Hedera Blockchain Services “Modularization” Code review

Hedera Hashgraph

Version 2.0 – October 3, 2024

Prepared By
Alvaro Lorenzo
Juan Jose Frias

Prepared For
Joe Blanchard
Michael Heinrichs

1 Executive Summary

Synopsis

During the summer of 2024, Hedera Hashgraph engaged NCC Group to conduct a security assessment of Hedera. Hedera is a faster and more secure alternative to traditional blockchains. Using Hashgraph consensus, it ensures efficient transaction verifications. Users can access Hedera network services via APIs for tasks such as creating accounts, minting tokens, and executing smart contracts.

Scope

NCC Group's evaluation included:

- **hedera-node**: Implements the Hedera cryptocurrency, consensus, smart contract, and file services on the Platform.

Code Review was performed on one of the libraries hosted on https://github.com/hashgraph/hedera-services/tree/develop/hedera-node.

A first thorough review was performed for the version tagged as v0.49.7 and a follow-up assessment was conducted on version v0.54.0.

Key Findings

The assessment did not identify significant issues with the provided code base. A number of Low and Informational findings have been detailed.

Of significance was the presence of a number of secrets being present, all of them unused or part of test cases. However, it is still considered a good coding practice to remove unused secrets from code bases, even if they don't pose a risk.

Additionally, it was highlighted that the system allowed the creation of an unencrypted TCP port, which, if used, could expose sensitive information. The responsibility of establishing such an insecure connection would rest on users, and it is understood that this feature is part of a business decision for compatibility. However, consideration should be given in the future to disable this option as to further harden the solution.

Security Requirements

No evidence was found that the following security requirements were being violated by the reviewed source code:

- Security model rules highlighted on https://docs.hedera.com/hedera/core-concepts/smart-contracts/security
- Handling of system accounts as highlighted on https://github.com/hashgraph/hedera-services/blob/develop/hedera-node/docs/system-accounts-operations.md

However, it should be noted that the smart contract review was outside the scope of this assessment, as any source code outside the hedera-node element.

Strategic Recommendations

The code base should undergo an additional round of review once it reaches Release Candidate status. Additionally, any new functionality should be individually assessed during development once implemented.

2 Table of Contents

3 Document Control

Public Distribution Notice

Proprietary Information

Document Data

Data Classification	Client Confidential
Client Name	Hedera Hashgraph
Project Reference	E012962
Proposal Reference	O-211117
Document Title	Hedera Blockchain Services “Modularization” Code review
Author	Alvaro Lorenzo

Document History

Version	Issue Date	Issued by	Change Description
0.1	2024-06-04	Alvaro Lorenzo	Draft for NCC Group internal review only
0.2	2024-07-10	Mario Rivas	Revised QA
1.0	2024-07-10	Alvaro Lorenzo	Released to client
2.0	2024-10-03	Alvaro Lorenzo	Released to client

4 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
Code Comments Suggest Incomplete or Missing Code	New	CWC	Low
Unencrypted Communications Supported	New	MYU	Low
Private Keys Present in Test Code	New	4G3	Low
Empty "Case" Statement	New	3E2	Info
Empty "Catch" Statements	New	99E	Info
Unused or Deprecated Code	New	U96	Info
Hard-Coded Secrets in Test Code	New	AQW	Info
Docker Container Running as Root	New	KQR	Info

5 Risk Ratings

The table below gives a key to the ratings used throughout this report to provide a clear and concise risk scoring system.

It should be stressed that quantifying the overall business risk posed by any of the issues found in any test is outside our remit. This means that some risks may be reported as high from a technical perspective but may, as a result of other controls unknown to us, be considered acceptable.

Risk Rating	CVSS Score	Explanation
Critical	9.0 - 10	A vulnerability was discovered that has been rated as critical. This requires resolution as quickly as possible.
High	7.0 - 8.9	A vulnerability was discovered that has been rated as high. This requires resolution in the short term.
Medium	4.0 - 6.9	A vulnerability was discovered that has been rated as medium. This should be resolved as part of the ongoing security maintenance of the system.
Low	1.0 - 3.9	A vulnerability was discovered that has been rated as low. This should be addressed as part of routine maintenance tasks.
Info	0 - 0.9	A discovery was made that is reported for information. This should be addressed in order to meet leading practice.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

Rating	Description
High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

Rating	Description
High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

6 Finding Details

Low Code Comments Suggest Incomplete or Missing Code

Overall Risk	Low	**Finding ID**	NCC-E012962-CWC
Impact	Undetermined	**Component**	Code Review
Exploitability	Undetermined	**Category**	Other
		Status	New

Impact

Code comments that indicate that functionality is incomplete or broken may lead to scenarios where security critical areas, that may prevent a vulnerability or improve the security of the solution, are left missing and otherwise forgotten about.

