How to use Analog files

Table of Contents

1. Introduction

Analog allows a server to detect missing patches and vulnerabilities for client machines centrally. It also allows sysadmins to download, deploy and install the patches to clients to fix these vulnerabilities, thus improving risk posture of the system.

This document helps users understand the components of Analog (database files for server and client) and how to use them together with OESIS to satisfy a common use case of detecting vulnerabilities on clients and deploying patches to fix them, all triggered from a central server.

2. Databases

An Analog package contains many files to be deployed to either server or client. The typical structure of the package is as follows:

```
analog.zip
|_header.json                          # publish time and checksum of server and client files
|_server                               # files to be used on server
  |_ap_support_chart.json              # auto-patching support chart for Windows
  |_ap_support_chart_mac.json          # auto-patching support chart for Mac
  |_cves.json                          # detailed information of all supported CVEs
  |_kb_info.json                       # json representation of kb_base, kb_cves, and kb_tree in wiv-lite.dat
  |_os_info.json                       # detailed information of all OS
  |_patch_aggregation.json             # list of all patches in patch.dat and patch_mac.dat
  |_patch_associations.json            # list of patch associations
  |_patch_status.json                  # detailed information of all supported patches
  |_patch_system_aggregation.json      # list of all patches in wuo.dat
  |_products.json                      # list of all OESIS-supported products
  |_vuln_associations.json             # list of 3rd-party vulnerability associations
  |_vuln_system_associations.json      # list of OS vulnerability associations
|_client                    # files to be used on client
  |_ap_checksum.dat         # contains expected checksum for 3rd-party software on Windows
  |_ap_checksum_mac.dat     # contains expected checksum for 3rd-party software on Mac
  |_compliance              # contains compliance packages
  |_liv.dat                 # contains vulnerability detection logic for packages on Linux
  |_mav.dat                 # contains vulnerability detection logic for MacOS
  |_patch.dat               # contains patching logic for 3rd-party software on Windows
  |_patch_mac.dat           # contains patching logic for 3rd-party software on Mac
  |_vmod.dat                # contains vulnerability detection logic for 3rd-party software
  |_vmod-vuln-oft.dat       # same as vmod.dat but is smaller by stripping of vulnerability details
  |_v2mod.dat               # contains vulnerability detection logic for 3rd-party software
  |_v2mod-vuln-oft.dat      # same as v2mod.dat but is smaller by stripping of vulnerability details
  |_wuo.dat                 # contains download and install details for system software on Windows
  |_wiv-lite.dat            # contains vulnerability detection logic for system software on Windows
```

The files for server usage are in plain JSON format so the server can read and load them into its own database easily. The files for client are in binary format to be read by Vulnerability module in OESIS. Not all the files are required in every use case so users can ignore the files they do not need.

2.1. Synchronization

The workflow to fetch the package, read the server files and deploy client files is shown in the below diagram.

OPSWAT releases new package every day and recommends users to keep their copy relatively up to date so that new vulnerabilities can be detected soon after they appear and old, invalid download links for patches are updated often.

2.2. Database structure

The following sections show the schema of each server file and explain how to use it.

2.2.1. cves.json

```
{
  "cve": string,                    // CVE identifier
  "cwe": string,                    // CWE identifier of the CVE
  "description": string,            // description of the CVE
  "published_epoch": number,        // published timestamp of the CVE
  "last_modified_epoch": number,    // last modified timestamp of the CVE
  "references": [                   // reference links to find out more about the CVE
    {
      "url": string,                // reference URL
      "name": string,
      "reference_type": string,
      "source": string
    },
    ...
  ],
  "cvss_2_0":                       // (optional)
  {                                 // https://www.first.org/cvss/v2/guide
    "score": string,
    "access_vector": string,
    "access_complexity": string,
    "authentication": string,
    "confidentiality_impact": string,
    "integrity_impact": string,
    "availability_impact": string,
    "source": string,
    "impact_score": string,
    "exploitability_score": string
  },
  "severity_index": number,         // OPSWAT severity score of the CVE, range 0 - 100
  "severity": string,               // severity rating of the CVE based on OPSWAT score.
                                    // possible values are "LOW", "MODERATE", "IMPORTANT", "CRITICAL"
  "cvss_3_0":                       // (optional)
  {                                 // https://www.first.org/cvss/v3.1/specification-document
    "impact_score": string,
    "vector_string": string,
    "attack_vector": string,
    "attack_complexity": string,
    "privileges_required": string,
    "user_interaction": string,
    "scope": string,
    "confidentiality_impact": string,
    "integrity_impact": string,
    "availability_impact": string,
    "base_score": string,
    "base_severity": string,
    "exploitability_score": string,
    "revision": string              // "0" for CVSS 3.0, "1" for CVSS 3.1
  }
}
```

This file contains detailed information about all CVEs referenced in other files.

