# A Comparison of Detection Methods for Vulnerabilities (CVE) on Linux Systems

Bc. Jan Dobeš

Master's thesis
2022

Tomas Bata University in Zlín
Faculty of Applied Informatics

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

# ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Jan Dobeš** |
| Osobní číslo: | **A19348** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Forma studia: | **Kombinovaná** |
| Téma práce: | **Porovnání metod detekce zranitelností (CVE) na linuxových systémech** |
| Téma práce anglicky: | **Comparison of Detection Methods for Vulnerabilities (CVE) on Linux Systems** |

## Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Nastudujte využití Open Vulnerability and Assessment Language (OVAL®) metadat pro detekci zranitelností.
3. Pomocí vlastní aplikace/knihovny otestujte úspěšnost detekce zranitelností s využitím OVAL metadat.
4. Srovnejte úspěšnost s metodou využívající pouze metadata linuxových softwarových repozitářů.
5. Určete, zda se obě dvě metody navzájem doplňují, nebo zda jedna metoda může plně nahradit druhou.
6. Proveďte diskuzi nad výhodami a nevýhodami obou metod a možnosti integrace do systému VMaaS.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Common Vulnerabilities and Exposures, The MITRE Corporation [online]. [cit. 2020-11-27]. Dostupné z: https://cve.mitre.org/index.html
2. Open Vulnerability and Assessment Language, The MITRE Corporation [online]. [cit. 2020-11-27]. Dostupné z: https://oval.mitre.org/index.html
3. Red Hat and OVAL Compatibility [online]. [cit. 2020-11-27]. Dostupné z: https://access.redhat.com/articles/221883?extIdCarryOver=true&sc_cid=701f2000001OH6kAAG
4. Evolving OVAL in Red Hat [online]. [cit. 2020-11-27]. Dostupné z: https://www.redhat.com/en/blog/evolving-oval
5. Collection of OVAL definitions from several sources [online]. [cit. 2020-11-27]. Dostupné z: https://www.itsecdb.com/oval/
6. Red Hat Product Security Center [online]. [cit. 2020-11-27]. Dostupné z: https://access.redhat.com/security/
7. RPM Versioning Guidelines, Fedora Project [online]. [cit. 2020-11-27]. Dostupné z: https://docs.fedoraproject.org/en-US/packaging-guidelines/Versioning/
8. VMaaS Project Documentation [online]. [cit. 2020-11-27]. Dostupné z: https://github.com/RedHatInsights/vmaas/blob/master/README.md

| | |
|---|---|
| Vedoucí diplomové práce: | **doc. Ing. Martin Sysel, Ph.D.**<br>Ústav počítačových a komunikačních systémů |
| Datum zadání diplomové práce: | **3. prosince 2021** |
| Termín odevzdání diplomové práce: | **23. května 2022** |

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
①

L.S.

**doc. Mgr. Milan Adámek, Ph.D.** v.r.
děkan

**prof. Mgr. Roman Jašek, Ph.D., DBA** v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Ve Zlíně                                    Jan Dobeš, v. r.

podpis autora

## ABSTRAKT

Tato práce porovnává metody detekce zranitelností (CVE) na linuxových systémech. Zaměřuje se na využití Open Vulnerability and Assessment Language (OVAL®) metadat a srovnání úspěšnosti detekce s metodou využívající pouze metadata linuxových softwarových repozitářů v systému VMaaS. Cílem je určit, zda se obě dvě metody navzájem doplňují, nebo zda jedna metoda může plně nahradit druhou. Praktická část zpracovává OVAL® metadata a představuje samostatně fungující aplikaci vyhodnocující zranitelnosti systémů za pomoci těchto metadat. Následně jsou porovnány výsledky s výstupem z veřejně dostupného VMaaS API.

Klíčová slova: zranitelnost, CVE, RPM, OVAL, bezpečnost

## ABSTRACT

This thesis compares methods of CVE vulnerability detection on Linux systems. It's focused on usage of Open Vulnerability and Assessment Language (OVAL®) metadata and comparison with method using only Linux software repository metadata in the VMaaS system. Goal of the thesis is to decide if both methods complement each other or if one method superseeds the second. The practical part processes OVAL® metadata and presents a standalone application capable of evaluating system vulnerabilities using these metadata. Results are then compared to the output of the publicly available VMaaS API.

Keywords: vulnerability, CVE, RPM, OVAL, security

Rád bych poděkoval vedoucímu mé diplomové práce doc. Ing. Martinovi Syslovi, Ph.D. za vedení a cenné rady při tvorbě této práce.

**TABLE OF CONTENTS**

# INTRODUCTION

Vulnerability detection and patching is a never-ending effort in today's software world. Everyone can be affected by security flaws in software which can be misused by various third parties. Even though every individual can be targeted by computer criminals, it's more likely for bigger companies. To keep all software secure and stable should be one of the biggest priorities for all businesses to avoid data leaks or financial loss.

But the task is not as easy as to make sure all software is installed in latest version, this could make computer systems unstable as various types of updates for software are released. This is why software providers need to triage all updates by type, severity, etc. and correctly link them with fixed vulnerabilities. System administrators can then update only necessary parts of a system to avoid regressions in system functionality. Sometimes it's only known that some software is vulnerable but there is no available update for it. Full remediation of the vulnerability is not possible in this case but the information is still valuable as system administrator can mitigate the vulnerability (e.g. by changing configuration of the vulnerable application).

This thesis deals with vulnerability detection on Linux systems, specifically on RPM-based systems. It attempts to extend existing VMaaS service to start using OVAL metadata for better vulnerability detection. The goal is to compare efficiency and other differences between the current VMaaS service implementation (based on RPM repository metadata) and the new extension using OVAL metadata. The assumption is that both methods of detection will complement with each other and more possible vulnerabilities will be detected, if it's true, then also advantages and disadvantages of integrating this solution with VMaaS will be discussed.

# I. THEORETICAL PART

# 1 Vulnerability

Vulnerability is a flaw in software, firmware, hardware or software component. It's a weakness that can be exploited (accidentally or intentionally) and this action can have negative impact on confidentiality, integrity or availability of impacted components. [1]

This thesis deals with software vulnerabilities only, as they are considerably easier detect and repair than firmware and hardware vulnerabilities (although these can be sometimes mitigated on software level). The nature of open-source software also helps with vulnerability detection and also with a process of creating patch.

Common software vulnerabilities:

- Unsafe memory access (e.g. buffer overflow)
- Input validation errors (e.g. code injection, SQL injection, cross-site scripting)
- Privilege escalation (software grants to the user more permissions than user should have)

Software which is taking advantage of a vulnerability is called exploit. Often is linked with criminal activity and allows attackers to access protected systems or install their malicious code to affected systems. Even if some exploits aren't used for criminal activity, their use is always at least morally problematic, acting against the will of original software authors or breaking license agreements.

With increasing number of new software, and thus new vulnerabilities discovered in software, there was a need to have a common system to identify vulnerabilities across various services, tools, and databases. The CVE is current standard to identify, define and catalog publicly disclosed cybersecurity vulnerabilities, established by the MITRE Corporation in 1999. [2]

## 1.1 CVE

Common Vulnerabilities and Exposures (CVE) is a system providing reference and definition for publicly disclosed vulnerabilities. The CVE abbreviation is also used in the sense of some single vulnerability itself. Today, each CVE is defined by an identifier in format `CVE-YEAR-IDENTIFIER` (where `YEAR` stands for a year and `IDENTIFIER` for incremental numeric identifier), brief description and references to various services or databases. The CVE itself is not a database, it's an unique identifier for a vulnerability to link various CVE databases and other services describing given vulnerability. [2]

### 1.1.1 Severity ranking

Depending how severe given vulnerability is, each CVE is categorized by it's severity. Red Hat CVE database defines following severity levels: [10]

- Critical Impact – given to vulnerabilities that could be easily exploited by a remote unauthenticated attacker and lead to system compromise (arbitrary code execution) without requiring user interaction
- Important Impact – given to vulnerabilities that can easily compromise the confidentiality, integrity or availability of resources
- Moderate Impact – given to vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the confidentiality, integrity or availability of resources under certain circumstances
- Low Impact – given to all other issues possibly having security consequences

### 1.1.2 Common Vulnerability Scoring System

Besides severity, Common Vulnerability Scoring System (CVSS) is used to specify how severe the software vulnerability is. Previously, CVSS version 2 was used for vulnerabilities in Red Hat database until 2016, and currently CVSS version 3.1 is used. [10]

CVSS v3.1 provides a *Base Score* – numeric value from `0.0` (no risk) to `10.0` (highest risk), which is based on sum of individual *Base Metrics* – providing more details about constant aspects of a vulnerability. These *Base Metrics* are: [12]

- `Attack Vector (AV)` – Reflects the context by which vulnerability exploitation is possible, based on the metric value. The *Base Score* will be higher the more remote an attacker can be. Possible values are:

  - *Network (N)* – The vulnerable component is bound to the network stack and the number of possible attackers can be high (including entire internet).
  - *Adjacent (A)* – The vulnerable component is bound to the network stack, but the attack is limited at the protocol level to a logically adjacent topology. The attack must be launched from the same shared physical or logical network, or from within other secured/limited domain.
  - *Local (L)* – The vulnerable component is not bound to the network stack, the attack needs to use local read/write/execute capabilities. The attacker either obtains an access to the system (physical or remote, e.g. using SSH), or relies on user interaction by another person to do actions needed to exploit the vulnerability.
  - *Physical (P)* – The attack requires physical access to the system.

- `Attack Complexity (AC)` – Describes conditions beyond the attacker's control that must exist in order to exploit the vulnerability. The *Base Score* will be higher the lower the complexity is. Possible values are:
  - *Low (L)* – Specialized conditions or circumstances do not exist. The attacker can exploit the vulnerable component repeatably.
  - *High (H)* – Success of the attack depends on conditions beyond the attacker's control.

