150819050919050565fea264697066735822122091 88a98d6a0554dd699864a1d2c9ca3a424030ce0ae1b3180
        ↪  a78ece0d60a826d64736f6c634300081c0033"
    }
    }
}
]' --rpc-url 127.0.0.1:32841
```

Here, the authorization sets the delegation designation for `0xFB2C8Ff1f4c60Abf5dcC3940a6C1F393e6d764dd`. The bytecode corresponds to the following contract:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract POC {
    function attack() public {
        address a = 0xFB2C8Ff1f4c60Abf5dcC3940a6C1F393e6d764dd;
        a.call("");
    }

    fallback() external payable {

    }
}
```

5. For the call above, `geth` returns a gas cost of 46712, while `nethermind` returns 49212. The 2500 gas difference precisely matches the cost of a cold vs. warm storage read.

**Recommendation:** Consider refactoring `OverridableCodeInfoRepository` to ensure that `debug` RPC methods return accurate values.

#### 3.2.4 `OverridableCodeInfoRepository` fails to follow 7702 delegations in some cases causing incorrect simulations

*Submitted by guhu95*

**Severity:** Informational

**Context:** OverridableCodeInfoRepository.cs#L23-L29

**Summary:** The linked method ignores `bool followDelegation` which is true when the delegation's code should be loaded. So if the code override for a simulated transaction is an EIP-7702 pointer (`0xef0100<delegation-address>`), instead of following the delegation like in the class's other methods, the method will return the pointer bytecode instead of executable code.

**Finding Description:** This method is used directly above it, which in turn is used in two places where delegation must be followed:

1. `CodeInfoRepositoryExtensions.GetCachedCodeInfo`, which is used in `VM.InstructionCall` to load the code for CALL* opcode execution. This will result in an execution error if the address with override is called.
2. `TransactionProcessorBase`'s `BuildExecutionEnvironment` which also both load incorrect code, and will also incorrectly calculate the gas due to the incorrectly returned null `delegationAddress`.

**Impact:** If a simulation or tracing request tries to set a code override for with an EIP-7702 type pointer, only some places will account for it, but the execution will incorrectly revert.

**Recommendation:** If the overriden code is a 7702 pointer, the method should try to load its delegation destination if the flag is `true` and return the extracted `delegationAddress`.