
ARTIFACT SPECIMENS

What actually arrives, field by field.

Vendors describe their outputs. This document shows them. Every assurance artifact our products emit is reproduced below as a labelled specimen so an assessor, an Information System Security Officer, or a contracting officer can see the exact structure before signing anything.

Read this first. Every block on this page is a *specimen of structure*. The field names, relationships, and formats are real. The values are illustrative and are written in angle brackets or as obvious sample text. Nothing here is output from a live system, and no value on this page should be treated as evidence about any deployment. Populated artifacts from a running instance are provided under a mutual non-disclosure agreement or during a pilot.

01 Evidence Pack manifest

02 Software Bill of Materials

03 AI Bill of Materials

04 Model card

05 Data card

06 Control mapping

07 POA&M row

08 Audit record entry

09 How to verify a pack

10 How to request populated artifacts

SPECIMEN 01

Evidence Pack manifest

The manifest is the root of the pack. It binds a named artifact to the evidence generated about it, and it is what an assessor verifies first. Produced by DevSecOps Evidence Builder and, in the same shape, by Vigil and Halo Security Platform.

SPECIMEN

Structure is accurate. Values are illustrative.

```

{
  "manifestVersion": "1.0",
  "packType": "evidence",
  "generatedAt": "<RFC 3339 timestamp>",
  "generatedBy": { "product": "DevSecOps Evidence Builder", "version": "<semver>" },
  "subject": {
    "kind": "container-image | repository | system",
    "identifier": "<registry reference or repository URL>",
    "digest": "sha256:<64 hex characters>" // the artifact actually assessed
  },
  "tenant": { "id": "<tenant identifier>", "classification": "<label>" },
  "contents": [
    { "path": "sbom/cyclonedx.json", "sha256": "<hash>", "mediaType": "application/vnd.cyclonedx+json" },
    { "path": "findings/normalized.json", "sha256": "<hash>", "mediaType": "application/json" },
    { "path": "controls/mapping.json", "sha256": "<hash>", "mediaType": "application/json" },
    { "path": "poam/poam.csv", "sha256": "<hash>", "mediaType": "text/csv" },
    { "path": "remediation/patches.diff", "sha256": "<hash>", "mediaType": "text/x-diff" }
  ],
  "attestation": {
    "predicateType": "<in-toto style predicate identifier>",
    "chain": ["repository", "sbom", "findings", "remediation", "controlMapping"],
    "signature": { "algorithm": "<signature algorithm>", "keyId": "<key identifier>", "value": "<signature>" }
  },
  "verification": {
    "offline": true, // verifiable with no network access
    "instructions": "verify/README.txt"
  }
}

```

WHAT AN ASSESSOR CAN DO WITH IT

- ✓ Confirm the digest in `subject` matches the artifact actually deployed
- ✓ Recompute every hash in `contents` independently
- ✓ Verify the signature without contacting us and without network access
- ✓ Follow the attestation chain from source to control mapping

WHAT IT DELIBERATELY DOES NOT CONTAIN

- ✓ Any credential, key material, or token
- ✓ Customer operational data of any kind
- ✓ Our remediation templates, mapping logic, or scoring thresholds
- ✓ Anything above the classification the deployment is authorized to hold

SPECIMEN 02

Software Bill of Materials

Emitted in CycloneDX and SPDX. Neither format is ours, both are published standards, and the point of showing it is that the component entries carry the traceability an assessor needs back to findings.

SPECIMEN

CycloneDX shape. Values are illustrative.

```

{
  "bomFormat": "CycloneDX",
  "specVersion": "<spec version>",
  "serialNumber": "urn:uuid:<uuid>",
  "version": 1,
  "metadata": {
    "timestamp": "<RFC 3339 timestamp>",
    "component": { "type": "container", "name": "<image name>", "version": "<tag>" }
  },
  "components": [
    {
      "type": "library",
      "name": "<component name>",
      "version": "<version>",
      "purl": "pkg:<ecosystem>/<namespace>/<name>@<version>",
      "licenses": [ { "license": { "id": "<SPDX license identifier>" } } ],
      "hashes": [ { "alg": "SHA-256", "content": "<hash>" } ],
      "supplier": { "name": "<supplier>" }
    }
  ],
  "dependencies": [
    { "ref": "<component purl>", "dependsOn": [ "<component purl>" ] }
  ]
}

```

Federal context: OMB M-26-05, issued 23 January 2026, rescinded the government-wide secure software attestation common form previously set by M-22-18 and M-23-16, and directs agencies to set their own risk-based assurance requirements. Agencies may still require a self-attestation and an SBOM. Producing both continuously, rather than on demand, is the point of the product.

