Cyberscope
A TAC Security Company

Audit Report

MCN Chain

November 2025

Repository	https://github.com/mcnchain/MCNBlockchain
Commit	5f7e274f09965163d5e517c1922cc8e695a6ae70
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/mcnchain/MCNBlockchain
Commit	5f7e274f09965163d5e517c1922cc8e695a6ae70

Audit Updates

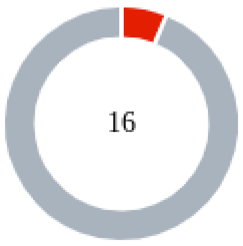
Initial Audit	29 Oct 2025
Corrected Phase 2	25 Nov 2025

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	DFD	Duplicate Fee Deduction	Unresolved
●	AGI	Agnostic GETH Implementation	Unresolved
●	CR	Centralization Risk	Unresolved
●	CFD	Concurrent Fee Distribution	Unresolved
●	EBGL	Excessive Block Gas Limit	Unresolved
●	GSV	Global State Variables	Unresolved
●	HV	Hardcoded Values	Unresolved
●	MFIM	Missing Fee Incentivization Mechanism	Unresolved
●	MFC	Missing Flag Configurations	Unresolved
●	MIV	Missing Input Validation	Unresolved
●	NPFS	Non Persistent Fee State	Unresolved
●	PRA	Permissive RPC Access	Unresolved
●	PFD	Possible Funds Discrepancy	Unresolved
●	RUC	Redundant Unlock Configuration	Unresolved
●	USA	Unconstrained System Access	Unresolved
●	UPPA	Unrestricted P2P Port Access	Unresolved

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	15

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	15	0	0	0

DFD - Duplicate Fee Deduction

Criticality	Critical
Location	split.js#L42
Status	Unresolved

Description

The `split.js` script is configured to operate in `'total'` gas share mode, which calculates and distributes transaction fees based on the full `effectiveGasPrice * gasUsed`. Under EIP-1559, `effectiveGasPrice` includes both the base fee and the miner tip. However, the base fee is already burned by the protocol, meaning that in `'total'` mode, the script effectively redistributes the base fee again from the miner's balance. This results in the miner unintentionally paying the base fee twice—once through protocol-level burning and again through redistribution.

This behavior can lead to a significant and unintended loss of funds for the miner.

Shell
```
var GAS_SHARE_MODE = 'total';
```

Shell

```
if (GAS_SHARE_MODE === 'miner'){
    var base = (b && b.baseFeePerGas != null) ?
bn(b.baseFeePerGas) : null;
    if (base){
      var tip = egp.sub(base); if (tip.lt(0)) tip =
bn(0);
      return tip.mul(gasUsed);
    } else {
      return egp.mul(gasUsed);
    }

  }
```

Recommendation

To prevent miners from incurring unintended losses, the `'miner'` mode should be used instead of `'total'`. This mode calculates fees based solely on the actual miner tip, ensuring that only the portion of the fee the miner actually receives is redistributed. This approach accurately reflects the miner’s real earnings and avoids double-counting the base fee, which is already burned by the protocol.

AGI - Agnostic GETH Implementation

Criticality	Minor / Informative
Location	geth.service#L13
Status	Unresolved

Description

The systemd service configuration for running the Geth client on the MCN chain references the Geth binary using a direct path (`/usr/local/bin/geth`) without specifying a particular version. This setup relies on the locally installed Geth binary, which may be updated or replaced without notice. As a result, the service does not enforce version consistency, potentially leading to unexpected behavior or incompatibilities if the binary is modified or upgraded. This lack of version pinning introduces a risk of inconsistencies across deployments and may expose users to security vulnerabilities if a compromised or untested code is executed.

Shell
```
ExecStart=/usr/local/bin/geth
```

Recommendation

To ensure consistency and reduce the risk of unexpected behavior, it is recommended to pin the Geth binary to a specific version. This can be achieved by referencing a versioned binary path or by managing the binary through a version control mechanism. Pinning the version guarantees that all deployments use tested and verified Geth release, minimizing compatibility issues and enhancing security.

CR - Centralization Risk

Criticality	Minor / Informative
Location	geth.service#L29 split.js#L9
Status	Unresolved

Description

The chain's initial configuration introduces a centralized setup, where a single validator is responsible for sealing blocks. While this approach simplifies early-stage coordination, it also introduces centralization risks that should be carefully considered. These risks include:

- **Single Point of Control**: With only one validator, the chain's operation depends entirely on a single entity, increasing the risk of downtime or manipulation.
- **Centralized Fee Distribution**: The fee-splitting script distributes transaction fees to a predefined list of validators, who may not yet be authorized to seal blocks. This creates a discrepancy between fee recipients and active participants in consensus.
- **Decentralization Erosion**: If new validators are not added promptly or fairly, the chain may remain centralized longer than intended, undermining its trustless design.