Description

A search for code comments with the terms “FIXME” or “TODO” and derivatives thereof identified a number of instances where developers have noted incomplete or missing code. Such comments usually indicate that further work is needed to implement a feature or, more often, to fix bugs and introduce error checking. These code changes or additions are often not made, and these sections of code can lead to erroneous or vulnerable behavior.

```
// Create all the nodes in the merkle tree for all the services
        // TODO: Actually, we should reinitialize the config on each step along the migration
        ↳ path, so we should pass
        //       the config provider to the migration code and let it get the right version of
        ↳ config as it goes.
        onMigrate(state, deserializedVersion, trigger, metrics);
        if (trigger == EVENT_STREAM_RECOVERY) {
            // (FUTURE) Dump post-migration mod-service state
        }
```

The following keywords were searched for within the code base:

- TODO (168)
- FIXME (1)
- HACK (14)

Recommendation

Review all instances of FIXME, TODO, and other such comments identified above, and implement the necessary code changes and additions in order to improve the overall robustness of the application.[1]

Where changes are not able to be implemented in a timely manner, said missing or broken functionality should be formally tracked within the development team’s issue/bug tracking system.

Consider using static anaylsis to track when such comments are added to code to prevent their usage and ensure they are captured in the development backlog instead.

1. Todo Comments Considered Harmful: https://c2.com/cgi/wiki?TodoCommentsConsideredHarmful

Low

Unencrypted Communications Supported

Overall Risk	Low	**Finding ID**	NCC-E012962-MYU
Impact	Low	**Component**	Code Review
Exploitability	Low	**Category**	Cryptography
		Status	New

Description

The `ConfigDrivenNettyFactory` class implemented at `\hedera-services\hedera-node\hedera-mono-service\src\main\java\com\hedera\node\app\service\mono\grpc\ConfigDrivenNettyFactory.java` could spawn Netty Servers without TLS enabled.

This class, and the `builderFor` method it contains, were referenced once in the source code, from the `startOneNettyServer` method in the `NettyGrpcServerManager` class, which in turn was called from the `start` method of the same class and from test code. For clarity, the following is the code snippet that shows the mentioned functionality:

```
@Override
    public void start(int port, int tlsPort, Consumer println) {
        try {
            hookAdder.accept(new Thread(() -> terminateNetty(port, tlsPort, println)));
            server = startOneNettyServer(false, port, println, SLEEPING_PAUSE);
            tlsServer = startOneNettyServer(true, tlsPort, println, SLEEPING_PAUSE);
        } catch (FileNotFoundException fnfe) {
            tlsServer = null;
            String message = nettyAction("Could not start", true, tlsPort, false);
            log.warn("{} ({}).", message, fnfe.getMessage());
            println.accept(message);
        } catch (Exception e) {
            throw new IllegalStateException(e);
        }
    }

    Server startOneNettyServer(boolean sslEnabled, int port, Consumer println, Pause pause)
    ↳ throws IOException {
        println.accept(nettyAction("Starting", sslEnabled, port, true));

        NettyServerBuilder builder = nettyBuilder.builderFor(port, sslEnabled);
        bindableServices.forEach(builder::addService);
        Server nettyServer = builder.build();

        var retryNo = 1;
        final var n = Math.max(0, startRetries);
        for (; retryNo <= n; retryNo++) {
            try {
                nettyServer.start();
                break;
            } catch (IOException e) {
                final var summaryMsg = nettyAction("Still trying to start", sslEnabled, port,
                ↳ true);
                log.warn("(Attempts={}) {}", retryNo, summaryMsg, e);
                pause.forMs(startRetryIntervalMs);
            }
        }
        if (retryNo == n + 1) {
            nettyServer.start();
        }
```

As a result, the `NettyGrpcServerManager` class was exposing encrypted and plaintext ports when spawning the server.

