Cyberscope

A TAC Security Company

Static Analysis Report

Unipoly

November 2025

Repository	https://github.com/Mohammadali-Ghods/UnipolyChain
Commit	3c38d295f994260c65a475c8c7c37024ab26ae91
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/Mohammadali-Ghods/UnipolyChain
Commit	3c38d295f994260c65a475c8c7c37024ab26ae91

Audit Updates

Initial Audit	10 Nov 2025

Assessment Scope

The scope of this assessment encompasses a comprehensive static analysis of the application's source code repository. The objective is to identify security vulnerabilities, code weaknesses, and potential misconfigurations that could impact the integrity, confidentiality, or availability of the application. The evaluation included the review of backend and frontend components (where applicable), with a focus on detecting insecure coding patterns, dependency risks, and deviations from best security practices. The assessment was performed using automated analysis techniques to ensure accurate and actionable results.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	5	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DREU	Dynamic Regular Expression Usage	Unresolved
●	HCU	Hardcoded Credentials Usage	Unresolved
●	UIM	Unsafe IFS Modification	Unresolved
●	UMU	Unsafe MemoryMarshal.CreateSpan Usage	Unresolved
●	UCI	Unsanitized Command-Line Input	Unresolved

DREU - Dynamic Regular Expression Usage

Criticality	Minor / Informative
Location	src/Nethermind/Nethermind.Runner/wwwroot/.../bundle.js#L2,3
Status	Unresolved

Description

The codebase constructs a regular expression dynamically using a non-literal argument passed to RegExp(). This approach can expose the application to Regular Expression Denial-of-Service (ReDoS) attacks, as maliciously crafted patterns could trigger excessive backtracking and block the event loop. Since RegExp execution in JavaScript runs on the main thread, such behavior can cause performance degradation or complete service unavailability.

Recommendation

Avoid constructing regular expressions from variable or user-controlled input. Use hardcoded, precompiled regex literals whenever possible. If dynamic construction is unavoidable, validate or sanitize the input before use. Consider employing a regex safety library such as recheck to analyze and prevent unsafe patterns. Additionally, enforce input constraints (e.g., maximum length) and consider using timeouts or sandboxing mechanisms for regex evaluation.

HCU - Hardcoded Credentials Usage

Criticality	Minor / Informative
Location	src/Nethermind/Nethermind.Wallet/DevWallet.cs#L19
Status	Unresolved

Description

Storing credentials directly in source code exposes sensitive data and poses a significant security risk. Attackers with access to the codebase (e.g., through repository leaks, insider threats, or dependency compromises) could easily retrieve and misuse the credentials to gain unauthorized access to systems, wallets, or other resources.

```
private const string AnyPassword = "#DEV_ACCOUNT_NETHERMIND_ANY_PASSWORD#";
```

Recommendation

The team is advised to remove all hardcoded credentials from the codebase. Instead, the team could store passwords and other sensitive information in a secure external configuration mechanism such as environment variables, a secrets manager (e.g., Azure Key Vault, AWS Secrets Manager, or HashiCorp Vault), or encrypted configuration files. Ensure credentials are loaded securely at runtime and not exposed in logs or version control.

UIM - Unsafe IFS Modification

Criticality	Minor / Informative
Status	Unresolved

Description

The `hive-results.sh` script modifies the special shell variable IFS (Internal Field Separator) globally to change how word splitting occurs. Changing IFS globally can have unintended side effects on other parts of the script or on sourced scripts, potentially altering how input is parsed and leading to logic errors or command injection vulnerabilities.

```
IFS=$'\n' results=($(sort -f <<<"${tmp[*]}")); unset IFS
```

Recommendation

Modifying IFS globally should be avoided. Instead, the team could set it locally within a command substitution or read statement to limit its scope. Alternatively, the team could use dedicated utilities such as `cut` , `awk` , or `mapfile` to safely process input without altering global shell behavior.

UMU - Unsafe MemoryMarshal.CreateSpan Usage

Criticality	Minor / Informative
Location	src/Nethermind/Nethermind.Blockchain/Find/BlockParameter.cs#L240 src/Nethermind/Nethermind.Consensus.Ethash/EthashCache.cs#L34,108 src/Nethermind/Nethermind.Core/Bloom.cs#L227 src/Nethermind/Nethermind.Core/Crypto/Hash256.cs#L32,33,137 src/Nethermind/Nethermind.Core/Crypto/Keccak.cs#L60,67 src/Nethermind/Nethermind.Core/Crypto/KeccakCache.cs#L116,152
Status	Unresolved

Description

The codebase uses `MemoryMarshal.CreateSpan` (or `MemoryMarshal.CreateReadOnlySpan`) without validating the length argument. These methods do not perform bounds checking, meaning that an incorrect or untrusted length value could lead to out-of-bounds memory access. This may result in undefined behavior, data corruption, or application crashes, compromising the program's stability and reliability.

```
public Span<byte> AsSpan() => MemoryMarshal.CreateSpan(ref _element0, ByteLength);
public Span<byte> BytesAsSpan => MemoryMarshal.AsBytes(MemoryMarshal.CreateSpan(ref
Unsafe.AsRef(in _bytes), 1));
public ReadOnlySpan<byte> Bytes =>
MemoryMarshal.AsBytes(MemoryMarshal.CreateReadOnlySpan(ref Unsafe.AsRef(in _bytes),
1));
public Span<byte> Bytes => MemoryMarshal.AsBytes(MemoryMarshal.CreateSpan(ref
Unsafe.AsRef(in _hash256), 1));
...
```

Recommendation

Before calling `MemoryMarshal.CreateSpan` or `MemoryMarshal.CreateReadOnlySpan`, the team could ensure the provided length is validated against the actual buffer size or data source boundaries. Only use these methods when absolutely necessary and when the memory region's size and lifetime are well-defined and controlled. Consider safer alternatives such as Span or ReadOnlySpan constructors when possible.

UCI - Unsanitized Command-Line Input

Criticality	Minor / Informative
Status	Unresolved

Description

The superchain.py script uses unsanitized command-line input as a file path in `json.dump` and `open`. This introduces a Path Traversal vulnerability, where an attacker could manipulate file paths (e.g., using ../ sequences) to write or overwrite arbitrary files outside the intended directory. Such behavior can lead to unauthorized file modification, data corruption, or even remote code execution if critical files are affected.

```
with open(path.join(output_dir, "runner", f"{chainName}-{l1}.json"), "w+") as f:
    json.dump(runner, f, indent=2)

with open(path.join(output_dir, "chainspec", "dictionary"), "wb+") as
nethermind_dict_file:
    nethermind_dict_file.write(nethermind_dict.as_bytes())
```

Recommendation

The team is advised to validate and sanitize all file path inputs received from command-line arguments before using them in file operations. The team could restrict file access to specific directories and use safe path resolution methods such as `os.path.abspath` and `os.path.normpath` to ensure paths remain within allowed boundaries. Additionally, implement allowlists for valid filenames or extensions and avoid directly concatenating user input into file paths.

Summary

A comprehensive static analysis was conducted on the application's source code repository to evaluate its overall security posture. The assessment aimed to identify vulnerabilities, insecure coding practices, and configuration issues that could expose the application to potential threats. Using automated tools, the analysis covered core components of the repository, including backend logic and frontend implementations. The findings highlight key areas requiring remediation to enhance code security, reduce attack surface, and ensure adherence to industry best practices. Overall, the assessment provides actionable insights to guide the development team in improving the robustness and security of the codebase.

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