SPECIMEN 03

AI Bill of Materials

An SBOM covers software components. An AI-BOM covers the parts of an AI system that an SBOM does not reach: the models, their provenance, the data lineage claimed for

them, and the inference path. Produced by AI Acquisition Assurance and by Aegis as part of an assessment.

SPECIMEN

Structure is accurate. Values are illustrative.

```
{
  "aiBomVersion": "1.0",
  "system": { "name": "<system name>", "owner": "<program office>", "classification": "<label>"
},
  "models": [
    {
      "name": "<model name>",
      "version": "<version or weights digest>",
      "weightsDigest": "sha256:<hash>",
      "license": "<model licence identifier>",
      "provenance": { "source": "<where the weights came from>", "retrievedAt": "<timestamp>" },
      "hostingMode": "self-hosted-in-boundary | authorized-managed-service",
      "modelCard": "cards/<model>.model-card.json"
    }
  ],
  "data": [
    { "role": "retrieval-corpus | fine-tuning | evaluation",
      "dataCard": "cards/<dataset>.data-card.json",
      "classification": "<label>",
      "egress": "none" }
  ],
  "inferencePath": {
    "externalCalls": false, // true only where an authorized managed
service is used
    "boundary": "<authorization boundary identifier>",
    "guardrails": ["input-sanitization", "output-filtering", "citation-or-suppress"]
  },
  "assessments": [
    { "type": "adversarial", "report": "assessments/<id>.report.pdf", "date": "<date>" }
  ]
}
```

Model card

A model card that lists no limitations is a marketing document. The fields below are the ones we require to be populated before a model card is considered complete, and the limitations section is mandatory.

SPECIMEN

Required fields. Values are illustrative.

Model	<name and version>
Weights digest	sha256:<hash>
Licence	<licence identifier and any use restrictions>
Hosting	self-hosted in boundary authorized managed service
Intended use	<the task the model is authorized to perform in this system>
Out of scope use	<uses explicitly not authorized>
Users	<who interacts with the output, and under what supervision>
Evaluation	
Suite	<evaluation suite identifier and version>
Conditions	<environment the evaluation was run in>
Results	<measured results, with the date measured>
Limitations	// mandatory, never empty
	<known failure modes>
	<conditions under which output degrades>
	<populations, inputs, or domains where performance is not characterized>
Human oversight	<the review step that stands between output and action>
Review	Owner <role> Last reviewed <date> Next review <date>

Data card

The companion to the model card. It answers the question an Information System Security Officer asks first, which is where the data came from and where it is allowed to go.

SPECIMEN

Required fields. Values are illustrative.

Dataset	<name and version>
Role	retrieval corpus fine-tuning evaluation
Classification	<label and handling caveats>
Provenance	<origin of the data and the authority under which it was obtained>
Collection	<how and when it was assembled>
Contents	<what the records represent, at a descriptive level>
Personal data	<present or absent; if present, the authority and the safeguards>
Residency	<where it is stored, and in which authorization boundary>
Egress	none // no path off the boundary exists for this dataset
Retention	<period, and the deletion mechanism>
Access	<roles permitted, and the enforcement point>
Known gaps	<coverage the dataset does not have>
Review	Owner <role> Last reviewed <date>

Control mapping

This is the join between engineering output and the assessor's baseline. One control is usually satisfied by several pieces of evidence, and one piece of evidence usually serves

several controls, which is why Vigil models it as a graph rather than a list.