While the decision of connecting through an encrypted or unencrypted port lays ultimately on the client establishing the connection, offering an unencrypted option increases the likeliness that a client might choose the unsafe port.

Recommendation

Remove the option to start Netty servers without TLS to ensure all connections are encrypted.

Location

- \hedera-services\hedera-node\hedera-mono-service\src\main\java\com\hedera\node\app\service\mono\grpc\ConfigDrivenNettyFactory.java

Low

Private Keys Present in Test Code

Overall Risk	Low	**Finding ID**	NCC-E012962-4G3
Impact	Undetermined	**Component**	Code Review
Exploitability	Low	**Category**	Data Exposure
		Status	New

Description

Several `.pem` and `.key` files containing private encryption keys were found in the code base. While these keys seemed to be used as part of test code, as opposed to the application, they should still be considered sensitive secrets and treated accordingly.

It should also be noted that in one specific instance, the key was in a path not explicitly marked as "test", which could point at it potentially being used for cases beyond testing.

This issue has been rated as Low on the premise that the impact is unknown.

Recommendation

Secrets should not be stored in code repositories. Instead, they should be stored separately, ideally in a password vault or storage mechanism designed for the purpose. The passwords currently stored in the code should be considered compromised and should be rotated as they are removed from the code base and transferred into the appropriate secret management platform.[2] [3] [4]

It would be prudent to integrate a static analysis process into the CI/CD pipeline or development process to ensure that credentials are kept out of code in the future. Any secrets that are committed to source and detected should be eliminated from the source should be rotated, as source control servers will retain the historical data.

Location

- \hedera-services\hedera-node\test-clients\yahcli\localhost\keys\account2.pem
- \hedera-services\hedera-node\test-clients\yahcli\localhost\keys\account55.pem
- \hedera-services\hedera-node\test-clients\devGenesisKeypair.pem
- \hedera-services\hedera-node\data\onboard\devGenesisKeypair.pem
- \hedera-services\hedera-node\hapi-utils\src\test\resources\vectors\genesis.pem
- \hedera-services\hedera-node\test-clients\src\main\resource\genesis.pem
- \hedera-services\hedera-node\hedera-mono-service\src\test\resources\test-hedera.key

2. Hardcoded Credentials Example: https://nakedsecurity.sophos.com/2014/04/03/is-amazon-hacking-our-apps-or-doing-us-all-a-security-favor/
3. SANS Top 25 Software Flaws: Number 11, Hardcoded Credentials: https://www.sans.org/blog/top-25-series-rank-11-hardcoded-credentials/
4. CWE-798: Use of Hard-Coded Credentials: https://cwe.mitre.org/data/definitions/798.html

Info

Empty “Case” Statement

Overall Risk	Informational	**Finding ID**	NCC-E012962-3E2
Impact	Low	**Component**	Code Review
Exploitability	Low	**Category**	Uncategorized
		Status	New

Description

A number of “case” statements without an associated action were discovered during the code review. While not in itself a security concern, this can indicate incomplete, unfinished, or untested code.

```
switch (sigType) {
               case ED25519 -> {
                  // Only match ED25519 signatures with ED25519 keys.
                  final var matchingKeyType = key.key().kind() == KeyOneOfType.ED25519;
                  // Valid ED25519 keys have a max length of 32 bytes.
                  final var validPrefixLength = prefix.length() <= ED25519_KEY_LENGTH;
                  if (matchingKeyType
                          && validPrefixLength
                          && key.ed25519OrThrow().matchesPrefix(prefix)) {
                      return pair;
                  }
               }
               case ECDSA_SECP256K1 -> {
                  // Only match ECDSA_SECP256K1 signatures with ECDSA_SECP256K1 keys.
                  final var matchingKeyType = key.key().kind() == KeyOneOfType.ECDSA_SECP256K1;
                  // Valid ECDSA_SECP256K1 keys have a max length of 33 bytes.
                  final var validPrefixLength = prefix.length() <= ECDSA_COMPRESSED_KEY_LENGTH;
                  if (matchingKeyType
                          && validPrefixLength
                          && key.ecdsaSecp256k1OrThrow().matchesPrefix(prefix)) {
                      return pair;
                  }
               }
               case CONTRACT, ECDSA_384, RSA_3072, UNSET -> {
                  // Skip these signature types. They never match.
               }
```

An example snippet from the application code is shown below (truncated for readability):

Empty “case” statements usually indicate areas where developers have not yet implemented functionality, fixed bugs, or introduced error checking. These changes are often left incomplete, and such sections of code often lead to erroneous or vulnerable behavior, though no such problems were noted during this assessment.