- `Privileges Required (PR)` – Describes the level of privileges the attacker must possess to successfully exploit the vulnerability. The *Base Score* will be highest if no privileges are required. Possible values are:
  - *None (N)* – The attacker can do the attack unauthorized.
  - *Low (L)* – The attacker requires basic user capabilities that could normally affect only setting and files owned by the user.
  - *High (H)* – The attacker requires significant permissions for the vulnerable component.

- `User Interaction (UI)` – Captures requirement for a human user (other than the attacker) to participate in the successful compromise of the vulnerable component. It determines if the attacker is able to do the attack solely or if an other separate user or a user-initiated process is needed. Possible values are:
  - *None (N)* – The vulnerable component can be exploited without any interaction from any separate user.

  - *Required (R)* – Some action of a separate user is needed before exploiting the vulnerable component. For example, the user needs to initialize the installation of an application and the system is vulnerable only until it finishes.

- `Scope (S)` – Captures whether a vulnerability in one vulnerable component impacts resources in components beyond its security scope. The *Base Score* will be highest if a scope change occurs. Possible values are:
  - *Unchanged (U)* – The exploited vulnerability can only affect resources managed by the same security authority.
  - *Changed (C)* – The exploited vulnerability can also affect resources beyond the security scope of the security authority.
- `Confidentiality (C)` – Measures impact to the confidentiality of the information managed by the exploitable software component. Confidentiality refers to limiting information access only to authorized users and preventing access by unauthorized ones. The *Base Score* will be highest if the loss of the confidentiality is high. Possible values are:
  - *High (H)* – Total loss of confidentiality, all resources within an affected component are exposed to the attacker. Or alternatively, only some restricted information with serious impact is obtained by the attacker (e.g. administrator password).
  - *Low (L)* – Some loss of confidentiality, access to some restricted information is obtained by the attacker, but the impact is limited.
  - *None (N)* – No loss of confidentiality in the vulnerable component.
- `Integrity (I)` – Measures the impact to integrity of a successfully exploited vulnerability. The *Base Score* will be highest if the consequence to the impacted component is high. Possible values are:
  - *High (H)* – Total loss of integrity or a complete loss of protection. For example, the attacker can modify any (or at least some important) files protected by the vulnerable component.
  - *Low (L)* – Modification of data is possible, but the attacker doesn't have control over impact of the modification or the amount of modifications is limited.
  - *None (N)* – No loss of integrity in the vulnerable component.

- `Availability (A)` – Measures the impact to the availability of the impacted component after the vulnerability was exploited. Refers to the loss of availability of the impacted service itself (e.g. networked service) and managed resources. The *Base Score* will be highest if the consequence to the impacted component is high. Possible values are:
  - *High (H)* – Total loss of availability. The attacker is able to fully deny access to resources in the impacted component.
  - *Low (L)* – Reduced performance or interruptions in resource availability. The attacker is not able to completely deny access to legitimate users.
  - *None (N)* – No impact to availability within the impacted component.

All *Base Metrics* for a given CVE can be written as a short string called *Vector*. Example *CVSS v3 Vector* of `CVE-2019-11043`: [11]

```
CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H
```

#### 1.1.3 Databases

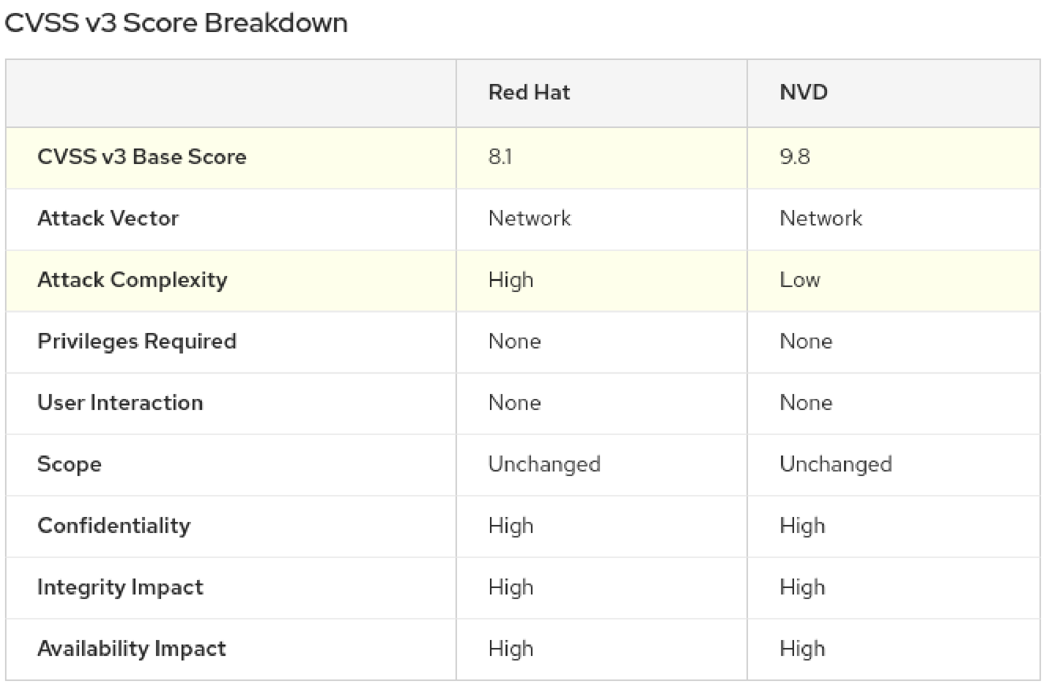
Several organizations and institutions are maintaining their CVE databases. Even though they often use same CVE identifiers and define same attributes to classify CVEs – CVSS score, Attack vector, Published date, etc., these details may vary in each database. Usually the difference is in how various CVE databases rank the CVE severity and exploitability. For example, Red Hat evaluate how affected by given CVE are their environments and products, while other CVE databases can use different environment for their assessment.

These are examples of CVE databases:

- CVE Mitre
- NIST NVD
- Red Hat CVE Database

## 1.2 Exploit

Exploits are programs or code that are designed to leverage a software vulnerability and cause unintended effects. There are already *known* exploits that have already



CVSS v3 Score Breakdown

| | Red Hat | NVD |
|---|---|---|
| CVSS v3 Base Score | 8.1 | 9.8 |
| Attack Vector | Network | Network |
| Attack Complexity | High | Low |
| Privileges Required | None | None |
| User Interaction | None | None |
| Scope | Unchanged | Unchanged |
| Confidentiality | High | High |
| Integrity Impact | High | High |
| Availability Impact | High | High |

Fig. 1.1 Example: different ranking of `CVE-2019-11043` in Red Hat and NIST NVD CVE databases [11]

been discovered by cybersecurity researches. Software developers can patch these exploits and release security updates to users. Then there are *unknown* or *zero-day* exploits created by cybercriminals as soon as they discover a vulnerability. When a *zero-day* exploit attack happens, the vulnerability is unknown to software developers and cybersecurity researchers and they need to quickly figure out how the exploit works and how to patch the vulnerability. [13]

# 2 Package

Software package is a distributable collection of files and their metadata. Every package is usually used for single application, library or some collection of data with common meaning. Probably biggest benefit of using software packages is more control over installed software (knowledge which version is exactly installed on the system, easy upgrades and downgrades, installed application integrity validation).

Also, there is a benefit of library code deduplication – all installed applications are using same version of some installed library, however, this requires carefulness from software maintainers and often makes releasing new versions of software difficult because of impossibility to upgrade some library because some other applications require an older version. For this reason some packaging systems are distributing libraries as separate packages and some packaging systems are not, in this case they are using libraries bundled in the same package as the application.[1)]

## 2.1 RPM

The RPM is a command line driven package format and management system capable of installing, uninstalling, verifying, querying, and updating computer software packages. Each software package consists of an archive of files along with information about the package like its version, a description, and the like. RPM provides library API, permitting advanced developers to manage such transactions from programming languages such as C or Python. [3]

RPM package format is used by various Linux distributions, such as: [3]

- Red Hat Enterprise Linux (RHEL)
- Fedora Project
- CentOS
- SUSE Linux Enterprise (SLES)
- OpenSUSE
- Mageia

[1)]This is related issue to dynamic and static linking during compilation, but on software package level.

- Tizen

### 2.1.1 Versioning

RPM package is identified by NEVRA string. It can be split into 5 components. Epoch component can be omitted, then it's by default considered as `0`. [7] Example:

```
bash-0:4.2.46-21.el7_3.x86_64
```

- `Name` – name of the package (`bash`)
- `Epoch` – incremented when it's needed to reset the version, it ensures that the package can be upgraded to a lower version (`0`)
- `Version` – version of the package, in this case contains 3 version levels with most significant version from the left (`4.2.46`)
- `Release` – number and string usually specifying number of additional patches applied and specific operating system which is this package built for (`21.el7_3`)
- `Architecture` – architecture which is this package built for (`x86_64`)

There are special architecture types, e.g. `noarch` – used for architecure-independent packages or `src` – used for packages containing only source code (these packages are used as an input for the rpmbuild tool together with a SPEC file producing built packages for specific architectures [7]).

### 2.1.2 Dependencies

RPM packages are defining their dependencies in their SPEC file and are categorizing them and their directives by their type: [6]

- Build-time – dependency that is only installed during the RPM build, typically these are requirements for library headers or build tools, which are not required for running the compiled application.
  - *BuildRequires* – directive to request a specific package to be available during the build, it should be an arch-independent dependency to be able to build the package for various architectures.