2.2.2. kb_info.json

```
{
  "id_os_map": {                    // Holds mappings from OS IDs to their name which will be used as keys
    "\\d+": "\\w+",                 // Mapping from OS ID (\d+) to its key (\w+)
    ...                             // Additional mappings can be added here
  },
  "\\w+": {                         // Key after being mapped in id_os_map
                                    // Holds info about an OS kb_base, kb_cves, and kb_tree
                                    // The key for an OS should follow this regex pattern
    "kb_base": {
      "\\d+\\.\\d+": {              // Regex pattern for a build version (\d+\.\d+)
        "build_version": string,    // Build version itself
        "kb_articles": [
          {
            "kb_id": number,
            "availability_date": number
          },
          ...
        ]
      },
      ...                           // Additional build versions can be added here
    },
    "kb_cves": {
      "\\d+": {                     // Regex pattern for a KB ID (\d+)
        "kb_id": number,            // KB ID itself
        "cves": array<number>       // Array of CVE IDs that this KB addresses
      },
```

```
      ...                           // Additional kbs can be added here
    },
    "kb_tree": {
      "\\d+": {                     // Regex pattern for a KB ID (\d+)
        "kb_id": number,            // KB ID itself
        "supersede_kbs": array<number>, // Array of KB IDs that this KB supersedes
        "cumulative": boolean,      // (optional) Indicates if this KB is cumulative
      },
      ...                           // Additional kbs can be added here
    }
  },
  ...                               // Additional OS entries can be added here
}
```

Example:

```
{
  "id_os_map": {
    "82": "win10",
    "81": "win11",
    "71": "winserver2016",
    "74": "winserver2019",
    "80": "winserver2022"
  },
  "win11": {
    "kb_base": {
      "26100.4351": {
        "build_version": "26100.4351",
        "kb_articles": [{
          "kb_id": 5063060,
          "availability_date": 1749600000
        }]
      },
      "26100.2033": {
        "build_version": "26100.2033",
        "kb_articles": [{
          "kb_id": 5044284,
          "availability_date": 1728345600
        }]
      }
    },
    "kb_cves": {
      "5008880": { "kb_id": 5008880, "cves": [202221911] },
      "5044284": {
        "kb_id": 5044284,
        "cves": [
          202420659, 202430092, 202437976, 202437982, 202437983, 202438149,
          202443500, 202443501, 202443506, 202443508, 202443509, 202443511,
          202443513, 202443514, 202443515, 202443516, 202443517, 202443518,
          202443519, 202443520, 202443523, 202443524, 202443525, 202443526,
          202443527, 202443528, 202443529, 202443532, 202443533, 202443534,
          202443535, 202443536, 202443537, 202443538, 202443540, 202443542,
          202443543, 202443546, 202443547, 202443550, 202443551, 202443552,
          202443553, 202443554, 202443555, 202443556, 202443557, 202443558,
          202443559, 202443560, 202443561, 202443562, 202443563, 202443565,
          202443570, 202443571, 202443572, 202443573, 202443574, 202443581,
          202443582, 202443583, 202443584, 202443585, 202443599, 202443615,
          20246197
        ]
      }
    },
    "kb_tree": {
      "5008880": {
        "kb_id": 5008880,
        "supersede_kbs": [5005537],
        "cumulative": true
      },
      "5044284": {
        "kb_id": 5044284,
        "supersede_kbs": [5039239, 5041571, 5040435, 5043080],
        "cumulative": true
      }
    }
  }
}
```

2.2.3. os_info.json

```
{
  "win": [
    {
      "os_family": string,          // Name of the Windows family
      "list": [                     // List of Windows editions
        {
          "os_name": string,        // Name of the Windows edition
          "os_id": number,          // ID of the Windows edition
          "versions": [             // List of Windows edition versions
            {
```

```
            "release_label": string  // Release version of the Windows edition
          },
          ...
        ]
      },
      ...
    ]
  },
  ...
],
"mac": [
  {
    "os_family": string,          // Name of the macOS family
    "list": [                     // List of macOS editions
      {
        "os_name": string,        // Name of the macOS edition
        "os_id": number           // ID of the macOS edition
      },
      ...
    ]
  },
  ...
],
"linux": [
  {
    "os_family": string,          // Name of the Linux family
    "list": [                     // List of Linux flavors
      {
        "os_name": string,        // Name of the Linux flavor
        "os_id": number,          // ID of the Linux flavor
        "versions": [             // List of Linux flavor versions
          {
            "release_label": string // Release version of the Linux flavor
          },
          ...
        ]
      },
      ...
    ]
  },
  ...
]
}
```