It was acknowledged by the engineering team that most these cases were explicitly included for clarity and future reference, so that developers know exactly for which cases actions should not be taken. For this reason, this issue is rated as Informational only.

Recommendation

Superfluous “case” statements should be removed or commented out as required.[5]

Location

- \hedera-node\hedera-app\src\main\java\com\hedera\node\app\signature\impl\SignatureExpanderImpl.java

5. Stack Overflow: How to Mark an Empty Conditional Block in Java: http://programmers.stackexchange.com/questions/120658/how-to-mark-an-empty-conditional-block-in-java

Info

Empty “Catch” Statements

Overall Risk	Informational	**Finding ID**	NCC-E012962-99E
Impact	Low	**Component**	Code Review
Exploitability	Low	**Category**	Auditing and Logging
		Status	New

Description

The codebase contained instances of empty catch blocks. Empty catch statements are generally considered poor coding practice, as the program is silently swallowing an error condition and then continuing execution. Failures that are met with no response should be avoided.

In general, if a catch statement has been reached, the program should at the very least log the error to a file, allowing the error to be identified later.

As an example, the following empty catch block was found within the code base:

```
private void addByteSource(@NonNull final ConfigurationBuilder builder, @NonNull final Bytes
↪ propertyFileContent) {
        requireNonNull(builder);
        requireNonNull(propertyFileContent);
        try {
            final var configurationList =
                    ServicesConfigurationList.PROTOBUF.parseStrict(propertyFileContent.toReadable
                    ↪ SequentialData());
            final var configSource = new
            ↪ SettingsConfigSource(configurationList.nameValueOrThrow(), 101);
            builder.withSource(configSource);
        } catch (ParseException | NullPointerException e) {
            // Ignore. This method may be called with a partial file during regular execution.
        }
    }
```

The engineering team clarified that these catches were explicitly coded to handled checked exceptions, which would prevent the program from compiling. In the above snippet, `ParseException` is required to either be caught or thrown in order for the program to compile.

However, it should be noted that `NullPointerException` is not a checked exception, which means that the above rationale would not apply to it.

The chances of exploiting this issue are negligible, and it has been reported for Informational purposes only.

Recommendation

Ensure that all conditions are met with a suitable response. A trivial example, in which a non-existent file is read, is given below:[6]

```
try
{
    using (StreamReader reader = new StreamReader(@"log.txt"))
    {
```

6. Why are empty catch blocks a bad idea?: https://stackoverflow.com/questions/1234343/why-are-empty-catch-blocks-a-bad-idea

```
        reader.ReadToEnd();
    }
}
catch (FileNotFoundException e)
{
    Console.WriteLine("File not found exception: {0}", e.ToString());
}
```

Location

- hedera-node\hedera-app\src\main\java\com\hedera\node\app\config\ConfigProviderImpl.java
- hedera-node\hedera-mono-service\src\main\java\com\hedera\node\app\service\mono\legacy\core\jproto\JKey.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\junit\RecordStreamAccess.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\junit\SubProcessHapiTestNode.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\HapiPropertySource.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\HapiSpecOperation.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\PropertySource.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\infrastructure\HapiSpecRegistry.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\infrastructure\selectors\RandomSelector.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\transactions\crypto\HapiCryptoTransfer.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\transactions\token\HapiTokenCreate.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\transactions\token\HapiTokenMint.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\utilops\ProviderRun.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\spec\utilops\RunLoadTest.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\suites\freeze\CommonUpgradeResources.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\suites\issues\Issue305Spec.java
- hedera-node\test-clients\src\main\java\com\hedera\services\bdd\suites\utils\validation\ValidationScenarios.java

- hedera-node\test-clients\src\yahcli\java\com\hedera\services\yahcli\config\ConfigUtils.java

Info

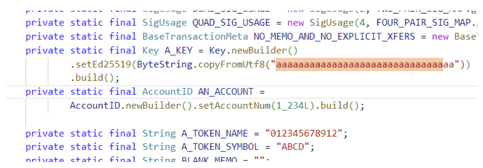
Unused or Deprecated Code

Overall Risk	Informational	**Finding ID**	NCC-E012962-U96
Impact	Undetermined	**Component**	Code Review
Exploitability	Undetermined	**Category**	Unwanted Software
		Status	New

Description

During the review of the source code, instances of what seemed like hard-coded secrets were identified and forwarded to the engineering team. After reviewing the provided evidence, the team pointed out that it was unused code that would be removed in future refactoring passes.