- Regular run-time – dependency that needs to be installed during the run-time (when end-users install the package)
  - *Requires* – directive to request a specific package to be installed
  - *Obsoletes* – directive to mark a specific package to be removed, it's often used as a mechanism when some package changes it's name to ensure that the old one will be uninstalled
  - *Provides* – directive to indicate that some arbitrary name will be provided by installing the package, it can be used when some other package requires such arbitrary name, for example a feature and there are multiple concurrent packages providing this feature
  - *Conflicts* – directive to mark a specific package as a not compatible with this package, making sure that they can't be installed on a system together, again, it can be used if there are multiple concurrent packages providing same feature to ensure only one of them can be installed at once
- Weak run-time – dependency that can be installed during the run-time, however it's not necessary for the package functionality, these directives are basically variants of the regular `Requires` directive
  - *Recommends*
  - *Supplements*
  - *Suggests*
  - *Enhances*

Example to require a dependent package `foo-lib` for our package `foo`:

```
Requires:  %{pkgname}-lib >= 0:1.2.3-7
```

Note that macros and specific version ranges can be used in directives.

## 2.2 RPM repository

### 2.2.1 Structure

RPM repository is a set of packages and their metadata. They can have various directory structure, are hosted on a server and are usually accessed using HTTP or FTP protocol. The key repository entrypoint is the `repomd.xml` file. This file contains references where in the repository are located other metadata types. Repositories can omit some metadata files, some metadata files also can be available in multiple data representations (XML/SQLite) and can use different compression algorithm – BZip2, Gzip, XZ, etc. These are few selected metadata file types: [8]

- `primary.xml` – used to store metadata about packages in the repository - names, versions, descriptions, checksums, etc.
- `primary_db.sqlite` – used to store same information as the primary.xml but in the SQLite format - allowing for example easier lookups
- `updateinfo.xml` – used to store information about advisories and list of packages belonging to them, also can contain additional advisory details, references to bug trackers, CVE list, etc. (not utilized by all RPM Linux distributions, some prefer other ways of distributing advisory metadata and can omit this file)
- `filelist.xml` – used to store list of files contained in binary RPM packages, using this file it's for example possible to request installation of a specific file and the package manager can quickly find in which package is the file contained
- `modules.yaml` – used to store modules, groups of packages that can be installed or updated as a single unit, it's a recent solution in RHEL 8 how to support different software versions during long support cycles – prior to this feature it was difficult to release new software versions into existing operating system, because the operating system is using dynamically linked libraries and updating any of such libraries for sake of one software could possibly break all other software, this led to overall old software available in repositories

### 2.2.2 Package managers

RPM command line tooling itself focuses only on working with single packages, but for working with repositories more complex package managers are needed. Tools like `YUM`, `DNF` or `Zypper` allow to index repository content, resolve multi-level dependencies between packages during installations, updates and removals and ensure atomicity of transactions.

- `YUM` – Package manager used by RHEL (up to version 7), Fedora (up to version 21) and CentOS (up to version 7) in past. [4]

- `DNF` – Package manager used by RHEL[2], Fedora and CentOS. It's intended as a replacement for YUM providing improved design and better performance. [4]
- `Zypper` – Package manager used by SLES and OpenSuse distributions. [5]

## 2.3 CPE

Common Platform Enumeration (CPE) is a standardized method of describing and identifying classes of applications, operating systems, and hardware devices.

In Red Hat each CPE uniquely identifies the Red Hat platform/product and the version. Example:

```
cpe:/a:redhat:advanced_virtualization:8.5::el8
```

It's important to identify CPEs to accurately assess the impact of a CVE on a particular package or module – for example one package can be vulnerable for one CPE (product + version) but not affected for different CPE. [9]

[2] There is still "yum" command in RHEL 8 for compatibility reasons, it's using DNF as a backend

# 3 Advisory

Advisory (erratum) is an unit for updates. Although software can be updated by updating individual packages, advisories exists to group updates of similar packages or updates fixing same security issue. System administrator can then apply an advisory and all relevant packages will be updated. Advisories are created and published by Linux distribution maintainers. Each advisory consists from identifier, description, severity rankings, attached list of updated packages, list of resolved CVEs and other metadata.

## 3.1 Red Hat Enterprise Linux advisory

The notation of RHEL advisory names (used as unique identifiers) is analogous to CVE naming notation. First component is advisory type, second component is year and the third component is incremental numeric identifier. The full advisory name has a following format: [14]

`TYPE-YEAR-IDENTIFIER`

### 3.1.1 Types

- `RHSA` – Security advisory – contains updates fixing security bugs, e.g. `RHSA-2021-0001`
- `RHBA` – Bug Fix advisory – contains updates fixing normal bugs, e.g. `RHBA-2021-0002`
- `RHEA` – Enhancement advisory – contains other updates (not known if it fixes any bugs), e.g. `RHEA-2021-0003`

Usually, security bugs are fixed by Security advisories but in general they can be fixed by any advisory type. For example, if it's discovered that some CVE is fixed by the package version contained in the advisory, after RHBA or RHEA advisory is released, the CVE can be linked to the advisory attributes and the type of the advisory won't change.

### 3.1.2 Attributes

- `Issued` – timestamp when the advisory was published

- `Updated` – timestamp when there was the last update of advisory attributes (Updated >= Issued)
- `Synopsis` – short description of the advisory
- `Severity` – defined only for security advisories, most likely is derived from severity of CVE(s) fixed by the advisory
- `Topic` – supportive text describing the context of the advisory
- `Description` – description of the content of the advisory, which packages it updates and which issues it solves
- `Solution` – instructions how to apply the advisory, typically referencing some knowledge base article
- `Affected Products` – list of products affected by the advisory
- `Fixes` – references to reported bugs fixed by the advisory
- `CVEs` – references to CVEs fixed by the advisory
- `References` – references to other related resources
- `Updated Packages` – content of the advisory, list of packages that will by updated when the advisory is applied (it doesn't install all packages referenced by the advisory, only those that are already installed in some older version)

# 4 OVAL

The Open Vulnerability and Assessment Language (OVAL) is an XML-based community standard for representing and exchanging security content. Its purpose is to enable the transfer of information across the entire spectrum of security tools and services. [15]

Current version of OVAL specification is *5.11.2*, but definitions in versions *5.11.1* and *5.10* are still available for download. [16]

## 4.1 Concepts

Each OVAL XML file contains several entities which are referenced between each other. Each entity has an *id* and *version* attributes. [15]

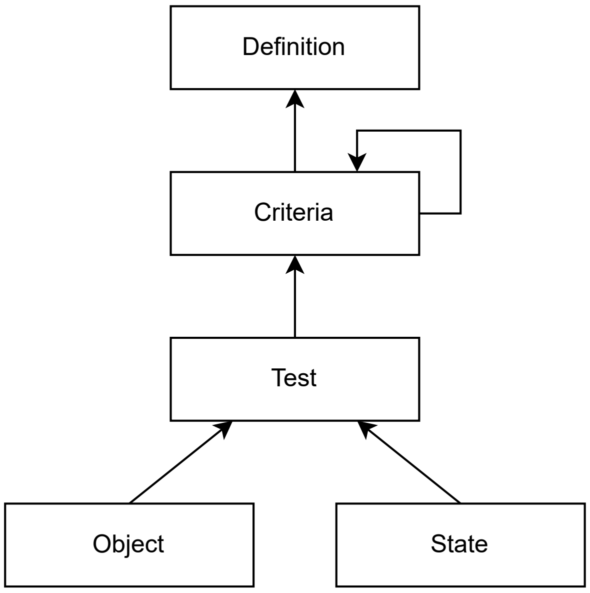


Fig. 4.1 Relations between OVAL entities
[source: own]

### 4.1.1 Definitions

*Definition* is the main entity in OVAL files. Each definition belongs to one of the *classes*: [15]

- Compliance – checking if the endpoint is compliant with specified policy

- Inventory – checking if specified software is installed on the endpoint
- Patch – checking if patch needs to be installed on the endpoint
- Vulnerability – checking if the endpoint is vulnerable
- Miscellaneous – definitions that do not fall into other classes

Security OVAL feed produced by Red Hat consists mostly from *Patch* definitions, Red Hat maps each *Patch* definition one-to-one to published Red Hat Security Advisory. There are also *Vulnerability* definitions in specific OVAL files with "-including-unpatched" string in their name. [17] [18]

Example definition:

```
<definition id="oval:tutorial:def:123" version="1" class="
   inventory">
  <metadata>
    <title>
      CoolWare NET-Suite is installed on the endpoint
    </title>
    <affected family="windows">
      <platform>Microsoft Windows 98</platform>
      <platform>Microsoft Windows 2000</platform>
      <platform>Microsoft Windows XP</platform>
      <product>CoolWare Net-Suite</product>
    </affected>
    <description>CoolWare NET-Suite is installed</description>
  </metadata>
  <criteria>
    <criterion comment="CoolWare iBrowse version 1.0 is
   installed" test_ref="oval:tutorial:tst:1"/>
    <criterion comment="CoolWare eMail version 1.5 is installed"
    test_ref="oval:tutorial:tst:2"/>
  </criteria>
</definition>
```

Inside a definition there is the metadata section – contains a definition title, operating systems and platforms the definition applies to (can be in form of CPEs), a description what is the definition checking, a list of advisories, a list of CVEs and other references (availability of metadata depends on definition class).

The other section of a definition is the *Criteria*. It describes the logical expression which is executed on the endpoint and results into a single boolean value – indicating if the endpoint is compliant, unpatched, vulnerable, etc. Each *Criteria* may contain zero or more *Criterion* or nested *Criteria* and represents a logical operator (`AND` – default if not specified, `ONE`, `OR`, `XOR`). [20]

*Criterion* is referencing a *Test* and represents a term in logical expression defined by *Criteria*.

### 4.1.2 Tests

*Test* defines relationship between and *Object* and zero or more *States*. It matches the expected information with information collected on the endpoint. [15]

`check_existence` and `check` attributes of a test are used to guide the comparison of collected values.

