KubeHuddle 2023

# GAIN BETTER VISIBILITY INTO CONTAINER IMAGE SIGNATURES


## IVAN WALLIS

Architect, Cloud Native Solutions at Venafi


May 17th and 18th, 2023

Toronto, ON

Canada

www.kubehuddle.com

# Agenda

1. How Code Signing Works
2. Introduction to SigScan
3. Signature Discovery for Containers
4. Signature Discovery for Files
5. Demo
6. Future Work

# Motivation

- Container Image security is under the spotlight as the industry continues to focus on modern software supply chain security
- Risks to running unsigned container images or images with unknown signatures (self-signed, standalone software keypairs, etc.).
- Current OSS tooling (e.g., sigstore/cosign) has limited enterprise key management support, which means users can generate local software keys or rely non-compliant code signing certificates.
- Enterprises lack visibility into what applications, container images, artifacts have been signed.

- Signature Discovery:
  - **Sigscan** can scan an OCI-compliant registry for cosign/notaryv2 signed container images, and report to screen or output to JSON. It can also scan the file system for signed artifacts.
- Inspired by Jetstack/paranoia tool for detecting certificates in container images

https://github.com/venafi/sigscan

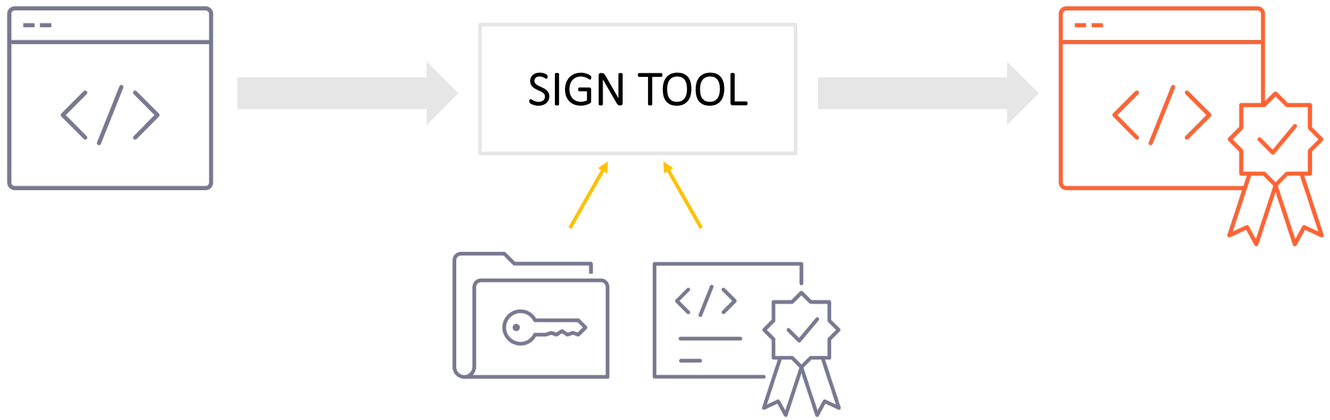
QUICK REVIEW

# How Code Signing Works

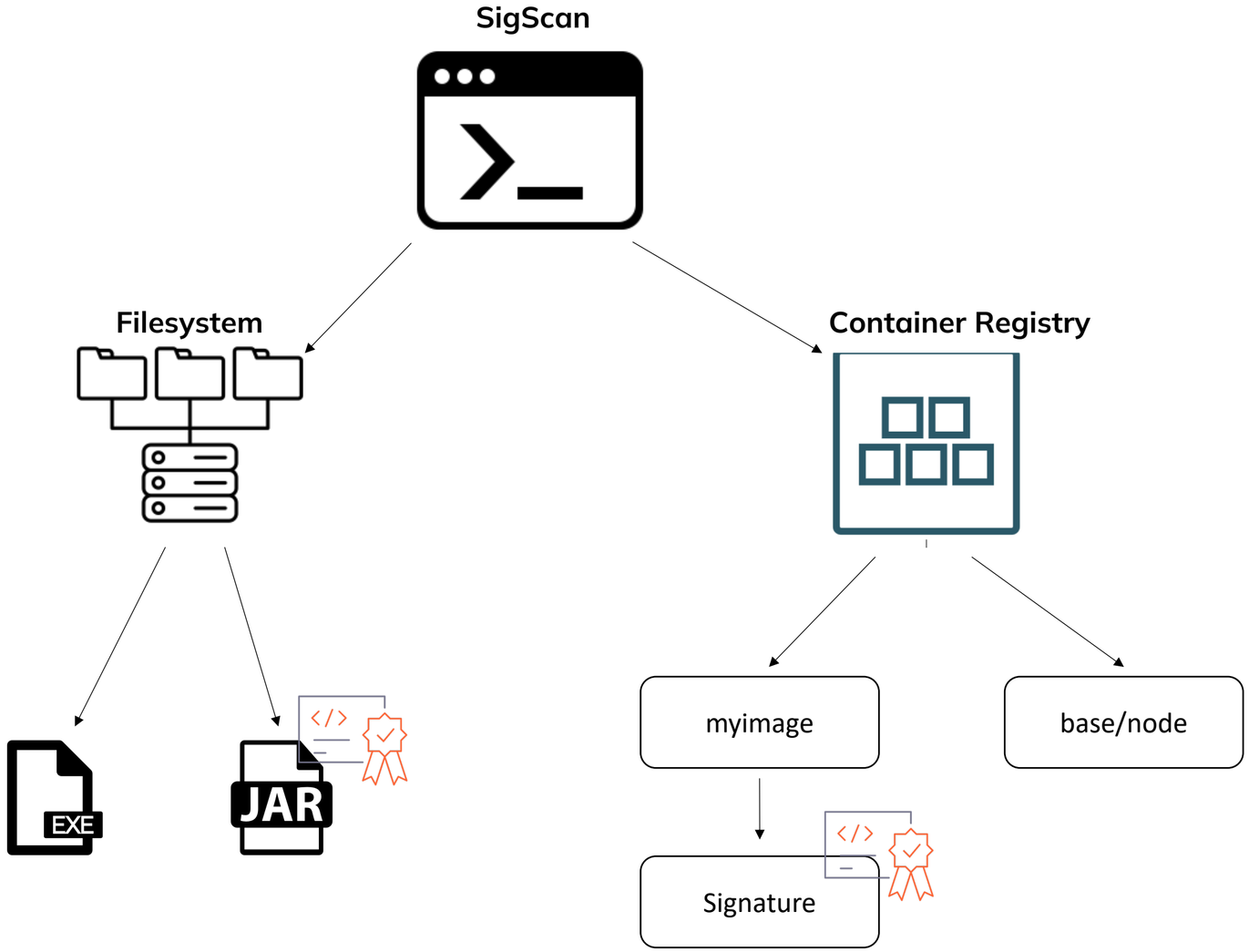
To sign an artifact / source code / software executable / macro / etc:

The signer uses the sign tool provided with their software development environment. The code sign tool does the following:

1. A hash is generated on the file(s)
2. The hash is encrypted with a private key (signature)
3. The code + signature + code signing certificate are are bundled together



# Sigscan Architecture



# OSS Resources Leveraged by SigScan

- Registry:
  - Oras Go -> https://github.com/oras-project/oras-go
  - Google Go Container Registry -> https://github.com/google/go-containerregistry

- Container Image Signing
  - Sigstore/cosign -> https://github.com/sigstore/cosign
  - Notary v2 Project -> https://github.com/notaryproject/notation