Shell
```
--miner.etherbase
0xfE4800cA5a48670cc5f591a51F517BA8F682D4Ae
```

Shell
```
var VALIDATORS = [
  "0x014B5831d8a4449Bd8bEbef4540B865a4563Cbd0",
  "0x57d5937C6942B5f59c44c7400834A4946C397734",
  "0x7e96EdCA7866403739E18Fd91582f9A86D4b085a",
  "0xfE4800cA5a48670cc5f591a51F517BA8F682D4Ae"

];
```

Recommendation

To mitigate centralization risks in the chain's initial setup, it is recommended to evaluate the feasibility of embedding a more decentralized configuration directly into the chain's genesis and operational parameters. This would reduce reliance on manual or external coordination and promote a more trustless environment from the outset. It is important to balance the simplicity of early-stage deployment with the long-term benefits of decentralization and resilience.

CFD - Concurrent Fee Distribution

Criticality	Minor / Informative
Location	split.js
Status	Unresolved

Description

The `_isSplitBusy` flag is intended to act as a lock to prevent reentrant execution of the `processBlocksAndDistribute()` function. However, this flag is not implemented using atomic operations or thread-safe constructs. In environments where multiple triggers may invoke the function concurrently, this can lead to a race condition. As a result, multiple instances of the function could run simultaneously, potentially causing inconsistent state updates or duplicate fund distributions.

Recommendation

By revising the `_isSplitBusy` flag into a more robust locking mechanism the team can ensure that only one instance of `processBlocksAndDistribute()` executes at a time. This prevents race conditions and ensures consistent and reliable fee distribution.

EBGL - Excessive Block Gas Limit

Criticality	Minor / Informative
Location	genesis-mainnet.json#L20
Status	Unresolved

Description

The genesis block sets the `gasLimit` to a value of `900,000,000`. This excessive limit allows a single transaction to consume a significant amount of computational resources, potentially leading to performance degradation or denial-of-service scenarios on node hardware. Such a configuration increases the risk of accidental or malicious transactions monopolizing system resources.

Shell
```
{
  "config": {
    ...
  "gasLimit": "900000000",
  ...

}
```

Recommendation

To mitigate the risk of resource exhaustion, it is recommended to reduce the `gasLimit` in the genesis block to more conservative values. This adjustment limits the maximum computational effort a single block can consume, helping to prevent performance degradation and potential denial-of-service attacks on network nodes.

GSV - Global State Variables

Criticality	Minor / Informative
Location	split.js
Status	Unresolved

Description

Global variables such as `_feePool`, `_lastProcessedBlk`, and `_isSplitBusy` are defined in the global scope of the `split.js` script without any form of namespacing. This practice increases the risk of variable name collisions, especially when multiple scripts are loaded into the same runtime environment, such as the Geth JavaScript console. If another script defines a variable with the same name, it could unintentionally overwrite or be overwritten, leading to unpredictable behavior or logic errors.

Recommendation

Encapsulate global state variables within a dedicated namespace object (e.g., `SplitGasState`). This approach isolates the script's internal state, prevents naming conflicts with other scripts, and improves code maintainability.

HV - Hardcoded Values

Criticality	Minor / Informative
Location	split.js
Status	Unresolved

Description

The script currently uses a hardcoded gas limit of `21000` and a fixed priority fee of `1 gwei` for all transactions. While this configuration may be sufficient for simple value transfers, it does not account for varying network conditions or more complex transaction types that may require higher gas limits or dynamic fee adjustments. As a result, transactions may fail if the actual gas requirements exceed the fixed values.

Recommendation

It is recommended to update the script to dynamically estimate the required gas limit and adjust the priority fee based on current network conditions. This ensures that transactions are more likely to succeed under varying loads and execution contexts, improving reliability and reducing the risk of failed transfers.

MFIM - Missing Fee Incentivization Mechanism

Criticality	Minor / Informative
Location	geth.service
Status	Unresolved

Description

The current system design incentivizes fee collection for a specific set of validators by directing mining rewards and transaction fees to a predefined etherbase address. However, users are not incentivized to include priority fees in their transactions. In low-congestion conditions, Geth prioritizes maintaining network liveness, allowing transactions with minimal or no priority fees to be processed. This behavior can reduce the effectiveness of fee-based incentives for validators, as transactions are still included even when they offer little economic reward.