- `check_existence` – Defines how many distinct groupings of information, as defined by the *Object*, must exist on the endpoint for the *Test* to evaluate as `true`. Possible values: [20]
  - `all_exists` – all objects defined by test exist on the endpoint
  - `any_exists` – zero or more objects defined by test exist on the endpoint
  - `at_least_one_exists` – at least one object defined by test exist on the endpoint (default)
  - `none_exists` – none of the objects defined by test exist on the endpoint
  - `only_one_exists` – only one object defined by test exist on the endpoint
- `check` – Defines how many of the collected values must satisfy the requirements given in the *State* for the *Test* to evaluate as `true`. Possible values: [20]
  - `all` – all of the object states are evaluated as true
  - `at least one` – at least one object state is evaluated as true
  - `none satisfy` – none of the object states are evaluated as true
  - `only one` – only one of the object states are evaluated as true

Example test:

```xml
<registry_test id="oval:tutorial:tst:1" version="4" comment="
   CoolWare iBrowse version 1.0 is installed" check_existence="
   at_least_one_exists" check="all" xmlns="http://oval.mitre.org/
   XMLSchema/oval-definitions-5#windows">
  <object object_ref="oval:tutorial:obj:1"/>
  <state state_ref="oval:tutorial:ste:1"/>
</registry_test>
```

### 4.1.3 Objects

*Object* specifies which information should be collected from the endpoint for evaluation. It needs to provide sufficient entities to uniquely identify the endpoint information. [15]

Example object:

```
<registry_object id="oval:tutorial:obj:1" version="3" comment="
   The registry key which holds the version of CoolWare iBrowse"
   xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#
   windows">
  <hive>HKEY_LOCAL_MACHINE</hive>
  <key>SOFTWARE\CoolWare\iBrowse</key>
  <name>Version</name>
</registry_object>
```

### 4.1.4 States

*State* specified the expected value for the *Object*, which is compared to the collected value during *Test* evaluation. [15]

Example state:

```
<registry_state id="oval:tutorial:ste:1" version="2" comment="
   The registry key matches with CoolWare iBrowse version 1.0
   installed" xmlns="http://oval.mitre.org/XMLSchema/
   oval-definitions-5#windows">
  <value>1.0</value>
</registry_state>
```

### 4.1.5 Other entities

There are other smaller entities useful for effective and shorter specification of objects and states. [15]

#### *Variables*

Entity providing a way to define a grouping of one or more values which can be referenced from other OVAL entities. For example can be used as an enumeration for *State* values.

#### *Sets*

This entity provides a way how to express more complex *Objects* which are result of logically combining other *Objects*.

#### *Filters*

This allows the explicit inclusion or exclusion of specific *Objects*. The filter is a *State* referenced from an *Object*.

#### *Regular Expressions*

OVAL Language supports a common subset of the regular expression character classes, operations, expressions, and other lexical tokens defined in Perl 5's regular expression specification. For example, it can be used in a *State* to represent a pattern which an *Object* should match.

## 4.2 Definition files and repositories

Example structure of a definition file: [15]

```
<?xml version="1.0" encoding="UTF-8"?>
<oval_definitions xmlns="http://oval.mitre.org/XMLSchema/
   oval-definitions-5" xmlns:oval="http://oval.mitre.org/
   XMLSchema/oval-common-5">
  <generator>
    <oval:product_name>OVAL-office</oval:product_name>
    <oval:schema_version>5.10.1</oval:schema_version>
    <oval:timestamp>
      2014-10-09T14:11:39.105-04:00
    </oval:timestamp>
  </generator>
  <definitions>
    ...
  </definitions>
  <tests>
    ...
  </tests>
  <objects>
    ...
  </objects>
  <states>
    ...
  </states>
</oval_definitions>
```

OVAL XML files with definitions and dependent entities are categorized by their purpose and gathered in community repositories.

Except the current official repository from Center for Internet Security[1)], there is a number of other repositories from various vendors, e.g.: [21]

- Cisco Systems, Inc.
- Debian Project
- IT Security Database
- NIST Computer Security Division
- Red Hat, Inc.
- SecPod Technologies
- SECURITY-DATABASE
- SUSE

This thesis further focuses only on OVAL data from *Red Hat* repository.

## 4.3 Existing OVAL tools

This section describes already existing tools using OVAL used in *Red Hat* environment and argues about their compatibility with VMaaS application.

### 4.3.1 OpenSCAP

OpenSCAP is an ecosystem providing multiple tools to assist administrators and auditors with assessment, measurement and enforcement of security baselines and policies. The ecosystem consists from multiple layers where higher level tools (GUI tools, daemons, third-party projects integration) depend on tools from lower layers (command-line tools). [22]

1) `https://oval.cisecurity.org/repository/download`

On the base level OpenSCAP provides the `oscap` command-line tool which can check security configuration settings of a system using rules based on various security standards and specifications.

Mainly it's using *SCAP* – line of specifications maintained by the NIST created to provide standardized approach for maintaining system security.

`oscap` tool mainly processes *XCCDF* – standard way of expressing a checklist content, and combines with other specifications like *CPE*, *CCE* and *OVAL* to create SCAP-expressed checklist that can be processed by SCAP-validated products. [23]

***Example usage***

```
# yum install openscap-scanner bzip2
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL8/
    rhel-8.oval.xml.bz2 | bzip2 --decompress > rhel-8.oval.xml
# oscap oval eval --report vulnerability.html rhel-8.oval.xml
$ firefox vulnerability.html &
```

This RHEL 8 example first installs the OpenSCAP scanner on the system. Then downloads the RHEL 8 OVAL specification file and runs the `oscap` command to generate a HTML report. In the last step it checks the HTML report in a web browser. [24]

***VMaaS compatibility***

There are several problems using this tool in VMaaS:

- It's focused to run on a single running system, but VMaaS is a service processing standardized subset of data already collected from a system.
- OVAL definition files contain also *Tests* (chapter 4.1.2) which are intended to execute on running system (checking file contents, file permissions, package signatures). VMaaS doesn't have many of these information and even if it had, it wouldn't be able to say the `oscap` tool where to find these information. Processing all these *Tests* also takes significant time, the run of the `oscap` tool can take up to few minutes, which is not acceptable for VMaaS.

### 4.3.2 CoreOS Clair

Clair is an API-driven analysis engine to monitor the security of Docker containers using a static analysis. It's scanning each container layer and notifies about found vulnerabilities. Vulnerability data in Clair are continuously updated from known sources, it supports not only vulnerability detection from Red Hat packages, but also from Ubuntu, Debian, Python and possibly other packages. [25]

#### *VMaaS compatibility*

Clair purpose for Docker containers is very similar to VMaaS purpose for RHEL systems. Both are using static vulnerability analysis and are maintaining certain vulnerability database. Even if it might make sense to re-use Clair engine for VMaaS, Clair's focus on Docker containers, container registries (Quay) and Red Hat OpenShift Operators [2)], and the fact it's a whole completely different engine with own specifics, would make it difficult to integrate.

[2)] https://www.redhat.com/en/blog/red-hat-quay-32-welcome-container-security-operator

# II. ANALYTICAL PART

# 5 VMaaS

Vulnerability Metadata as a Service (VMaaS) is a micro-service connecting metadata about RPMs, repositories, advisories and CVEs. Its main purpose is to detect vulnerabilities, old installed packages and provide a hint what can or needs be installed to correct these issues. It's an unauthenticated read-only API service which doesn't provide any system inventory management. The API is intended to be integrated with other services (for example those providing the system inventory management), applications and scripts. The micro-service is written in *Python*. [26]

## 5.1 Architecture

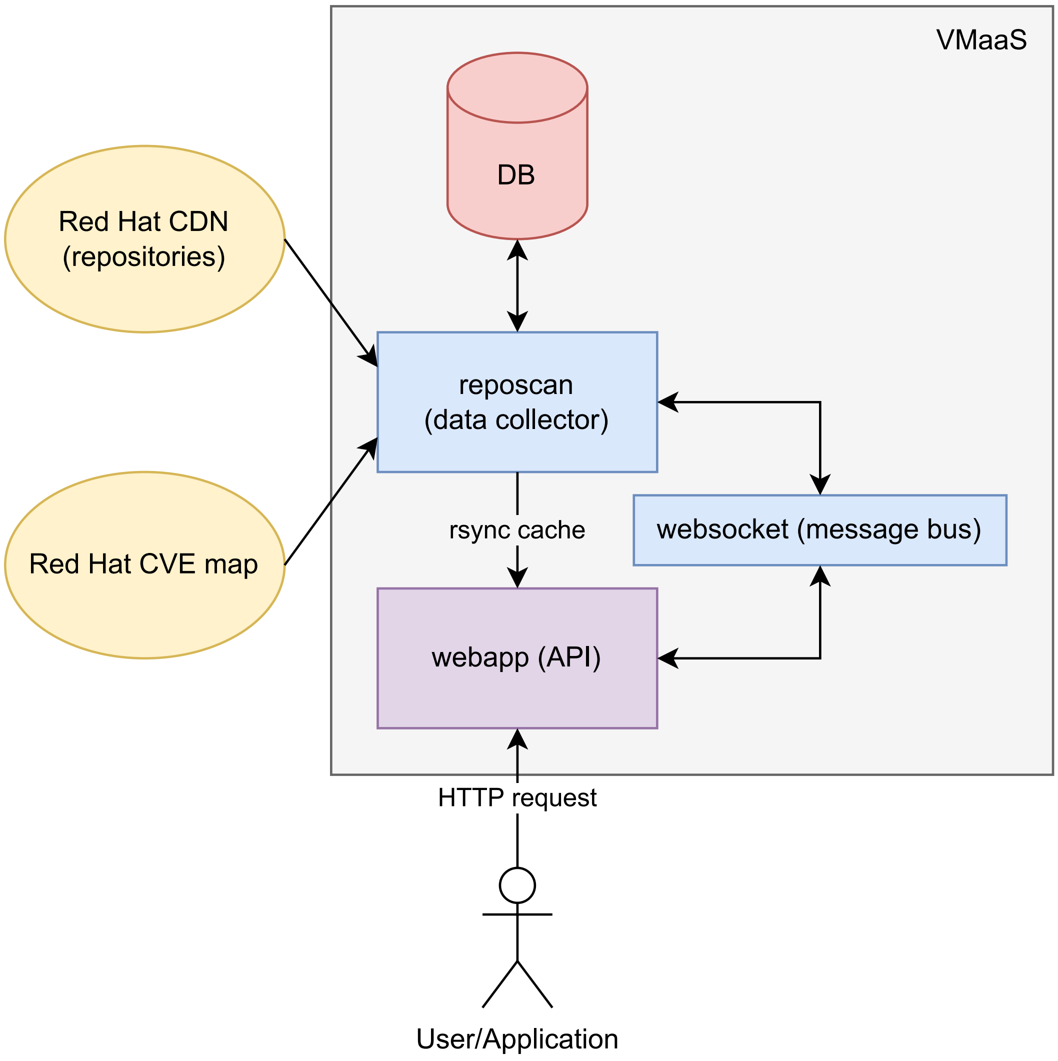


Fig. 5.1 VMaaS components and interactions [source: own]