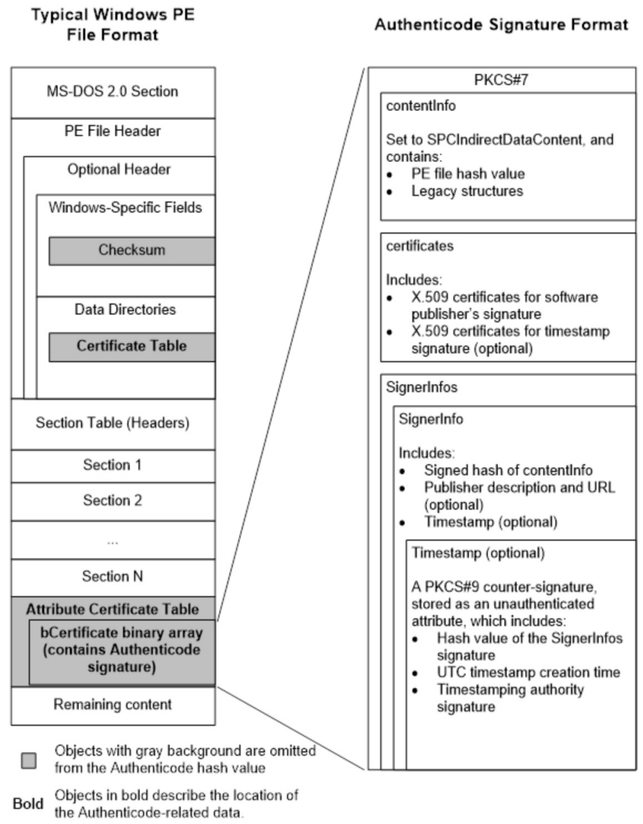
- Filesystem:
  - SAS Software Relic -> https://github.com/sassoftware/relic

# History of (Traditional) Signatures

- Embedded
  - Traditional applications
    - EXE
    - JAR
    - XML
    - …
- Detached
  - RedHat Simple Signing

```
[larizmen@localhost ~]$ cat
/home/larizmen/tmp/imagesignature/luisarizmendi/blog-django-
py@sha256\=e6d1995b66bd856a61c78cf28715a762459803a327cf3638b5b1033e41
95903f/signature-1

▯▯▯▯▯▯▯▯▯t▯▯~▯▯E▯▯▯$1▯e▯FV+%e▯d&'▯(YU+e▯▯▯d▯T▯▯)▯▯ɂE▯E▯i▯E▯ylJVJ▯▯▯▯z
▯▯▯9▯▯ʰnE▯U▯▯y)▯▯I9▯▯)Y▯y▯▯▯▯V▯zzř▯y▯)J▯:J▯▯▯▯H▯▯&▯e▯▯▯▯d▯)▯▯▯▯F▯fVf&▯
▯▯)▯I&▯▯I▯▯F▯▯)▯▯▯Ff▯&▯i▯)iI@▯fdb▯hjh▯▯h▯▯j▯j▯▯d▯b`▯fa▯▯▯▯▯▯Ē▯▯▯d▯▯▯▯
▯▯▯"▯sKJ▯RA▯▯

J2▯▯ ▯M.J⋆.B▯1▯3▯3T▯▯▯

t]bn▯▯▯▯▯▯▯▯▯▯▯ymm'▯1

#▯▯▯"▯▯▯7▯▯ ł▯k▯▯++(L▯8`"▯ ▯Ŷ▯▯%I+2▯y▯▯J3▯▯ᵢ▯▯▯#_$▯▯y▯d▯8▯1▯Y▯▯

▯▯▯▯▯▯▯▯Z^▯:▯▯k▯▯omMx▯␣▯▯▯▯R▯O▯_/I]▯c▯▯M▯'1▯▯R▯▯.▯▯yýjZ▯▯n▯▯l▯▯▯F▯▯▯▯=
▯▯▯▯▯▯▯▯▯▯▯3▯S▯Z\▯▯f▯]▯g▯▯N=7K舷N▯▯¥▯FZ▯qz

▯L▯▯V>▯0S▯▯▯▯⋆[C_~▯▯▯n▯|g▯K!]▯O▯▯#▯▯\(▯Wp,▯▯▯c▯▯▯[larizmen@localhost
~]$
```

```
ivan.wallis@07WKSMAC150894 sigscan % tar tvf test/tempdir1/hello-signed.jar
drwxrwxr-x  0 0      0           0 Feb 16 17:14 META-INF/
-rw-rw-r--  0 0      0         150 Feb 16 17:14 META-INF/MANIFEST.MF
-rw-rw-r--  0 0      0         316 Feb 16 17:14 META-INF/VSIGN-RSA2048-CERT.SF
-rw-rw-r--  0 0      0        1993 Feb 16 17:14 META-INF/VSIGN-RSA2048-CERT.RSA
-rw-rw-r--  0 0      0          12 Mar 27  2017 hello.txt
```