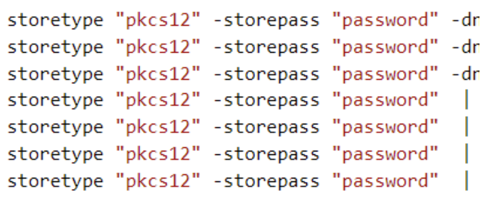
Examples of the provided snippets can be seen below, and a list of affected files in the Location subsection below:



```
private static final SigUsage QUAD_SIG_USAGE = new SigUsage(4, FOUR_PAIR_SIG_MAP.
private static final BaseTransactionMeta NO_MEMO_AND_NO_EXPLICIT_XFERS = new Base
private static final Key A_KEY = Key.newBuilder()
        .setEd25519(ByteString.copyFromUtf8("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"))
        .build();
private static final AccountID AN_ACCOUNT =
        AccountID.newBuilder().setAccountNum(1_234L).build();

private static final String A_TOKEN_NAME = "012345678912";
private static final String A_TOKEN_SYMBOL = "ABCD";
private static final String BLANK_MEMO = "";
```

Figure 1: Key set from a string of multiple "a" characters



```
storetype "pkcs12" -storepass "password" -dr
storetype "pkcs12" -storepass "password" -dr
storetype "pkcs12" -storepass "password" -dr
storetype "pkcs12" -storepass "password"  |
storetype "pkcs12" -storepass "password"  |
storetype "pkcs12" -storepass "password"  |
storetype "pkcs12" -storepass "password"  |
```

Figure 2: Passphrase for key stores set to "password"

Given that this code is unused and scheduled to be removed, this issue has been rated as Informational only.

Recommendation

Remove all unused code and files from the repository. When refactoring, consider adding elements as needed as opposed to trimming unused elements, to prevent missed pieces of code.

Ensure that the listed secrets are not used elsewhere or pose a risk if disclosed. If used, replace them with stronger ones and ensure that they are not present in the repository.

Location

- \hedera-services\hedera-node\cli-clients\src\test\java\com.hedera.services.cli.sign.test\TestUtils.java
- \hedera-services\hedera-node\data\keys\generate.sh
- \hedera-services\hedera-node\data\keys\generate.bat
- \hedera-services\hedera-node\hapi-fees\src\main\java\com\hedera\node\app\hapi\fees\pricing\BaseOperationUsage.java
- \hedera-services\hedera-node\hapi-utils\src\main\java\com\hedera\node\app\hapi\utils\exports\FileSignTool.java

Info

Hard-Coded Secrets in Test Code

Overall Risk	Informational	**Finding ID**	NCC-E012962-AQW
Impact	N/A	**Component**	Code Review
Exploitability	N/A	**Category**	Data Exposure
		Status	New

Description

Inspection of the code revealed that secrets were hard coded within code and configuration files. The presence of hard coded secrets means that anyone with access to the affected source code repositories or the compiled binaries and scripts could potentially gain control over the affected services and environments by using those secrets. In addition, hard coded secrets are not changeable without modifying the code. This presents an additional security problem because it is no longer possible for a user or administrator of the system to change the secret in question without potentially breaking some aspect of functionality. Proper rotation of credentials becomes more difficult if they are spread across different code bases, including various versions and branches of the same code base. Furthermore, propagation of such secrets to source control places a significant security burden on the development & build of infrastructure and makes the insider threat scenario almost impossible to mitigate.