### 5.1.1 Reposcan

*Reposcan* component acts as a data collector and aggregator from various sources:

- CDN – Directory tree containing all RPM (and also other) repositories published by Red Hat. The directory structure is published on a HTTP server, but is not public and can be accessed only using a client certificate. *Reposcan* doesn't crawl through the directory structure but have a repository list including all paths configured in advance.
- CVE Map – Generated XML file containing metadata of all CVEs known to Red Hat CVE database.

These sources complement each other. While the CDN provides information about packages, repositories and advisories, for CVEs it only provides their identifiers.

The component also contains a non-public management API to set which repositories should be synced, run the synchronization immediately, delete already synced data and other basic maintenance functionality.

The collection of metadata is periodic and happens every few hours, all collected metadata are stored in a *PostgreSQL 12* database. After each metadata sync optimized dump in *SQLite* format is generated, notification is sent to *Websocket* component and the *Webapp* component (listening to this message) fetches the generated dump using `rsync` protocol.

### 5.1.2 Webapp

The component is providing a publicly available API. It's using the optimized dump produced by the *Reposcan* component to operate API endpoints (some of them are described in chapter 5.2). The manipulation with dumps adds certain overhead to the component, but makes it independent on the *Database* and possibly more scalable.

### 5.1.3 Websocket

The *Websocket* component acts as a message bus between *Reposcan*, *Webapp* and also other external services that may be interested to synchronize with VMaaS. There are

2 main purposes of the *Websocket*:

- To inform *Webapp* component when the new optimized dump is prepared.
- To notify all interested services when all *Webapp* instances (in case more than one is running to handle the load) pulled the new dump and updated their internal data structures.

## 5.2 Public API endpoints

This is a simplified overview of the available endpoints, focusing on only those used for the advisory and vulnerability detection:

- `HTTP POST /api/v3/updates`
- `HTTP GET /api/v3/updates/{package_nevra}`
- `HTTP POST /api/v3/patches`
- `HTTP GET /api/v3/patches/{package_nevra}`
- `HTTP POST /api/v3/vulnerabilities`
- `HTTP GET /api/v3/vulnerabilities/{package_nevra}`

Each of these endpoints has two variants:

- `HTTP POST` – Used to provide a list of installed packages, a list of enabled repositories and a list of enabled modules (RHEL 8 systems only) on the system.
- `HTTP GET` – Used to provide a single installed package. It's not possible to provide a list of enabled repositories and a list of enabled modules using these endpoints (the default behavior is to return all possible unfiltered advisories and vulnerabilities for the given package).

And it is true that:

- All mentioned `POST` endpoints are using the same input format (*system profile JSON*). Also all mentioned `GET` endpoints are using the same input (single package NEVRA).

- For each of the `POST` and `GET` pairs (updates, patches, vulnerabilities) the output format is the same.

### 5.2.1 System profile format

`HTTP POST` API endpoints are accepting a *JSON* string representing the system profile.

```
{
  "package_list": [
    "glibc-2.28-127.el8_3.2.x86_64"
  ],
  "repository_list": [
    "rhel-8-for-x86_64-baseos-rpms"
  ],
  "modules_list": [],
  "releasever": "8",
  "basearch": "x86_64"
}
```

Parameters in this system profile represent:

- `package_list` – System has only one package installed. [1)]
- `repository_list`, – System has only one repository enabled.
- `modules_list` – System has zero modules enabled.
- `releasever` – System is subscribed to *8* version of all repositories.
- `basearch` – Systems architecture is *x86_64*. This also helps to determine the subscribed repositories. [2)]

[1)] On real systems there are hundreds of packages at minimum.

[2)] Not needed in this example, because the used single repository is only for the *x86_64* architecture, and even has the architecture mentioned in its label. In general, one repository label can express multiple repositories, combination of the repository label, releasever and basearch expresses always a single one.

### 5.2.2 API responses

Using the system profile to call various endpoints will produce following results.

***Updates endpoint***

```
$ curl -H "Content-Type: application/json"
       -X POST
       -d "@./profile.json"
  https://console.redhat.com/api/vmaas/v3/updates
```

```
{
  "update_list": {
    "glibc-2.28-127.el8_3.2.x86_64": {
      "available_updates": [
        {
          "package": "glibc-2.28-151.el8.x86_64",
          "erratum": "RHSA-2021:1585",
          "repository": "rhel-8-for-x86_64-baseos-rpms",
          "basearch": "x86_64",
          "releasever": "8"
        },
        {
          "package": "glibc-2.28-164.el8.x86_64",
          "erratum": "RHSA-2021:4358",
          "repository": "rhel-8-for-x86_64-baseos-rpms",
          "basearch": "x86_64",
          "releasever": "8"
        },
        {
          "package": "glibc-2.28-164.el8_5.3.x86_64",
          "erratum": "RHSA-2022:0896",
          "repository": "rhel-8-for-x86_64-baseos-rpms",
          "basearch": "x86_64",
          "releasever": "8"
        }
      ]
    }
  }
}
```

The response contains a list of available package updates, for each update the associated advisory (erratum) and the repository where the newer package can be found – only packages from repositories enabled on the system are returned.

***Patches endpoint***

```
$ curl -H "Content-Type: application/json"
       -X POST
       -d "@./profile.json"
  https://console.redhat.com/api/vmaas/v3/patches
```

```
{
  "errata_list": [
    "RHSA-2022:0896",
    "RHSA-2021:1585",
    "RHSA-2021:4358"
  ]
}
```

Compared to the Updates endpoint, this response contains only a list of advisories.

***Vulnerabilities endpoint***

```
$ curl -H "Content-Type: application/json"
       -X POST
       -d "@./profile.json"
  https://console.redhat.com/api/vmaas/v3/vulnerabilities
```

```
{
  "cve_list": [
    "CVE-2016-10228",
    "CVE-2019-25013",
    "CVE-2019-9169",
    "CVE-2020-27618",
    "CVE-2021-3326",
    "CVE-2021-27645",
    "CVE-2021-33574",
    "CVE-2021-35942",
    "CVE-2021-3999",
    "CVE-2022-23218",
    "CVE-2022-23219"
  ],
}
```

The response contains only a list of vulnerabilities associated with advisories.

Mentioned examples are not specifying all parameters usable in the input system profile, and also not all attributes returned in the response *JSON*. Complete specification of all options can be found in the latest version 3 API documentation. [27]

## 5.3 Vulnerability detection

As shown in the previous section, all evaluation endpoints are executing the same evaluation algorithm and the only difference is a slightly different view on resulting data. Using the *Vulnerabilities* endpoint as an example, let's examine the procedure:

1. Validate the input system profile. Filter out packages and repositories that do not exist.
2. Find the list of updates for each package in the input list. The application maintains a sorted list of all versions for each package name across all repositories.
3. Filter this list and keep only updates which are available in repositories enabled on the system.
4. Find the list of advisories associated with each package in the filtered update list.
5. Find the list of CVEs associated with each advisory in the advisory list.
6. Aggregate the CVE list for all input packages, make the list unique and sorted, and return.

This implies an important drawback of the current VMaaS vulnerability detection – it can only detect vulnerabilities which are **already patched**.

# 6 Red Hat OVAL

## 6.1 Repository structure

OVAL definitions version 2 provided by Red Hat [1] are grouped into directories and files by the product they are describing and the operating system version. [19]

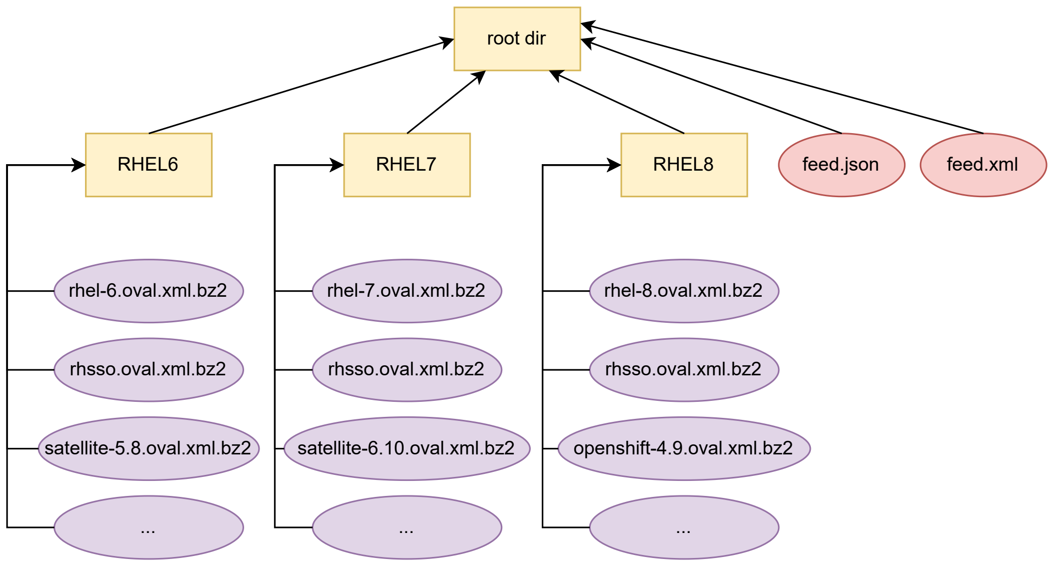


Fig. 6.1 Red Hat OVAL repository directory structure [source: own]

File `feed.json` (respectively equivalent `feed.xml`) acts as the index of the repository. It provides list of all OVAL definition files located in the repository, links to them, last updated timestamp of them, information about their length and format.