Recommendation

To improve validator incentives and ensure sustainable network participation, consider adjusting the fee model to encourage users to include priority fees in their transactions. This can be achieved by modifying the transaction inclusion logic or fee parameters so that transactions with higher priority fees are more likely to be processed. Aligning user incentives with validator rewards enhances economic fairness and strengthens the long-term security and reliability of the network.

MFC - Missing Flag Configurations

Criticality	Minor / Informative
Location	geth.service
Status	Unresolved

Description

Geth offers several runtime flags designed to safeguard node performance and prevent misuse of its JSON-RPC interface. These flags help mitigate resource exhaustion and reduce the risk of costly operational errors.

- The `--rpc.gascap` flag sets a maximum gas limit for read-only calls such as `eth_call` and `eth_estimateGas`. Without this restriction, clients could simulate transactions with excessively high gas values, leading to unnecessary CPU consumption. Enforcing a reasonable gas cap ensures that such requests are either rejected or truncated, preserving node efficiency.
- The `--rpc.txfeecap` flag limits the maximum transaction fee accepted via `eth_sendTransaction`. This prevents transactions with unreasonably high `maxFeePerGas` values from being processed. Setting this cap helps avoid accidental financial loss and protects against potential denial-of-service scenarios.

Additionally, Geth provides HTTP server controls to manage idle or slow client connections:

- The `--http.idleseconds` flag defines how long an idle connection is allowed to remain open. For example, `--http.idleseconds 30` will close any connection that remains inactive for more than 30 seconds.
- The `--http.timeout` flag sets a maximum duration for processing a request. For instance, `--http.timeout 15s` will abort any request that takes longer than 15 seconds to complete.

These flags are not currently configured in the provided service setup, leaving the node vulnerable to inefficient resource usage and potential abuse.

Recommendation

It is recommended to configure Geth's runtime flags,such as `--rpc.gascap`, `--rpc.txfeecap`, `--http.idleseconds`, and `--http.timeout`, to enforce limits on gas usage, transaction fees, and HTTP connection behavior. These settings help safeguard node performance, prevent resource exhaustion, and reduce the risk of misuse or accidental financial loss.

MIV - Missing Input Validation

Criticality	Minor / Informative
Location	split.js
Status	Unresolved

Description

The split distribution script does not perform any validation to ensure that the addresses listed in the `VALIDATORS` array are properly formatted addresses, nor does it verify that the weights in `VALIDATOR_WEIGHTS` are positive numbers. This lack of input validation could lead to unpredictable behavior during fee distribution, such as failed transactions or incorrect allocation of funds.

Recommendation

It is recommended to implement input validation for both validator addresses and their corresponding weights. Ensuring that each address is a valid address and that all weights are positive numbers will prevent misconfigurations and reduce the risk of failed or incorrect fee distributions.

NPFS - Non Persistent Fee State

Criticality	Minor / Informative
Location	split.js
Status	Unresolved

Description

The fee distribution script relies on a temporary filter created by `eth.filter("latest")` to trigger execution when new blocks are mined. However, this filter is not persistent and is lost if the Ethereum node (Geth) is restarted. As a result, the internal state variables `_feePool` and `_lastProcessedBlk` , which track accumulated fees and the last processed block number, are reset to their initial values. This leads to a loss of fee data collected while the script was inactive, and those fees are never distributed.

This behavior introduces an inconsistency where any downtime or restart of the node or script results in permanent loss of unprocessed fee data, undermining the integrity of the fee distribution mechanism.

Recommendation

Persist `_feePool` and `_lastProcessedBlk` using a reliable storage mechanism such as LevelDB or a local JSON file. This ensures that fee data and block tracking information are retained across node or script restarts, preventing data loss and maintaining the integrity of the fee distribution process.

PRA - Permissive RPC Access

Criticality	Minor / Informative
Location	geth.service
Status	Unresolved

Description

The current Geth configuration binds both HTTP and WebSocket interfaces to `127.0.0.1` , which restricts access to the local machine. However, the use of `--http.vhosts "*"` and `--http.corsdomain "*"` allows requests from any origin and any host header. If the node operator later exposes port 8545 to the public, intentionally or accidentally, this configuration would permit any website to interact with the node directly from a user's browser.

Such exposure could allow malicious web pages to send arbitrary JSON-RPC requests, including transaction submissions, on behalf of the user. This creates a significant security risk, especially since the node is configured with `--allow-insecure-unlock` and an unlocked account. While the current setup permits local-only binding, the permissive CORS and vhost settings would become vulnerable if network exposure occurs.

Recommendation

The team is advised to restrict `--http.vhosts` and `--http.corsdomain` to trusted domains only. This limits cross-origin access and reduces the attack surface in case of accidental network exposure.