The code base included test functionality which used a simple passphrase to encrypt `.pem` files:

```
default.payer.name=DEFAULT_PAYER
default.payer.pemKeyLoc=src/main/resource/genesis.pem
#default.payer.pemKeyLoc=previewtestnet-account2.pem
#default.payer.pemKeyLoc=stabletestnet-account50.pem
#default.payer.pemKeyLoc=mainnet-account950.pem
default.payer.pemKeyPassphrase=
default.proxy=n/a
default.queueSaturation.ms=100
default.realm=0
default.receiverSigRequired=false
```

This issue has been rated as Informational only, based on it affecting only test code.

Recommendation

Secrets should not be stored in code repositories. Instead, they should be stored separately, ideally in a password vault or storage mechanism designed for the purpose. The passwords currently stored in the code should be considered compromised and should be rotated as they are removed from the code base and transferred into the appropriate secret management platform.[7] [8] [9]

It would be prudent to integrate a static analysis process into the CI/CD pipeline or development process to ensure that credentials are kept out of code in the future. Any

7. Hardcoded Credentials Example: https://nakedsecurity.sophos.com/2014/04/03/is-amazon-hacking-our-apps-or-doing-us-all-a-security-favor/
8. SANS Top 25 Software Flaws: Number 11, Hardcoded Credentials: https://www.sans.org/blog/top-25-series-rank-11-hardcoded-credentials/
9. CWE-798: Use of Hard-Coded Credentials: https://cwe.mitre.org/data/definitions/798.html

secrets that are committed to source and detected should be eliminated from the source should be rotated, as source control servers will retain the historical data.

Location

- \hedera-services\hedera-node\test-clients\src\main\resource\spec-default.properties
- \hedera-services\hedera-node\test-clients\puv\localhost-entities\puvCatBeneficiary.yaml
- \hedera-services\hedera-node\test-clients\puv\localhost-entities\puvCatToken.yaml
- \hedera-services\hedera-node\test-clients\puv\localhost-entities\puvTacoBeneficiary.yaml
- \hedera-services\hedera-node\test-clients\puv\localhost-entities\puvTacoToken.yaml
- \hedera-services\hedera-node\test-clients\puv\localhost-entities\puvTreasury.yaml

Info

Docker Container Running as Root

Overall Risk	Informational	**Finding ID**	NCC-E012962-KQR
Impact	Medium	**Component**	Code Review
Exploitability	Low	**Category**	Configuration
		Status	New

Description

Several Docker images did not make use of the USER directive to specify an account for the operation of containers. As a result, those containers will run as the 'root' user (this is the default setting) which could make it easier for an attacker with access to the containers to break out to the underlying host. A less privileged non-root user should be specified instead.

Since it was unknown whether User Namespaces were enabled for these instances, and this finding has been raised as informational

Recommendation

Review the affected image files and modify them to specify a user for operation of containers based on them. All Docker images should make use of the USER directive, or switch to another user as part of the CMD or ENTRYPOINT commands, to ensure that they run as a nominated user and not as the default root user.

For added security, consider adopting user namespaces to mitigate the risk of container breakout issues. This should be considered for applications which require running as root.

This can be achieved with the following command: [10]

```
$ docker run --user <user> <image>
```

Location

- \hedera-services\hedera-node\test-clients\yahcli\Dockerfile
- \hedera-services\hedera-node\test-clients\validation-scenarios\Dockerfile
- \hedera-services\hedera-node\docker\Dockerfile
- \hedera-services\hedera-node\infrastructure\docker\containers\production\network-node-haveged\Dockerfile
- \hedera-services\hedera-node\infrastructure\docker\containers\production\network-node-base\Dockerfile
- \hedera-services\hedera-node\infrastructure\docker\containers\local-node\network-node-base\Dockerfile
- \hedera-services\hedera-node\infrastructure\docker\containers\local-node\network-node-haveged\Dockerfile

10. **Docker Article: USER Directive:** https://docs.docker.com/engine/reference/builder/#user

7 Contact Info

The team from NCC Group has the following primary members:

- Alvaro Lorenzo – Executive Principal Security Consultant
  alvaro.lorenzo@nccgroup.com
- Juan Jose Frias – Senior Security Consultant
  juanjose.frias@nccgroup.com

The team from Hedera Hashgraph has the following primary members:

- Michael Heinrichs – Hedera Hashgraph
  michael@hashgraph.com
- Joe Blanchard – Hedera Hashgraph
  joe@hashgraph.com