# Sigstore cosign signature discovery (1/2)

- Signature Specification
  - OCI object layout as follows:
    - payload -> Contents of signed data in byte-form
    - mediaType -> application/vnd.dev.cosign.simplesigning.v1+json
    - signature -> Base64-encoded signature
    - certificate -> OPTIONAL -> PEM-encoded X.509 certificate
    - chain -> OPTIONAL -> PEM-encoded x.509 certificate chain
    - bundle -> OPTIONAL -> JSON formatted, useful for offline verification of Tlog (i.e. Rekor)
    - rfc3161timestamp -> OPTIONAL -> JSON Formatted RFC 3161 timestamp from a TSA
  - Image signatures are stored in an OCI registry and are designed to make use of existing specifications.

# Sigstore cosign signature discovery (2/2)

- Discovery
  - Tag-based
    - Signatures are stored in an OCI registry in a predictable location, addressable by tag
    - If object is referenced by tag, tag must resolve to a digest (sha256:abcdef…)
      - Replace `:` with `-`
      - Append the `.sig` suffix
    - Signature is base64 encoded and stored as annotatio

```
"annotations": {
  "dev.cosignproject.cosign/signature": "MEUCIB
    /lWViyPxsJsbnIHj86sSbb7L3qvpEFoAiEA2ZChO
    /67CuAPQKJLBVsAc7bs9hBK8RpsdfjBsByGKJM="
}
```


```
"annotations": {
  "dev.sigstore.cosign/certificate": "-----BEGIN CERTIFICATE
    -----\nMIICrjCCAjSgAwIBAgIUAM4mURWUSkg06fmHmFfTmerYKaUwCgYIKoZI
    zj0EAwMw\nKjEVMBMGA1UEChMMc2lnc3RvcmUuZGV2MREwDwYDVQQDEwhzaWdzd
    G9yZTAeFw0y\nMTA0MDExNTU5MDZaFw0yMTA0MDExNjE4NTlaMDoxGzAZBgNVBA
    oMEmRsb3JlbmNA\nZ29vZ2xlLmNvbTEbMBkGA1UEAwwSZGxvcmVuY0Bnb29nbGU
    uY29tMFkwEwYHKoZI\nzj0CAQYIKoZIzj0DAQcDQgAE3R0ZtpfBd3Y8DaXuB1gM
    8JPlhsDIEfXO/WsMJEN1\n4hEn8wajX2HklqL7igZPFICv6tBUGylIHp2mTH2Nh
    v38mqOCASYwggEiMA4GA1Ud\nDwEB
    /wQEAwIHgDATBgNVHSUEDDAKBggrBgEFBQcDAzAMBgNVHRMBAf8EAjAAMB0G\nA
    1UdDgQWBBTy3UWIop0bNrdNgSrVHHD10qSASTAfBgNVHSMEGDAWgBTIxR0AQZok
    \nKTJRJOsNrkrtSgbT7DCBjQYIKwYBBQUHAQEEgYAwfjB8BggrBgEFBQcwAoZwa
    HR0\ncDovL3ByaXZhdGVjYS1jb250ZW50LTYwM2ZlN2U3LTAwMDAtMjIyNy1iZj
    c1LWY0\nZjVlODBkMjk1NC5zdG9yYWdlLmdvb2dsZWFwaXMuY29tL2NhMzZhMWU
    5NjI0MmI5\nZmNiMTQ2L2NhLmNydDAdBgNVHREEFjAUgRJkbG9yZW5jQGdvb2ds
    ZS5jb20wCgYI\nKoZIzj0EAwMDaAAwZQIwC15Gtd9F6W9lmJuoXMym9DfWlBpK5
    HEPak38WPXqowRp\n6p+2
    /3jSLkFT5Nn5fuISAjEAouVlX4zH2rlkfg45HnDJax7o6ZV+E0
    /6BdAms44D\nEj6T/GLK6XJSB28haSPRWB7k\n-----END CERTIFICATE
    -----\n"
}
```