Example:

When OS information is defined according to the JSON schem below:

```
{
  "win": [
    {
      "os_family": "Windows 11",
      "list": [
        {
          "os_name": "Windows 11 Enterprise",
          "os_id": 81,
          "versions": [
            {
              "release_label": "11-23h2-e"
            },
            ...
          ]
        },
        ...
      ]
    },
    ...
  ],
  "mac": [
    {
      "os_family": "macOS",
      "list": [
        {
          "os_name": "macOS Sonoma",
          "os_id": 1400
        },
        ...
      ]
    },
    ...
  ],
  "linux": [
    {
      "os_family": "Debian-based",
      "list": [
        {
          "os_name": "Ubuntu",
          "os_id": 4,
          "versions": [
```

```
            {
              "release_label": "24.04"
            },
            ...
          ]
        },
        ...
      ]
    },
    ...
  ]
}
```

The OS attributes such as *os_id* can be used to map with other OS attributes in Server files like:

- *os_deny* and *os_allow* in the patch_associations.json file, representing the *os_id* from the os_info.json file
- *os_type* in the vuln_associations.json file, representing the *os_id* from the os_info.json file
- etc.

This mapping enables the visualization of OS family, OS name, OS version, and additional details for enhanced display within the Server component of their product.

2.2.4. patch_aggregation.json

```
{
  "_id": number,                        // patch identifier
  "schema_version": string,             // schema version of the data
  "product_name": string,               // name of the product
  "vulnerabilities": array<string>,     // (optional) identifiers of the vulnerabilities (CVE)
  "release_note_link": string,          // link to release note webpage of the product
  "eula_link": string,                  // link to EULA webpage of the product
  "latest_version": string,             // the latest version of the product at the time of generation
  "language_default":string,            // (optional) default language to use in download links
  "architectures": array<string>,       // The architectures of the product that the patch applies to.
                                        // possible values are: empty array (cannot determine),
                                        //                      “x86_32”, “x86_64”, “x86_all”,
                                        //                      “arm”, “arm64”, “arm_all”.
  "fresh_installable": number,          // Whether the patch supports fresh install (1) or not (0).
  "release_date": string,               // release date of product, it can be different with website data at most 1 day
  "requires_reboot": string,            // requires a reboot after installation
  "download_links": [
    {
      "architecture": string,           // (optional) architecture of the product to use this link,
                                        // only "32-bit", "64-bit" or "arm64" for now
      "os_architecture": string,        // (optional) arch of the OS to use this link.
      "language": string,               // (optional) language of the product to use this link, e.g: "en-US", "fr"...
      "link": string,                   // download link
      "os_id ": array<number>,          // (optional) list os support of file installer
      "sha256": string                  // (optional) SHA256 hash of the download file
    },
    ...
  ],
  "requires_uninstall_first": string    // requires an uninstall first before installation.
}
```

This file contains supplementary data about each patch identifier (release note link, EULA link, download link…) for the server to query on its own instead of having to ask the clients.

Note: The field “architectures” is different from “download_links.architecture”. The latter is the architecture of the product corresponding to each download link in case the vendor has a different link for each architecture. Thus, if the vendor uses only one link for all architectures, the field is not needed and may not exist. The former is all product architectures that the patch applies and is aggregated from all values of the latter.

Example:

After having found the matching patch identifier from patch_association, the server uses it together with some client-side data like language name and architecture name to look up more information about the patch from patch_aggregation.

In below example, if the server knows that the patch identifier for the considering product is 27, its language is “en-US” and its architecture is “64-bit” then it knows that the product is Firefox, the latest available version is “101.0.1” and the appropriate download link is in the second object in *download_links* array.

```
{
  "_id": 27,
  "schema_version": "1.0-1.0",
  "product_name": "Mozilla Firefox",
  "vulnerabilities": [],
  "release_note_link": "https://www.mozilla.org/en-US/firefox/101.0.1/releasenotes/",
  "eula_link": "https://www.mozilla.org/en-US/about/legal/eula/",
  "latest_version": "101.0.1",
  "download_links": [
    {
      "link": "https://ftp.mozilla.org/pub/firefox/releases/101.0.1/win64/fr/...",
      "architecture": "64-bit",
      "language": "fr"
```

```
    },
    {
      "link": "https://ftp.mozilla.org/pub/firefox/releases/101.0.1/win64/en-US/...",
      "architecture": "64-bit",
      "language": "en-US"
    },
    {
      "link": "https://ftp.mozilla.org/pub/firefox/releases/101.0.1/win64/ro/...",
      "architecture": "64-bit",
      "language": "ro"
    },
    ...
  ],
  "language_default": "en-US"
}
```

In the case that there is no information about language of the product from the client then the server can use the language in *language_default* field, which contains the default language of the product (mostly “en-US”).