Every definition file is currently compressed using `Bzip2` compression and contains custom Red Hat *definitions*, *tests*, *objects* and *states*. Documentation about mentioned custom entities and their attributes is lacking or is obsolete, it's needed to inspect the content of these files directly – and due to the file size rather programmatically.

[1] https://access.redhat.com/security/data/oval/v2/

Shortened example of the `feed.json` with single *entry*:

```json
{
  "feed": {
    "id": "red-hat-vulnerabilities-oval",
    "title": "Red Hat Vulnerabilities OVAL",
    "updated": "2022-04-25T20:37:12.293039",
    "entry": [
      {
        "id": "oval:/RHEL8/rhel-8.oval.xml.bz2",
        "title": "OVAL Definitions",
        "link": [
          {
            "href": "https://access.redhat.com/security/data/oval/v2/RHEL8/rhel-8.oval.xml.bz2",
            "length": 613308
          }
        ],
        "updated": "2022-04-25T17:23:09Z",
        "content": {
          "type": "application/x-bzip2",
          "src": "https://access.redhat.com/security/data/oval/v2/RHEL8/rhel-8.oval.xml.bz2"
        },
        "format": {
          "schema": "https://oval.mitre.org/language/version5.10/index.html",
          "version": "5.10"
        }
      }
    ]
  }
}
```

## 6.2 Definition specifics

There are only definitions of following classes:

- Patch – Most of the definitions in every definitions file. Each definition is mapped one-to-one to the published advisory.
- Vulnerability – Definitions included only in files with the `-including-unpatched` suffix. Each definition appears to be mapped one-to-one to a CVE.
- Miscellaneous – These definitions are located in otherwise empty definition files. They are used as a placeholder until some *Patch* or *Vulnerability* definitions are added and for purposes of this work are irrelevant.

Both Patch and Vulnerability definitions contain a list of affected *CPEs* – indicating a list of operating systems and product versions for which the definition is intended. Evaluating some definition on a different platform than mentioned in the list can produce invalid results – system may appear as vulnerable even if it isn't, or vice versa.

### 6.2.1 Patch class

Patch definition contains one reference to an *advisory* and one or more references to *CVEs*. The meaning is that if the definition evaluates it's *criteria* as `true`, the system is vulnerable to the list of referenced *CVEs* and can install the *advisory* to fix these *CVEs*.

### 6.2.2 Vulnerability class

Vulnerability definition can represent following states: [28]

- *Not affected* – CVE is not affecting given operating system or product. This definition always evaluates as `false`, because *criteria* are contradictory. Example:

```
<criteria operator="AND">
  <criterion comment="openssl is installed" test_ref="
   oval:com.redhat.cve:tst:202123841001"/>
  <criterion comment="openssl is not installed" test_ref="
   oval:com.redhat.unaffected:tst:19990428002"/>
</criteria>
```

- *Affected* – CVE is affecting given operating system or product.
- *Under investigation* – CVE is currently under investigation if it affects given operating system or product.
- *Fix deferred* – CVE is affecting given operating system or product and it may be fixed in the future.
- *Will not fix* – CVE is affecting given operating system or product, but Red Hat has decided to not fix this CVE for this operating system or product version. Mostly CVEs of moderate and low severity are in this category.

Except the *Not affected* state, this information is contained in the definition body, including the list of affected package names. Example:

```
<affected>
  <resolution state="Will not fix">
    <component>wireshark</component>
    <component>wireshark-cli</component>
    <component>wireshark-devel</component>
  </resolution>
</affected>
```

## 6.3 Object, state and test specifics

Red Hat OVAL is using custom object, state and test types to detect vulnerabilities from RPM packages installed on the system, defined in the Linux definitions schema: [29]

- `rpminfo_object` – Defines a *package name* which is evaluated. The schema specifies also optional *behaviors* and *filter* child elements, but they don't seems to be used for any package. Example:

```
<red-def:rpminfo_object id="oval:com.redhat.
   rhsa:obj:20191259001" version="638">
  <red-def:name>dotnet</red-def:name>
</red-def:rpminfo_object>
```

- `rpminfo_state` – Defines a package *architecture*, *epoch* and *version*. The schema specifies other optional child elements, but only the *signature_keyid* is used to check RPM signature. This will not be possible to check in VMaaS, because there is not the information about package signature in the incoming package profile. VMaaS will need to assume that all packages have correct signatures (this matches with the current way of detection). Example:

```
<red-def:rpminfo_state id="oval:com.redhat.
   rhsa:ste:20220496003" version="637">
  <red-def:arch datatype="string" operation="pattern match">
   aarch64|s390x|x86_64</red-def:arch>
  <red-def:evr datatype="evr_string" operation="less than">0
   :6.0.102-1.el8_5</red-def:evr>
</red-def:rpminfo_state>

<red-def:rpminfo_state id="oval:com.redhat.
   cve:ste:201620012001" version="636">
  <red-def:signature_keyid operation="equals">199
   e2f91fd431d51</red-def:signature_keyid>
</red-def:rpminfo_state>
```

  - The `arch` string is represented by either *equals* (single architecture) or *pattern match* (a regular expression), but in practice the regular expression is always `OR` relation similar to this.
  - The `evr` string is bound to an operation which can be either *equals* or *less than* (in practice representing that this exact version is vulnerable or all previous versions are).

- `rpminfo_test` – Defines a relation between a single `rpminfo_object` and zero or more instances of `rpminfo_state`. Most important attributes of the test are *check* and *check_existence* (may not be present and using the default value), described in chapter 4.1.2. Example:

```
<red-def:rpminfo_test check="at least one" comment="dotnet
   is earlier than 0:6.0.102-1.el8_5" id="oval:com.redhat.
   rhsa:tst:20220496005" version="637">
  <red-def:object object_ref="oval:com.redhat.
   rhsa:obj:20191259001"/>
  <red-def:state state_ref="oval:com.redhat.
   rhsa:ste:20220496003"/>
</red-def:rpminfo_test>
```

- `rpmverifyfile_object` – Defines files inside RPM to verify on the system. Red Hat OVAL files are usually only checking only the `/etc/redhat-release` to verify certain version of Red Hat operating system is installed.
- `rpmverifyfile_state` – Defines various information about files inside RPM like package-related attributes to verify file is part of a RPM, or file attributes.
- `rpmverifyfile_test` – Defines a relation between a single `rpmverifyfile_object` and zero or more instances of `rpmverifyfile_state`. This test is an analogy to running `rpm -V` command on a system to verify a RPM package. [29] This test will not be possible to evaluate in VMaaS as there is no information about files installed on the system. However, as this test is practically used only to validate the installed operating version, it doesn't affect the vulnerability detection – VMaaS evaluates vulnerabilities based on the list of repositories in the incoming system profile (repositories are also implying the operating system) and it doesn't make sense to run the test in this scenario.

Red Hat OVAL definition files also contain (less frequently) objects, states and tests from other schemas:

- `textfilecontent54_object` – Defines a specific block in a text file to evaluate. It is used to define a section in module files on RHEL 8 systems. Example:

```
<ind-def:textfilecontent54_object id="oval:com.redhat.
   cve:obj:20177189044" version="636">
  <ind-def:filepath datatype="string">/etc/dnf/modules.d/php
   .module</ind-def:filepath>
  <ind-def:pattern operation="pattern match">\[php\][\w\W]*<
   /ind-def:pattern>
  <ind-def:instance datatype="int">1</ind-def:instance>
</ind-def:textfilecontent54_object>
```

- `textfilecontent54_state` – Defines various file-related entities to check. For example the regular expression of an expected text in the file section specified by the object. Example:

```
<ind-def:textfilecontent54_state id="oval:com.redhat.
   cve:ste:20177189002" version="636">
 <ind-def:text operation="pattern match">\nstream\s*=\s
   *7\.2\b[\w\W]*\nstate\s*=\s*(enabled|1|true)|\nstate\s*=\
   s*(enabled|1|true)[\w\W]*\nstream\s*=\s*7\.2\b</
   ind-def:text>
</ind-def:textfilecontent54_state>
```

- `textfilecontent54_test` – Defines a relation between a single `textfilecontent54_object` and zero or more instances of `textfilecontent54_state`. Implementation of this test can be problematic in VMaaS, file content information is not available. However, there is an information about enabled modules in incoming RHEL 8 system profiles, but it can't be evaluated as a file content. The second usage of this test in Red Hat OVAL is to check if a specific kernel version is set in the `grub.cfg` file to run on the next boot.

```
<ind-def:textfilecontent54_test check="all" comment="Module
   php:7.2 is enabled" id="oval:com.redhat.
   cve:tst:20177189087" version="636">
  <ind-def:object object_ref="oval:com.redhat.
   cve:obj:20177189044"/>
  <ind-def:state state_ref="oval:com.redhat.
   cve:ste:20177189002"/>
</ind-def:textfilecontent54_test>
```

- `uname_object` – Defines an object for the `uname_test`, currently there is only one object without any attributes and child elements, defining system as a whole. [30]

- `uname_state` – Defines an information about machine where the test is running on. It can specify attributes like *machine hardware name*, *host name*, *operating system name*, *release*, *version*, or *processor type*.