# Artifacts

- OCI Artifact Manifest Specification (OCI 1.1):
  - Define content addressable artifacts in order to store them along side container images
  - Examples of artifacts that may be stored along with container images are Software Bill of Materials (SBOM), Digital Signatures, Provenance data, Supply Chain Attestations, scan results, and Helm charts.

- OCI Image Manifest
  - Existing image spec already supports artifact representations (and their signatures)

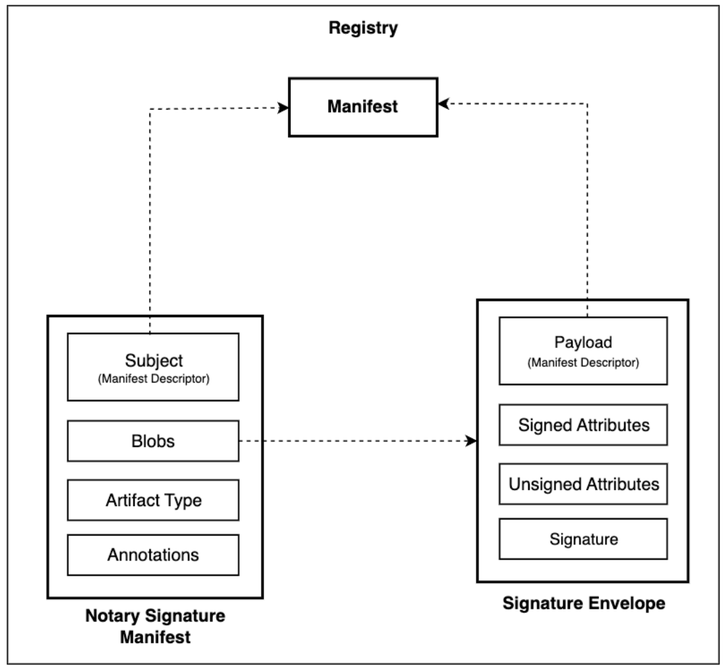
- OCI 1.1 is in RC phase:
  - Hasn't been released  with an active discussion on whether to leverage existing **image.manifest** or enable **artifact.manifest**
  - **Update -> PR #999 removed the artifact manifest media type**

# Referrer API

| ID | Method | API Endpoint | Success | Failure |
|---|---|---|---|---|
| end-1 | `GET` | `/v2/` | `200` | `404` / `401` |
| end-2 | `GET` / `HEAD` | `/v2/<name>/blobs/<digest>` | `200` | `404` |
| end-3 | `GET` / `HEAD` | `/v2/<name>/manifests/<reference>` | `200` | `404` |
| end-4a | `POST` | `/v2/<name>/blobs/uploads/` | `202` | `404` |
| end-4b | `POST` | `/v2/<name>/blobs/uploads/?digest=<digest>` | `201` / `202` | `404` / `400` |
| end-5 | `PATCH` | `/v2/<name>/blobs/uploads/<reference>` | `202` | `404` / `416` |
| end-6 | `PUT` | `/v2/<name>/blobs/uploads/<reference>?digest=<digest>` | `201` | `404` / `400` |
| end-7 | `PUT` | `/v2/<name>/manifests/<reference>` | `201` | `404` |
| end-8a | `GET` | `/v2/<name>/tags/list` | `200` | `404` |
| end-8b | `GET` | `/v2/<name>/tags/list?n=<integer>&last=<integer>` | `200` | `404` |
| end-9 | `DELETE` | `/v2/<name>/manifests/<reference>` | `202` | `404` / `400` / `405` |
| end-10 | `DELETE` | `/v2/<name>/blobs/<digest>` | `202` | `404` / `405` |
| end-11 | `POST` | `/v2/<name>/blobs/uploads/?mount=<digest>&from=<other_name>` | `201` | `404` |
| end-12a | `GET` | `/v2/<name>/referrers/<digest>` | `200` | `404` / `400` |
| end-12b | `GET` | `/v2/<name>/referrers/<digest>?artifactType=<artifactType>` | `200` | `404` / `400` |
| end-13 | `GET` | `/v2/<name>/blobs/uploads/<reference>` | `204` | `404` |