2.2.5. patch_associations.json

```
{
  "_id": number,                    // primary key of the record
  "schema_version": string,         // schema version of this association record
  "v4_pid": number,                 // product identifier of the linked product
  "v4_signatures": array<number>,   // signature identifiers of the linked product
  "version_pattern": string,        // (optional) version pattern (regular expression) of the linked product
  "ranges": [                       // (optional) version ranges of the linked product
    {                               // only "start" and "limit" (inclusive) allowed
      "start": string,              // null start means start at "0"
      "limit": string               // null limit means no limit
    },
    ...
  ],
  "comment": string,                // (optional) some notes about the patching for this product
  "is_latest": bool,                // whether the linked patch identifier links to the latest
                                    // patch for this product, in all product lines
  "os_deny": string,                // (optional) ranges of OESIS OS identifiers or OS version (from GetOSInfo) to block
                                    // '[' and ']' are inclusive start and end identifiers
                                    // '(' and ')' are exclusive start and end identifiers
  "os_allow": string,               // (optional) ranges of OESIS OS identifiers (from GetOSInfo) to allow
  "patch_id": number,               // identifier of the linked patch data
  "arch": string,                   // defined architecture of product support patch
  "channel_pattern": string,        // defined channel update pattern for product
  "title": string                   // the title of the patch
}
```

This file contains the mappings (associations) between OESIS product identifiers to a patch identifier, plus some optional filters so that a correct patch can be downloaded and applied to each specific product.

Except “_id”, “schema_version”, “patch_id” and “comment”, all other fields are filters to make the mapping from input product information to output patch identifier correct. If a filter (field) does not appear then it is not needed to do the correct mapping.
Note: Field “is_latest” is only used in case there is more than one association matching the input product information: the association with “is_latest” set to true is preferred.

Example:

In the below example, if an input product has product identifier 100327, signature identifier 100387, version 7.0.0 and the operating system it runs on has identifier 1010 then it satisfies the patch association and can use patch identifier 100027.

```
{
  "_id": 5100027,
  "schema_version": "1.0-1.0",
  "is_latest": false,
  "v4_pid": 100327,
  "v4_signatures": [
    100387
  ],
  "version_pattern": "^7\\.",
  "ranges": [
    {
      "start": "7.0.0",
      "limit": "7.0.9999"
    }
  ],
  "os_deny": "[-2, 1009]",          // os_id <= 1009 is denied (-2 means no upper/lower limit)
  "os_allow": "(1009, -2]",         // os_id > 1009 is allowed (-2 means no upper/lower limit)
  "patch_id": 100027,
  "title": "Winzip 7"
}
```

If the input product fails any of the association filters, e.g: it has version 7.1 which is bigger than limit version here, then it does not match the association.

Then, the server can look up *patch_id* in patch_aggregation.json to download the appropriate patch or pass the patch identifier to GetLatestInstaller on the client to ask the client to download the patch itself.

2.2.6. patch_system_aggregation.json

```
{
  "_id": string,                    // patch identifier
  "schema_version": string,         // schema version of the data
  "product_name": string,           // name of the related product
  "vulnerabilities": array<string>, // identifiers of the vulnerabilities, only CVEs for now
  "kb_id": number,                  // (optional) Microsoft KB this patch belongs to.
  "release_note_link": string,      // release note of the patch
  "architectures": array<string>,   // The architecture of the product that the patch applies to.
                                    // possible values are: empty array (cannot determine),
                                    //                      “x86_32”, “x86_64”, “x86_all”,
                                    //                      “arm”, “arm64”, “arm_all”.
  "download_link": {
    "architecture": string,         // (optional) arch of the product to use this link,
                                    // only "32-bit", "64-bit" or "arm64" for now
    "os_architecture": string,      // (optional) arch of the OS to use this link.
    "language": string,             // (optional) language of the product to use this link, e.g: "en-US", "fr"...
    "link": string,                 // download link
    "sha1": string                  // SHA1 hash of the download file
  },
  "release_date": string,           // release date of product, it can be different with website data at most 1 day
  "requires_reboot": string,        // requires a reboot after installation
  "description": string,            // description of the patch
  "title": string,                  // title of the patch
  "category": string,               // category of the patch
  "severity": string,               // severity level of the patch
  "vendor": string,                 // vendor of the patch
  "os_info": array<string>,         // list of operating systems that can apply the patch
  "optional": bool                  // whether the patch is considered optional
}
```

This file is only used when clients use Windows Update Offline (WUO) feature to detect missing system patches on Windows platform.