SPECIMEN

Structure is accurate. Control identifiers shown are real NIST SP 800-53 Rev. 5 identifiers; the evidence references and statuses are illustrative.

```
{
  "baseline": "NIST SP 800-53 Rev. 5",
  "overlay": "<agency or programme overlay, if any>",
  "mappings": [
    {
      "control": "CM-8", // System Component Inventory
      "implementationStatus": "implemented | partially implemented | planned | not applicable",
      "evidence": [
        { "ref": "sbom/cyclonedx.json", "sha256": "<hash>", "generatedAt": "<timestamp>",
"freshness": "<age>" }
      ],
      "narrative": "<how the evidence satisfies the control, in the assessor's language>",
      "inheritedFrom": "<platform or provider, where the control is inherited>"
    },
    {
      "control": "RA-5", // Vulnerability Monitoring and Scanning
      "implementationStatus": "implemented",
      "evidence": [
        { "ref": "findings/normalized.json", "sha256": "<hash>", "generatedAt": "<timestamp>",
"freshness": "<age>" },
        { "ref": "poam/poam.csv", "sha256": "<hash>", "generatedAt": "<timestamp>",
"freshness": "<age>" }
      ],
      "narrative": "<...>"
    }
  ],
  "coverage": { "controlsInBaseline": "<n>", "controlsWithEvidence": "<n>", "stale": "<n>" }
}
```

Export to OSCAL is available where the receiving programme uses it. We do not claim OSCAL validation against a specific profile without naming the profile.

POA&M row

Delivered as CSV so it imports into the tracker a programme already runs. The columns below are the ones we populate; a programme with a different template tells us the column names and we emit those instead.

SPECIMEN

Column structure is accurate. Values are illustrative.

```
weakness_id,control,weakness_description,source,detected_at,severity,
exploitability,asset,owner_role,remediation,evidence_ref,evidence_sha256,
scheduled_completion,status,risk_acceptance_ref
```

example row, illustrative only

```
<id>,RA-5,<short description of the weakness>,<scanner or assessment>,<date>,
<severity>,<exploitability signal>,<affected component purl>,<owning role>,
<proposed remediation>,findings/normalized.json,<hash>,<date>,open,<reference if accepted>
```

Scheduled completion dates are derived from the service level agreement a programme sets, not from a default we impose. A vendor that ships you due dates without asking for your remediation policy has guessed them.

Audit record entry

Every access, inference, and data flow across the portfolio writes an entry in this shape. Entries are hash chained, so removing or altering one breaks verification of every entry after it.

SPECIMEN

Structure is accurate. Values are illustrative.

```

{
  "seq": "<monotonic sequence number>",
  "timestamp": "<RFC 3339 timestamp>",
  "tenant": "<tenant identifier>",
  "actor": { "type": "user | service | agent", "id": "<identifier>", "authn": "<method>" },
  "action": "<verb, for example read, classify, infer, export, publish>",
  "resource": { "kind": "<type>", "id": "<identifier>", "classification": "<label>" },
  "decision": { "outcome": "allow | deny | transform", "policy": "<policy identifier and version>" },
  "inference": {                                     // present only when a model was called
    "model": "<model name and version>",
    "hostingMode": "self-hosted-in-boundary | authorized-managed-service",
    "groundingRefs": ["<document identifiers the answer was grounded in>"]
  },
  "prevHash": "sha256:<hash of the previous entry>",
  "entryHash": "sha256:<hash of this entry>"
}

```

Where an audit write fails, the operation it describes fails with it. An action that cannot be recorded does not proceed.

— VERIFICATION

How to verify a pack without involving us.

An evidence package you cannot check yourself is a claim, not evidence. Every pack ships with the instructions below, and all of it works with no network access and no software from us.

1

Recompute the hashes

Hash every file listed in the manifest with any standard tool and compare against the recorded value. A mismatch means the pack has been altered since it was signed.

2

Check the signature

Verify the manifest signature against the public key delivered separately from the pack. The verification step is offline by design.

3

Match the subject

Confirm the digest in the subject block is the digest of the artifact you actually deployed. This is the step that catches evidence produced about a different build.

If any of those three steps fails, the pack should be rejected, including one of ours. We would rather be caught by your process than trusted by it.