PFD - Possible Funds Discrepancy

Criticality	Minor / Informative
Location	split.js#L91
Status	Unresolved

Description

The script responsible for distributing transaction fees includes a function, `safeSend`, which attempts to send ETH to designated addresses, including a burn address (`0x000000000000000000000000000000000000dEaD`). However, this function does not verify whether the transaction was successfully mined. If the transaction fails, the script continues execution and deducts the intended amount from the `_feePool` regardless of the outcome.This may result in silent loss of funds intended for burning or distribution.

Shell
```
function safeSend(from, to, valueWei){
  var v = bn(valueWei);
  if (!to || v.lte(0)) return null;
  try {
    var f = currentFees();
    var tx = { from: from, to: to, value: asWeiHex(v), gas:
GAS_LIMIT };
    if (f.mode === "eip1559"){ tx.maxPriorityFeePerGas =
f.prio; tx.maxFeePerGas = f.max; }
    else { tx.gasPrice = f.gasPrice; }
    return eth.sendTransaction(tx);
  } catch(e){
    console.log("send failed:", to, fromWeiStr(v), "err:",
e);
    return null;
  }}
```

Recommendation

It is recommended to verify the success of each transaction before updating the `_feePool`. This ensures that funds are only deducted when the ETH transfer is successfully mined, preventing silent loss of funds intended for burning or distribution.

RUC - Redundant Unlock Configuration

Criticality	Minor / Informative
Status	Unresolved

Description

The Geth node is configured with the `--allow-insecure-unlock` flag, which permits account unlocking via HTTP or WebSocket RPC requests. This setting is intended for development environments and poses a security risk in production. Although the validator key is already unlocked at startup using the `--unlock` and `--password` flags, enabling `--allow-insecure-unlock` unnecessarily broadens the unlocking surface.

If an attacker gains local access or if the RPC interface is inadvertently exposed, they could issue RPC calls to re-unlock accounts or keep them unlocked indefinitely, potentially leading to unauthorized fund transfers.

Recommendation

It is recommended to remove the `--allow-insecure-unlock` flag. Relying solely on `--unlock` and `--password` at startup ensures the validator key is available for mining without exposing it to additional RPC-based unlock attempts. This reduces the risk of unauthorized access and strengthens the node’s security posture.

USA - Unconstrained System Access

Criticality	Minor / Informative
Location	geth.service
Status	Unresolved

Description

The Geth node is configured to run under a non-privileged user using systemd. However, the associated systemd unit file lacks hardening directives, leaving the Geth process with broad read access to the host filesystem and unrestricted read/write access within the user's home directory.

This unrestricted access significantly increases the system's attack surface. In the event of a compromised execution an attacker could:

- Access sensitive files on the host system.
- Modify user-owned scripts or binaries to establish persistence.
- Deploy malware in shared directories.

Recommendation

It is recommended to apply standard systemd hardening flags to the Geth service unit such as:

Shell
```
- ProtectSystem=strict
- ProtectHome=true
- PrivateTmp=true
- NoNewPrivileges=yes
```

These directives limit the Geth process to a read-only view of system directories, restrict access to other users' home directories, isolate temporary files, and block privilege escalation. This reduces the attack surface and mitigates the risk of persistence or lateral movement in the event of a compromise.

UPPA - Unrestricted P2P Port Access

Criticality	Minor / Informative
Location	geth.service
Status	Unresolved

Description

The Geth node is configured with its default peer-to-peer (P2P) settings, which leave port 30303 open for both TCP and UDP traffic and enable public peer discovery. While the HTTP and WebSocket APIs are bound to the local interface (127.0.0.1), the P2P service listens on all network interfaces. This allows any external Ethereum client that knows the node's enode URL to initiate a handshake, maintain a connection, and send traffic.

As a result, the node is exposed to unsolicited connections from the public internet. Malicious actors could exploit this by flooding the node with malformed blocks, oversized transaction pools, or other resource-intensive requests, potentially degrading performance or causing denial-of-service conditions.

Recommendation

For private chains, disable public peer discovery and restrict P2P connections by using one of the following startup options:

- `--nodiscover` along with `--bootnodes <trusted-enodes>` to connect only to known peers.
- `--maxpeers 0` if the node is intended to operate as a solo validator.
- `--netrestrict <CIDR>` to limit peer connections to a specific LAN or VPN range. Additionally, firewall or externally close port 30303 to block unwanted access.

Summary

MCN implements an EVM network based on Geth. Scripts automate genesis initialization, and network configuration. A JavaScript component collects transaction fees and distributes them to validators, developers, and ecosystem addresses. This audit investigates security issues, business logic, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a TAC blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A *TAC Security* Company

The Cyberscope team

cyberscope.io