Similar to patch_aggregation.json, but for system patches. This file contains supplementary data about each system patch (Windows-only) so that the server can query and know more about the patch.

A minor difference with patch_aggregation.json is that the detect of patch identifier for system patches must happen on the client. GetLatestInstaller on client returns *analog_id* field that serves as a patch identifier for the server to query in patch_system_aggregation.

Example:

In the example below, after the client calls GetLatestInstaller and get the list of missing system patches, it sends the list of *analog_id* to the server. The server can look up an identifier like "f30ba29a-decf-4b18-a24e-8a63260618a1" in patch_aggregation and know the vulnerabilities being fixed and the download link.

```
{
  "_id": "f30ba29a-decf-4b18-a24e-8a63260618a1",
  "schema_version": "1.0-1.0",
  "product_name": "Windows Security Update",
  "release_note_link": "https://www.catalog.update.microsoft.com/ScopedViewInline.aspx...",
  "kb_id": 4519338,
  "download_link": {
    "link": "http://download.windowsupdate.com/c/msdownload/update/software/secu/2019/...",
    "architecture": "64-bit",
    "sha1": "fffbd25b18d1239db1a2..."
  },
  "vulnerabilities": [
    "CVE-2019-1454",
    "CVE-2019-1371",
    ...
  ],
  "optional": false
}
```

2.2.7. products.json

```
{
  "product": {
    "id": number,                   // OESIS identifier of the product
    "name": string                  // product name
  },
  "vendor": {
    "id": number,                   // OESIS identifier of the vendor
    "name": string                  // vendor name
  },
  "signatures": [                   // a product can have many signature identifiers
    {
      "id": number,                 // OESIS signature identifier
      "name": string,               // signature name
      "support_3rd_party_patch": bool, // whether the signature can be patched as 3rd-party patching or not
      "fresh_installable": number,  // fresh_installable values:
```

```
                                    // 0 means the product cannot be freshly installed.
                                    // 1 means the product can be freshly installed.
      "validation_supported": number,  // (Optional) validation support status
                                    // If it has value 1 then validation is supported for the product
                                    // If it does not exist then validation is not supported for the product
      "3rd_party_patch_properties": {  // 3rd-party patch properties
        "installation_mode": number    // 0 means the product not support 3rd-party patch
                                    // 1 means the product supported per-machine mode.
                                    // 2 means the product supported both per-machine and per-user mode.
                                    // 3 means the product supported per-user mode.
      }
    },
    ...
  ],
  "marketing_names": array<string>,    // marketing names of the product
  "support_system_patch": bool,        // whether it can be patched as system patching or not
  "support_3rd_party_patch": bool      // whether the product can be patched as 3rd-party patching or not
}
```

Example:

```
{
  "product": {
    "id": 947,
    "name": "McAfee LiveSafe - Internet Security"
  },
  "vendor": {
    "id": 379,
    "name": "McAfee, Inc."
  },
  "signatures": [
    {
      "id": 959,
      "name": "McAfee LiveSafe - Internet Security",
      "support_3rd_party_patch": false
    }
  ],
  "marketing_names": [
    "McAfee VirusScan",
    "McAfee Multi Access - Internet Security"
  ],
  "support_system_patch": false,
  "support_3rd_party_patch": false
}
```

2.2.8. vuln_associations.json

```
{
  "_id": number,                    // identifier of the association
  "schema_version": string,         // schema version of the data
  "v4_vid": number,                 // vendor identifier of the linked product
  "os_type": number,                // OS type identifier the linked product runs on (Linux, Windows or Mac)
  "v4_pids": array<number>,         // identifiers of the linked product(s)
  "ranges": [                       // version ranges of the linked product
    {
      "start": string,              // null start means start at "0"
      "limit": string               // null limit means no limit
    },                              // only "start" and "limit" (inclusive) allowed
    ...
  ],
  "cve": string,                    // CVE identifier
  "cpe": string,                    // (optional) CPE of the linked CVE
  "channel_pattern": string,        // (exists in vuln_associations_1.1 object) the update channel of the product
  "timestamp": number,              // last modified time
  "v4_signatures": array<number>    // (optional) signature identifiers of the linked product
}
```

This file contains the mapping between one or many 3rd-party software to a CVE. If the data about the software satisfies all filters inside an association then the software is affected by the CVE the association links to.

Example:

In the example below, if the querying software has vendor identifier 41, product identifier 2985 and version “50.0” then it is affected by CVE-2019-17026.

```
{
  "_id": 2141106,
  "v4_vid": 41,
  "v4_pids": [2985],
  "ranges": [
    {
      "start": "0.0",
      "limit": "68.4.0.999998"
    }
  ], //  0.0 <= version (from GetVersion) <= 68.4.0.999998
  "cve": "CVE-2019-17026",
  "timestamp": 1579748542,
```

```
  "schema_version": "1.0-1.0",
  "os_type": 1
}
```

Note: For CVEs with *channel_pattern* information, they are stored in the *vuln_associations_1.1* object, while others are by default stored in the *vuln_associations* object.