# Notary v2 signature discovery (1/2)

- Notary v2 provides for multiple signatures of an OCI artifact (including container images)




`application/vnd.oci.artifact.manifest.v1+json`

```
ivan.wallis@07WKSMAC150894 oci-layout % oras discover ivanvenafi.azurecr.io/net-monitor:v1 -o json
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.index.v1+json",
  "manifests": [
    {
      "mediaType": "application/vnd.oci.artifact.manifest.v1+json",
      "digest": "sha256:d16bb71c4ffc4c7c600756ccfc74f51936d5972bfffeedd7838465f5ea8a71f0",
      "size": 763,
      "annotations": {
        "io.cncf.notary.x509chain.thumbprint#S256": "[\"b92f6247135415dd98bd4cdc85839195f4fd6e64cbff3d16be4f9ef8bc50b58a\",
c4edcbc506994ed56550ea2f556ef75a7763cc9d9935512853b0d5a0b72660d\"]",
        "org.opencontainers.artifact.created": "2023-02-17T16:12:13Z"
      },
      "artifactType": "application/vnd.cncf.notary.signature"
    }
  ]
}
```

`application/vnd.cncf.notary.signature`

# Notary v2 signature discovery (2/2)

- artifactType -> Required property references the Notary version of the signature
- blobs -> Required collection of only one OCI descriptor referencing signature envelope
  - mediaType -> type of signature envelope blob

    `application/jose+json`

    `application/cose`
- subject -> Required artifact descriptor referencing the signed manifest, including, but not limited to image manifest, image index, OCI artifact manifest
- annotations -> Required property contains metadata for the artifact manifest.
  - Uses `io.cncf.notary` namespace reserved for use in Notary
  - `io.cncf.notary.x509chain.thumbprint#S256` A required annotation containing the list of SHA256 thumbprints of signing certificate and chain (including root).

Backward Compatibility: Notary v2 supports using OCI Image Manifest to store signatures in registries with partial support for OCI Image Specification 1.1

# Artifact/SBOM signature discovery

<Experimental>

- SBOMs can be *attached* to containers via the in-progress OCI references API
- SBOM objects are represented as an OCI Image Manifest V1
- This means they can be signed like container images!
- Cosign uses mediaType `application/vnd.dev.cosign.artifact.sig.v1+json` to represent the signature.  Annotations remain consistent with container image signature payload.