- `uname_test` – Defines a relation between a single `uname_object` and zero or more instances of `uname_state`. Evaluating this test is equivalent to parsing output of `uname -a` command on a system. In Red Hat OVAL definition files this test is supposed to validate currently running version of *kernel* (which can be different than a version of the latest installed *kernel* package). Unfortunately, VMaaS is currently doing only a static analysis based on installed packages and doesn't have an information about currently running kernel from incoming system profiles – the system may be in fact vulnerable, if has only the newest *kernel* package installed, but not running it (usually requires a system reboot).

Tab. 6.1 Overview of compatibility with VMaaS
[source: own]

| *Entity* | *Supported by data in VMaaS* |
|---|---|
| `rpminfo_object` | yes |
| `rpminfo_state` | partial, without the *signature_keyid* |
| `rpminfo_test` | yes |
| `rpmverifyfile_object` | no |
| `rpmverifyfile_state` | no |
| `rpmverifyfile_test` | no |
| `textfilecontent54_object` | partial, only RHEL 8 module files |
| `textfilecontent54_state` | partial, only RHEL 8 module files |
| `textfilecontent54_test` | partial, only RHEL 8 module files |
| `uname_object` | no |
| `uname_state` | no |
| `uname_test` | no |

# 7 Adding OVAL support to VMaaS

## 7.1 Benefits

- OVAL could provide a higher detection coverage of fixed CVEs. In some cases there are missing CVEs in advisory metadata.
- OVAL also supports detection of vulnerabilities which do not have a fix. And also provides a reason why there is not a patch for a given known CVE.
- OVAL definition files represent a smaller amount of data than repositories on the Red Hat CDN. This makes periodic pulling of latest data substantially faster.

## 7.2 Problems

- VMaaS is using repository labels to filter the result advisory/CVE list to contain only those affecting the system. But OVAL is using CPE labels to determine which *Definitions* should be evaluated on the system.

  However, there is a way how to map repository labels to CPE labels using provided mapping file published on a Red Hat page. [31]

  For example, for the `rhel-7-server-rpms` repository it can be found a following mapping in the file:

```
{
  "rhel-7-server-rpms": {
    "cpes": [
      "cpe:/o:redhat:enterprise_linux:7::server"
    ]
  }
}
```

  OVAL *Definitions* associated with any of the CPEs from the list then apply for the evaluation.

- VMaaS is using SQL database and it may be difficult to store a logical *Criteria* tree in there.

## 7.3 Scope of changes

- The OVAL integration into VMaaS should not change the overall application workflow, it should be integrated into existing components, where it's possible (extending Reposcan and Webapp components, etc.).

- Format of the input system profile should preferably not change. The OVAL evaluation should not require more data to be collected on systems, as it brings requirements to other applications in the ecosystem to do that.

  However, detection of small part of vulnerabilities may not be available because of that, as described in chapter 6.3. For example, in future, the information about currently running *kernel* version on the system could be easily added. This would enable a support for `uname_*` entities.

# III. PRACTICAL PART

VMaaS is a complicated system consisted from multiple interacting processes[1). For purposes of this thesis simplified standalone application will be implemented. The application will be designed using same programming conventions as VMaaS is using. This means the application will rely only on an OVAL evaluation, will be easy to test, and at the same time will be ready for easy integration into the VMaaS – either will require only minor code changes, or could be potentially also used as an external library.

The practical part is describing an implementation of this application – used technologies, application structure, database schema, examples of usage, etc. And is also demonstrating the functionality on a set of testing system profiles.

1) Intended to run inside Docker containers.

# 8 Technologies

## 8.1 Programming language and environment

As VMaaS is written in the Python 3 and this application is a potential extension for VMaaS, Python 3 was chosen as well. Python is an interpreted programming language and doesn't produce a binary. The application is tested with the Python 3.10 on Fedora 35 Linux distribution. However, it should be compatible also with Python 3.X versions and operating systems (potentially requiring minor changes).

## 8.2 Python dependencies

The application is using mostly modules included in the Python 3 standard library: [32]

- `argparse` – Module implementing an interface to define and parse command line arguments.
- `array` – Module implementing an array type of basic values (`int`, `char`, `float`), which is more memory-efficient than a standard Python `list`.
- `bz2` – Module providing a `Bzip2` compression and decompression API. This is used to decompress the downloaded OVAL definition files.
- `datetime` – Module providing functions for time and date parsing and formatting.
- `json` – Module implementing API for parsing and creating JSON data.
- `logging` – Module providing functions for logging, defining multiple logging levels, etc.
- `os` – Module providing operating system dependent functionality.
- `re` – Module providing support for regular expressions.
- `sqlite3` – Module providing API for working with *SQLite* databases.
- `sys` – Module providing functionality that interacts strongly with the interpreter.
- `time` – Module providing various time-related functions. For example the `sleep` functionality.

- `typing` – Module providing support for Python type annotations.
- `xml.etree.ElementTree` – Module implementing API for parsing and creating XML data.

And couple of modules installed from the Python Package Index (PyPi)[1]:

- `aiohttp` – Asynchronous HTTP client and server. In the application it is used the server part to provide an API endpoint performing the evaluation. [33]
- `requests` – HTTP client library providing a simple interface. In the application it is used to download all metadata files. [34]

## 8.3 Database

The application is using the *SQLite* database. *SQLite* is a small, fast, portable database engine that doesn't require a SQL server running, but it's working as a library inside an application instead. The whole database consists from a single file. [35]

Although VMaaS is using primarily a *PostgreSQL* database server, it's using *SQLite* as a format for cached data loaded in the API. This application simplified this and is using only a *SQLite* database.

Foreign key enforcement is enabled for all database connections. This is not enabled in *SQLite* by default and enabling this can help to detect some database errors early and also makes the behavior more similar to *PostgreSQL* (where it's enabled by default).

```
PRAGMA foreign_keys = ON
```

[1]https://pypi.org/

# 9 Implementation

## 9.1 Database schema

### 9.1.1 Re-used VMaaS tables

The application is re-using some database tables from VMaaS, but with most of their attributes stripped (for example, the `cve` table doesn't contain CVE metadata, only CVE identifiers).

- `package_name` – Table for storing unique package names. Columns:
  - *id* (integer)
  - *name* (text)
- `evr` – Table for storing unique *epoch*, *version* and *release* combinations. Columns:
  - *id* (integer)
  - *epoch* (text)
  - *version* (text)
  - *release* (text)
- `arch` – Table for storing unique architectures. This table contains a static list of values populated during database initialization. No further inserts into this table are done. Columns:
  - *id* (integer)
  - *name* (text)
- `cve` – Table for storing unique CVE identifiers. Columns:
  - *id* (integer)
  - *name* (text)
- `content_set` – Table for storing unique repository labels. Columns:
  - *id* (integer)
  - *name* (text)
- `repo` – Table for storing unique repository label, architecture and releasever combinations. Columns:

  - *id* (integer)
  - *name* (text)
  - *basearch_id* (integer, foreign key to `arch`)
  - *releasever* (text)

Couple of notable tables from VMaaS are not a part of this application, because they are not needed for OVAL evaluation:

- `arch_compatibility` – Table containing architecture pairs, which are compatible to update between each other. It's not usual to update a package to a different architecture package, but it's possible. OVAL can handle this directly in its detection logic.
- `package` – Table containing mapping between package names and EVRs. OVAL is evaluating these entities separately, thus this mapping table is not needed.

### 9.1.2 New tables

- `cpe` – Table for storing unique CPE identifiers. Columns:
  - *id* (integer)
  - *name* (text)
- `cpe_content_set` – Table for mapping CPEs and repository labels. Columns:
  - *cpe_id* (integer, foreign key to `cpe`)
  - *content_set_id* (integer, foreign key to `content_set`)
- `cpe_repo` – Table for mapping CPEs and repository label, architecture and basearch combinations. Columns:
  - *cpe_id* (integer, foreign key to `cpe`)
  - *repo_id* (integer, foreign key to `repo`)
- `oval_operation_evr` – Table for storing supported EVR operations used during the evaluation. Columns:
  - *id* (integer)
  - *name* (text)

This is a static table containing only these text values:

- `"equals"`
- `"less than"`

- `oval_check_rpminfo` – Table for storing supported `check` attributes of OVAL test. Columns:
  - *id* (integer)
  - *name* (text)

  This is a static table containing only these text values:
  - `"at least one"`

- `oval_check_existence_rpminfo` – Table for storing supported `check_existence` attributes of OVAL test. Columns:
  - *id* (integer)
  - *name* (text)

  This is a static table containing only these text values:
  - `"at_least_one_exists"`
  - `"none_exist"`

- `oval_definition_type` – Table for storing supported definition types. Columns:
  - *id* (integer)
  - *name* (text)

  This is a static table containing only these text values:
  - `"patch"`
  - `"vulnerability"`

- `oval_criteria_operator` – Table for storing supported operators used during the criteria evaluation. Columns:
  - *id* (integer)
  - *name* (text)

  This is a static table containing only these text values:

  - `"AND"`
  - `"OR"`

- `oval_stream` – Table used for storing records for individual OVAL definition files, this includes a timestamp of a definition file last update, which can be used for an optimization during a periodic synchronization. Columns:
  - *id* (integer)
  - *oval_id* (text)
  - *updated* (timestamp)
- `oval_rpminfo_object` – Table used for storing `rpminfo` objects. Columns:
  - *id* (integer)
  - *stream_id* (integer, foreign key to `oval_stream`)
  - *oval_id* (text)
  - *package_name_id* (integer, foreign key to `package_name`)
  - *version* (integer)
- `oval_rpminfo_state` – Table used for storing `rpminfo` states. Columns:
  - *id* (integer)
  - *stream_id* (integer, foreign key to `oval_stream`)
  - *oval_id* (text)
  - *evr_id* (integer, foreign key to `evr`)
  - *evr_operation_id* (integer, foreign key to `oval_operation_evr`)
  - *version* (integer)
- `oval_rpminfo_state_arch` – Table used for storing architecture requirements of a `rpminfo` state. Columns:
  - *rpminfo_state_id* (integer, foreign key to `oval_rpminfo_state`)
  - *arch_id* (integer, foreign key to `arch`)
- `oval_rpminfo_test` – Table used for storing `rpminfo` tests. Columns:
  - *id* (integer)
  - *stream_id* (integer, foreign key to `oval_stream`)
  - *oval_id* (text)