2.2.9. vuln_system_associations.json

2.2.9.1. windows_vuln_system_associations

```
{
  "cve": string,
  "kb_articles": [
    {
      "article_name": string,
      "os_id": array<number>,
    },
    ...
  ],
  "v4_vid": number,
  "v4_pids": array<number>,
  "v4_signatures": array<number>,
  "cpe": string,
  "schema_version": string,
  "os_type": number
}
```

Example:

```
{
  "cve": "CVE-2021-36970",
  "kb_articles": [
    {
      "article_name": "4487000",
      "os_id": [51, 52, 53, 54, 55]
    }
  ],
  "v4_vid": 90,
  "v4_pids": [1090],
  "v4_signatures": [1103],
  "cpe": "cpe:/o:microsoft:windows",
  "schema_version": "1.0-1.0",
  "os_type": 1
}
```

2.2.9.2. linux_vuln_system_associations

```
{
  "cve": string,
  "affected_os": [
    {
      "os_name": string,
      "v4_vid": number,
      "v4_pids": array<number>,
      "v4_signatures": array<number>,
      "affected_packages": [
        {
          "package_name": string,
          "ranges": [
            {
              "start": string,
              "limit_except": string
            },
            ...
          ]
        },
        ...
      ]
    },
    ...
  ],
  "schema_version": string,
  "os_type": number
}
```

Example:

```
{
  "cve": "CVE-2005-0488",
  "affected_os": [
    {
      "os_name": "Debian 12",
      "v4_vid": 200074,
      "v4_pids": [200040],
      "v4_signatures": [200041],
      "affected_packages": [{
        "package_name": "krb5",
```

```
        "ranges": [{
          "start": "0",
          "limit_except": "1.8.3+dfsg-4"
        }]
      }]
    }
  ],
  "schema_version": "1.0-1.0",
  "os_type": 2
}
```

2.2.9.3. *macos_vuln_system_associations*

```
{
  "cve": string,
  "v4_vid": number,
  "v4_pids": array<number>,
  "v4_signatures": array<number>,
  "affected_os": [
    {
      "os_name": string,
      "os_id": number,
      "ranges": [
        {
          "start": string,
          "limit_except": string
        },
        ...
      ]
    },
    ...
  ],
  "cpe": string,
  "schema_version": string,
  "os_type": number
}
```

Example:

```
{
  "cve": "CVE-2024-44246",
  "v4_vid": 100011,
  "v4_pids": [100076],
  "v4_signatures": [100192],
  "affected_os": [{
    "os_name": "Sequoia",
    "os_id": "1500",
    "ranges": [{
      "start": "15.0",
      "limit_except": "15.2"
    }]
  }],
  "cpe": "cpe:/o:apple:macos",
  "schema_version": "1.0-1.0",
  "os_type": 4
}
```

3. Required client data

Processing on server requires data on client about operating system and 3rd-party software. This section describes the (minimum) client data needed to fulfill the processing of each server file and how to get them.

3.1. Scan 3rd-party software patches

The files patch_associations.json and patch_aggregation.json require information about both the software and the operating system. Information about software can be found in OESIS functions GetProductInfo and GetVersion. Information about the operating system can be found in GetOSInfo.

This data, collected by client and send to server, can be represented as follow:

```
{
  "os": {                          // from GetOSInfo
    "version": string,
    "architecture": string,
    "service_pack": string,
    "os_type": number,
    "os_id": number
  },
  "product": {
    "signature_id": number,        // from GetProductInfo
    "product_id": number,
    "vendor_id": number,
    "version": string,             // from GetVersion
    "architecture": string,        // architecture name from GetVersion
```

```
    "language": string              // language name from GetVersion
  }
}
```

When the server receives this data, it should query its database to find the patch identifier (and optionally the download link if it wants to cache the patch for other clients), then it pass this identifier back to the client as parameter for GetLatestInstaller to download the patch and InstallFromFiles to apply the downloaded patch to the client machine. Then, the client queries this information again and send it to the server to report new software status.

3.2. Scan system patches

If the client uses Windows Update Offline feature (WUO) to scan missing patches for system software then GetLatestInstaller will return *analog_id* field for each patch. The client then collect these identifiers and send them to the server for it to query patch_system_aggregation to find related information and possibly cache the patch to reuse later.