</Experimental>

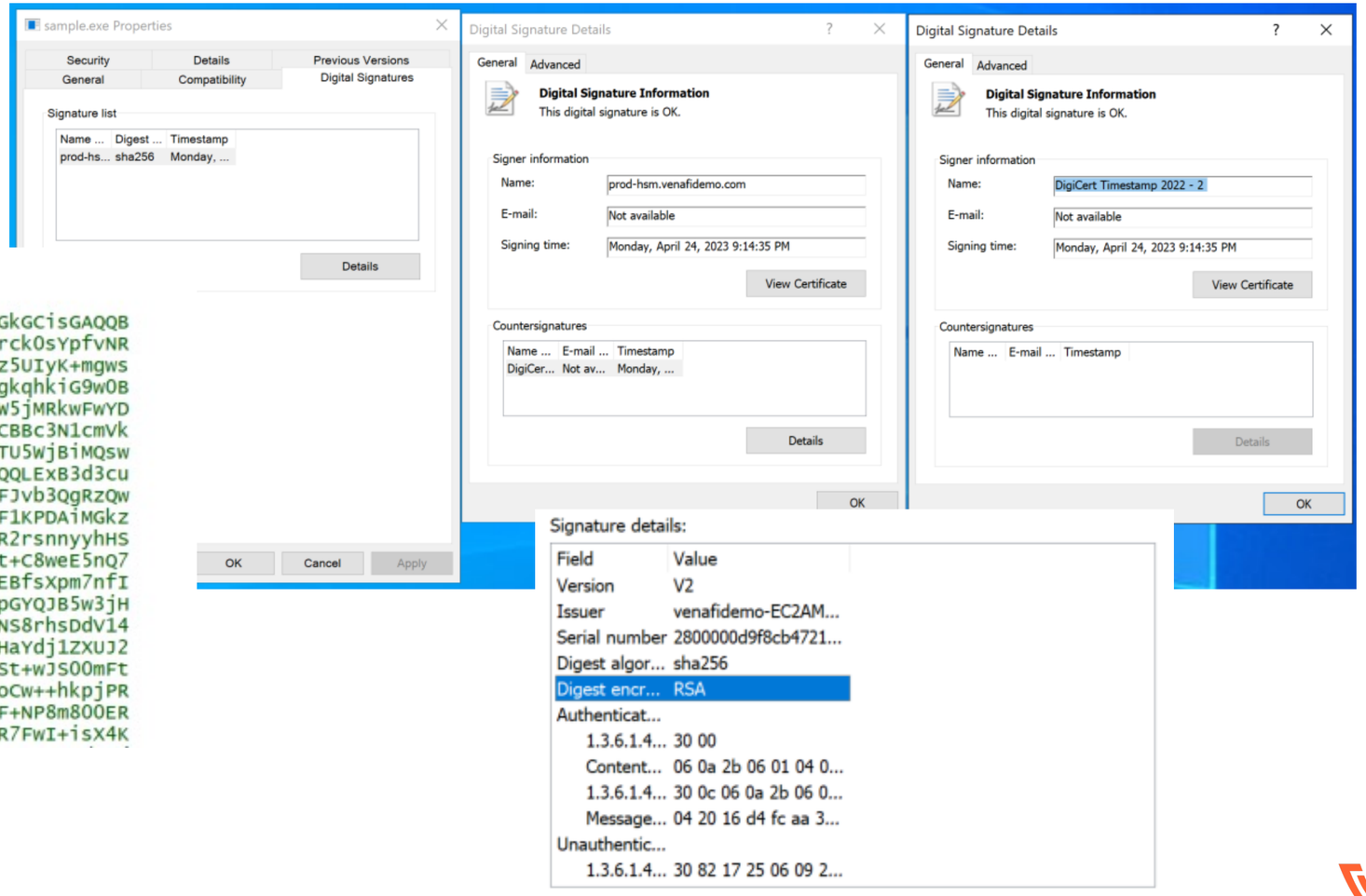
# OCI compatibility

| Sigstore cosign | Notaryv2 (notation) |
| --- | --- |
| • AWS Elastic Container Registry<br>• GCP's Artifact Registry and Container Registry<br>• Docker Hub<br>• Azure Container Registry<br>• JFrog Artifactory Container Registry<br>• The CNCF distribution/distribution Registry<br>• GitLab Container Registry<br>• GitHub Container Registry<br>• The CNCF Harbor Registry<br>• Digital Ocean Container Registry<br>• Sonatype Nexus Container Registry<br>• Alibaba Cloud Container Registry<br>• Red Hat Quay Container Registry 3.6+ / Red Hat quay.io<br>• Elastic Container Registry<br>• IBM Cloud Container Registry<br>• Cloudsmith Container Registry | • OCI Artifacts<br>  • Zot<br>  • Azure Container Registry<br>  • Amazon Container Registry<br>  • GAR<br>  • ORAS Project registry<br>  • GitHub<br>  • Bundle Bar<br>  • Docker Hub<br>• Image Manifest<br>  • Any OCI compliant registry |

# File signature discovery

- Windows executables/DLL/PS1 scripts rely on PE/COFF format

```
120
121  }
122  # SIG # Begin signature block
123  # MIIevQYJKoZIhvcNAQcCoIIerjCCHqoCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
124  # gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
125  # AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQU+T5+AU4YEP/Ovz5UIyK+mgws
126  # rZigghjyMIIFjTCCBHWgAwIBAgIQDpsYjvnQLefv21DiCEAYWjANBgkqhkiG9w0B
127  # AQwFADBlMQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYD
128  # VQQLExB3d3cuZGlnaWNlcnQuY29tMSQwIgYDVQQDExtEaWdpQ2VydCBBc3N1cmVk
129  # IElEIFJvb3QgQ0EwHhcNMjIwODAxMDAwMDAwWhcNMzExMTA5MjM1OTU5WjBiMQsw
130  # CQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3d3cu
131  # ZGlnaWNlcnQuY29tMSEwHwYDVQQDExhEaWdpQ2VydCBUcnVzdGVkIFJvb3QgRzQw
132  # ggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQC/5pBzaN675F1KPDAiMGkz
133  # 7MKnJS7JIT3yithZwuEppz1Yq3aaza57G4QNxDAf8xukOBbrVsaXbR2rsnnyyhHS
134  # 5F/WBTxSD1Ifxp4VpX6+n6lXFllVcq9ok3DCsrp1mWpzMpTREEQQLt+C8weE5nQ7
135  # bXHiLQwb7iDVySAdYyktzuxeTsiT+CFhmzTrBcZe7FsavOvJz82sNEBfsXpm7nfI
136  # SKhmV1efVFiODCu3T6cw2Vbuyntd463JT17lNecxy9qTXtyOj4DatpGYQJB5w3jH
137  # trHEtWoYOAMQjdjUN6QuBX2I9YI+EJFwq1WCQTLX2wRzKm6RAXwhTNS8rhsDdV14
138  # Ztk6MUSaMOC/CNdaSaTC5qmgZ92kJ7yhTzm1EVgX9yRcRo9k98FpiHaYdj1ZXUJ2
139  # h4mXaXpI8OCiEhtmmnTK3kse5w5jrubU75KSOp493ADkRSWJtppEGSt+wJSO0mFt
140  # 6zPZxd9LBADMfRyVw4/3IbKyEbe7f/LVjHAsQWCqsWMYRJUadmJ+9oCw++hkpjPR
141  # iQfhvbfmQ6QYuKZ3AeEPlAwhHbJUKSWJbOUOUlFHdL4mrLZBdd56rF+NP8m8OOER
142  # ElvlEFDrMcXKchYiCd98THU/Y+whX8QgUWtvsauGi0/C1kVfnSD8oR7FwI+isX4K
```



# Demo

# Considerations

- Performance
  - API rate limiting
- Compatibility
  - Registry OCI spec implementations
- Artifact Types (Signed)
  - SBOMs
  - Attestations
  - Vulnerability reports
  - Helm charts
- FileSystem Types
  - XML
  - PDF

# Future Enhancements

- Improved readability, visualization
- Ability to export results to Grafana
- Ability to export results to policy controllers
- Ability to map signer identities to Issuing Authority, CMDB, etc.

# Thank You

- Download and test Sigscan -> https://github.com/venafi/sigscan
- Contribute via Issues or PR

**Ivan Wallis**
**Twitter: @CryptoDevGeek**
**GitHub: zosocanuck**