The data sent by client can be in the following format:

```
{
  "signature_id": number,           // OESIS signature identifier of the system software
  "patches": array<string>          // list analog_ids of missing system patches
}
```

Upon receiving this data, the server can download the patches, send them to the client and call InstallFromFiles for each patch to apply it into the client's system.

3.3. Scan 3rd-party software vulnerability

Using the same client data as in section 3.1, server can query vuln_associations.json to find the CVEs affecting the queried software, then look up the CVE identifiers in cves.json to get more information about them to display to the end-user.

No data from the server need to be sent back to the clients for this use case.

3.4. Scan system vulnerability

If the client uses WUO to scan missing system patches then it can use Windows Vulnerability detection feature (WIV) to also query CVEs affecting the operating system. The client calls GetProductVulnerability with the signature of Windows Update Agent to get the CVEs list and sends the list to the server, which consults cves.json to get full information about them to display to the administrator.

3.5. Compliance package

The "compliance" folder contains data binaries files (including libwaresource.dll, libwaresource.dylib and libwaresource.so for Windows, Mac and Linux) as well as related documentations (support charts, moby, …) for Compliance module of MetaDefender Endpoint Security SDK (MDES SDK). It is essential for initializing MDES SDK, regardless of whatever module are being used. Previously, Compliance data package was shipped along with Engine package in every release through OPSWAT Portal. Now it can be obtained in Analog through VCR gateway for faster data access for customers who require daily updates, ensuring timely and efficient delivery of the latest information.

Folder structure:

```
compliance/
|__windows/
|  |__bin/
|  |  |__libwaresource.dll          # SDK Compliance Data Binary for Windows
|  |__docs/                         # Support charts
|__linux/
|  |__bin/
|  |  |__arm64/
|  |  |  |__libwaresource.so        # SDK Compliance Data Binary for Linux arm64
|  |  |__x64/
|  |  |  |__libwaresource.so        # SDK Compliance Data Binary for Linux x64
|  |  |__x86/
|  |  |  |__libwaresource.so        # SDK Compliance Data Binary for Linux x86
|  |__docs/                         # Support charts
```

4. Schema version

Some server files include a schema version in format *<start_version>-<limit_version>*. Start and limit version have format *number.number*. Schema version is the version of the data structure it belongs to. It is actually a range, with the start and the limit version mark the range of versions that can be processed the same way by the cloud server.

For example: If schema version of a data structure is "1.0-1.5" then cloud servers that can process version 1.0 can use the same processing for this data. The same for cloud server that can process version 1.1, 1.2, 1.3, … until 1.5. Cloud server that can only process versions 0.9 or 1.6 cannot use this data.

Structure of patch_associations.json with more than one schema version co-exist::

```
{
  "oesis": [{
```

```
    {
      "header": {...}
    },
    {
      "patch_associations": {...}, // default is 1.0
      "patch_associations_1.1": {...},
      ...
    }
  }]
}
```

5. Version comparison

A common processing step required when using Analog database is comparing versions. Different implementations of this step can lead to incorrect results across programming languages and platforms. This is due to the wide variety of possible values in version string and their assumed meanings.

Different software are versioned differently, even if they come from the same organization! Although there are suggestions to use Semantic Versioning (at https://semver.org/), not all software follow it and we cannot enforce them.

To mitigate the issue somewhat, we provide a reference implementation in Ruby language to compare versions. It is based on just two main assumptions selected from observation of many software versions on Windows and Mac to make it generic enough for many cases on these platforms:

1. A version is formatted as a mix of number (must-have) and non-number (optional) components, with left-to-right comparison order.
2. Two versions having different non-number components at the same position are not comparable. *Position* is the index of the component in an array formed by splitting number and non-number components in the version.

The code is:

```ruby
class Version
  include Comparable

  attr_reader :version_string, :number_segments, :constant_segments

  def initialize(vstr)
    @version_string = vstr.gsub('_', ' ').gsub(/^v(\d)/, '\1')
    @number_segments = []
    @constant_segments = []
    return if vstr.empty? # don't need to process anymore if vstr is empty (invalid)

    temp_segment = ''
    old_status = get_char_status @version_string.chr
    @version_string.chars.each do |char|
      new_status = get_char_status char
      if new_status == old_status
        temp_segment += char
      else
        # found something different, backup the "temp_segment" that is built thus far to appropriate storage
        if old_status == 1 # if we've encountered number til now then save that number
          @number_segments << temp_segment.to_i
        else # else, we will treat it as string constant
          @constant_segments << temp_segment
        end
        temp_segment = char # assign 'char' as first character of new segment
        old_status = new_status # new segment has new character status
      end
    end
    if old_status == 1 # if we've encountered number til now then save that number
      @number_segments << temp_segment.to_i
    else # else, we will treat it as string constant
      @constant_segments << temp_segment
    end

    # remove zeros at the end of version
    count_zero_number_tail = 0
    while @number_segments.size >= 2 && @number_segments.last == 0
      @number_segments.pop
      count_zero_number_tail += 1
    end
    count_zero_number_tail += 1 if old_status != 1
    while !@constant_segments.empty? && count_zero_number_tail > 0
      @constant_segments.pop
      count_zero_number_tail -= 1
    end
  end

  # compare between this version object and another version object or string
  # return 1 if this > other, 0 if this == other, -1 if this < other and nil if they are not comparable
  def <=>(other)
    return nil unless Version.is_valid_version(other) && Version.is_valid_version(@version_string)

    other_version = other.is_a?(Version) ? other : Version.new(other)

    minimum_num_constants = [@constant_segments.size, other_version.constant_segments.size].min
```

```ruby
minimum_num_numbers = [@number_segments.size, other_version.number_segments.size].min

if get_char_status(@version_string[0]) == 1 # first char is digit, so compare digit first, then constant strings
  # compare pairs of number segments having the same index
  j = 0
  for i in 0...minimum_num_numbers
    return 1 if @number_segments[i] > other_version.number_segments[i]
    return -1 if @number_segments[i] < other_version.number_segments[i]

    next unless j < minimum_num_constants
    return nil if @constant_segments[j] != other_version.constant_segments[j]

    j += 1
  end
else # first char is character, so compare constants first, then numbers
  # compare pairs of constant segments having the same index
  # the constant parts must be equal for the 2 versions to be comparable
  j = 0
  for i in 0...minimum_num_numbers
    if j < minimum_num_constants
      return nil if @constant_segments[j] != other_version.constant_segments[j]

      j += 1
    end
    return 1 if @number_segments[i] > other_version.number_segments[i]
    return -1 if @number_segments[i] < other_version.number_segments[i]
  end
end
```

```ruby
    # if it reaches here then the number segments are equal until now,
    # check the longer segment and see if there is only 0 in the left parts
    if @number_segments.size != other_version.number_segments.size
      longer_number_segments = (@number_segments.size > other_version.number_segments.size ?
                                @number_segments : other_version.number_segments)
      only_zero = true
      for i in minimum_num_numbers...longer_number_segments.size
        if longer_number_segments[i] != 0
          only_zero = false
          break
        end
      end
      return 0 if only_zero
      return 1 if @number_segments.size > other_version.number_segments.size

      return -1
    end
    0
  end

  # return type of a character, 1 means number, 0 means alpha character, -1 means other (e.g: '(', '.')
  def get_char_status(char)
    return 1 if /\p{Digit}/ =~ char
    return 0 if /\p{Alpha}/ =~ char
    -1
  end

  def self.is_valid_version(version)
    return true if version && version.to_s =~ /\d/
    false
  end
end
```

6. Changelog

31 July 2025 changes:

- Add new value for require reboot in patch_aggregation.json

22 July 2025 changes:

- Add field “sha256” in patch_aggregation.json.
- Add fields “sha1” and “optional” in patch_system_aggregation.json.

26 May 2025 changes:

- Add kb_info.json

04 Feb 2025 changes:

- Add vuln_system_associations.json.

31 Jan 2025 changes:

- Add optional field “signature_name_map” in ap_support_chart.json and ap_support_chart_mac.json.

13 Jan 2025 changes:

- Add compliance packages to Analog.

17 Dec 2024 changes:

- Add field "cisa" in cves.json.

07 Oct 2024 changes:

- Add os_info.json.

04 Oct 2024 changes:

- Add installation_mode to products.json.

30 Sep 2024 changes:

- Add mav.dat to Analog.

13 Jun 2024 changes:

- Add patch_status.json to Analog.

10 Jun 2024 changes:

- Added new attributes in vuln_associations.json.

07 Jun 2024 changes:

- Added new attributes in patch_system_aggregation.json.

08 May 2024 changes:

- Added ap_support_chart.json and ap_support_chart_mac.json in analog.

26 Feb 2024 changes:

- Add liv.dat to Analog.

28 Dec 2023 changes:

- Requires at least "cvss_2_0" or "cvss_3_0" in each cve in cves.json file.

06 Dec 2023 changes:

- Add v2mod.dat and v2mod-vuln-oft.dat to Analog.

28 Nov 2023 changes:

- Add field "support_3rd_party_patch" for each signature in products.json.
- Add field "title" in patch_associations.json

08 Nov 2023 changes:

- Add a required field “requires_uninstall_first” in patch_aggregation.json.

03 Aug 2023 changes:

- vuln_associations.json added an optional field "v4_signatures".

27 Jun 2023 changes:

- patch_system_aggregation.json added field "title".