  - *rpminfo_object_id* (integer, foreign key to `oval_rpminfo_object`)
  - *check_id* (integer, foreign key to `oval_check_rpminfo`)
  - *check_existence_id* (integer, foreign key to `oval_check_existence_rpminfo`)
  - *version* (integer)

- `oval_rpminfo_test_state` – Table used for storing states associated to a `rpminfo` test. Columns:
  - *rpminfo_test_id* (integer, foreign key to `oval_rpminfo_test`)
  - *rpminfo_state_id* (integer, foreign key to `oval_rpminfo_state`)

- `oval_module_test` – Table used for storing modularity tests used to detect enabled modules on RHEL 8 systems. Columns:
  - *id* (integer)
  - *stream_id* (integer, foreign key to `oval_stream`)
  - *oval_id* (text)
  - *module_stream* (text)
  - *version* (integer)

- `oval_criteria` – Table used for storing criteria used to evaluate definitions: Columns:
  - *id* (integer)
  - *definition_id* (integer, foreign key to `oval_definition`)
  - *operator_id* (integer, foreign key to `oval_criteria_operator`)

- `oval_criteria_dependency` – Table used for representing a tree structure of criteria dependencies. Combination of column values (`NULL` and `NOT NULL`) describes if the child element is another criteria or a test. Columns:
  - *id* (integer)
  - *parent_criteria_id* (integer, foreign key to `oval_criteria`)
  - *dep_criteria_id* (integer, foreign key to `oval_criteria`)
  - *dep_test_id* (integer, foreign key to `oval_rpminfo_test`)
  - *dep_module_test_id* (integer, foreign key to `oval_module_test`)

- `oval_definition` – Table used for representing a definition, the main evaluation unit. Columns:
  - *id* (integer)
  - *stream_id* (integer, foreign key to `oval_stream`)
  - *oval_id* (text)
  - *definition_type_id* (integer, foreign key to `oval_definition_type`)
  - *criteria_id* (integer, foreign key to `oval_criteria`)
  - *version* (integer)
- `oval_definition_test` – Table used for storing a list of `rpminfo` tests used by a definition. This is a redundant table, used only for an optimization in the evaluator application. Columns:
  - *definition_id* (integer, foreign key to `oval_definition`)
  - *rpminfo_test_id* (integer, foreign key to `oval_rpminfo_test`)
- `oval_definition_cve` – Table used for storing a list of CVEs associated with a definition. When a definition evaluates as `true`, the system is vulnerable to these CVEs. Columns:
  - *definition_id* (integer, foreign key to `oval_definition`)
  - *cve_id* (integer, foreign key to `cve`)
- `oval_definition_cpe` – Table used for storing a list of CPEs associated with a definition. This is used in the evaluator application to determine definitions affecting an input system profile. Columns:
  - *definition_id* (integer, foreign key to `oval_definition`)
  - *cpe_id* (integer, foreign key to `cpe`)

The relationship diagram can be found in attachments (1.1).

## 9.2 Application architecture

The application is implemented as a Python module called `vmaas_oval`. It provides:

- 3 commands:

  - Command to download all metadata files into a local directory.
  - Command to extract all metadata files from local directory and to populate the database with relevant data.
  - Command to run an application (HTTP API) performing system profile evaluations.

- Parsers for OVAL feed index, OVAL definition files and CPE to repository mapping.
- Functions populating the database.
- Evaluator algorithm in form of a HTTP server, or a command evaluating a single system profile.
- Infrastructure functions for downloading, decompressing, logging, etc.

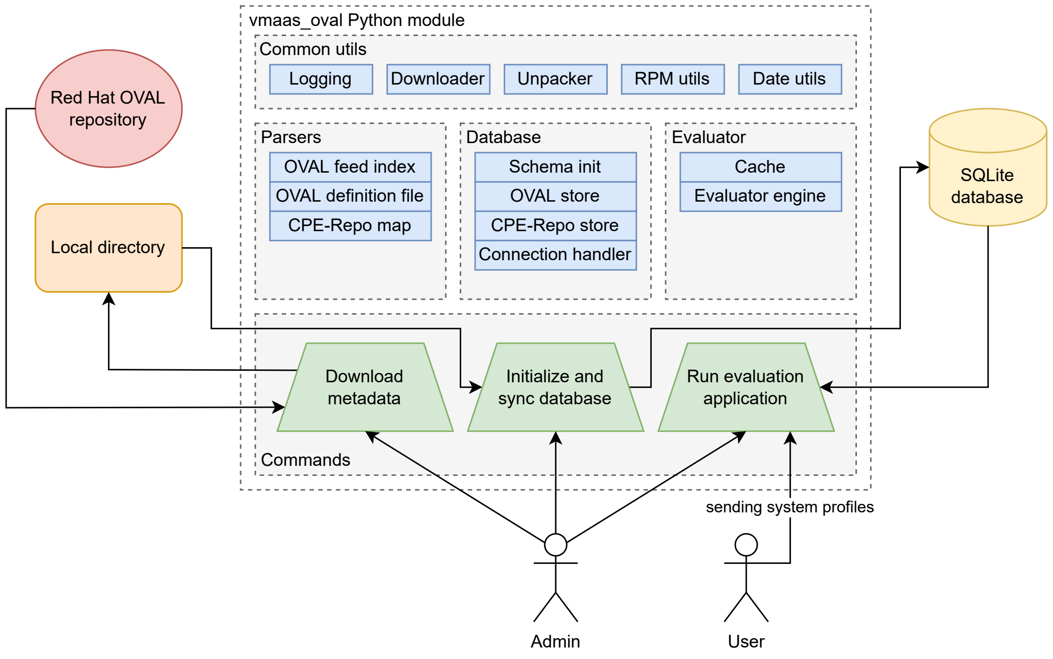


Fig. 9.1 Overview of the application architecture [source: own]

## 9.3 Commands

### 9.3.1 Download metadata

This command downloads all required metadata files into a local directory – the directory can be specified by the `--metadata-dir` parameter (if not specified, files are download into `downloaded_metadata` created in the current directory).

First, it downloads the CPE to repository mapping file, then it continues with downloading the OVAL JSON feed file, where the location of all OVAL definition files is found. In the local directory, directory structure from the source OVAL repository is preserved.

Following features of the downloader are present:

- It currently downloads all metadata files every time the command is executed, existing files are overwritten.
- The downloader supports a retry mechanism in case of download failure. There are a maximum 3 attempts in total.
- Download progress is logged to the standard output.

Currently, around 300 OVAL definition files is downloaded, size of the local directory with downloaded files is around 40 megabytes and the process takes couple of minutes to complete.

Example run:

```
$ python3 -m vmaas_oval.download_metadata --help
```

### 9.3.2 Initialize and populate database

When all metadata files are downloaded into a local directory, initialization and population of the database can take place.

Again, this command is also using the `--metadata-dir` parameter to specify the source directory with downloaded files (if not specified, `downloaded_metadata` is used).

Database file can be specified by the `--database` parameter (if not specified, the default database file is `database.sqlite` in the current directory). If the file doesn't exist, empty database is created.

First, the schema described in chapter 9.1 is initialized, in code it is defined in the `vmaas_oval/database/schema.py` file. All table and static data definitions are written in an idempotent way – the schema initialization can be executed repeatedly, it doesn't change the result.

Then, the CPE to repository mapping is populated – this populates `cpe`, `content_set`, `repo` and respective relation tables. The mapping file contains both content sets and repositories – content sets are a name for only repository labels (majority of mappings), repositories are defined by a label and release version and/or architecture (it's using a specific syntax in the mapping file labels like double under-scores).

And finally, each OVAL definition file is populated one by one.

1. OVAL definition XML file is decompressed.
2. XML is loaded into the parser class. The parser class explicitly mentions list of supported and unsupported OVAL objects, states, tests, etc. If any unexpected type or value occurs in the file, it's logged as a warning to standard output. Parser class keeps successfully parsed information as the class instance attributes.
   - There is a certain simplification in the parsing of modularity tests. Instead of parsing complex regular expressions describing a content of the module file, the parser simply takes the module name from the test comment.
3. Parsed data are stored into the database.
   - The database import code is idempotent, although it doesn't use any ORM.
   - Incremental updates are supported. The `version` field used by OVAL entities is respected and affected rows are updated. Rows from older file versions, which are not present in the latest definition file, are deleted.
   - Insert, update and delete queries are changing many rows at once to achieve a better performance.
   - The application is caching various database ID mappings during the population to decrease a number of performed queries and to achieve a better performance.
   - OVAL definition files, which haven't changed from the last population run, are skipped during the database import. This changed status is obtained by comparing the OVAL definition file `updated` value in the database and the value in metadata file. This behavior can be overridden by adding the `--force` parameter to the command.

Currently, the complete initialization and population of empty database takes couple of minutes.

Example run:

```
$ python3 -m vmaas_oval.initialize_db --help
```

### 9.3.3 Run evaluator application (HTTP API)

This is the final command to run once data are populated into the database. It loads the database file, which can be specified again by the `--database` parameter (if not specified, the default database file is `database.sqlite` in the current directory).

Running the command starts a Python HTTP server provided by the `aiohttp` library. The server is running on port `8000` and is providing a single **HTTP POST** API endpoint – `/vulnerabilities`. This endpoint accepts a system profile JSON compatible with VMaaS format (described in chapter 5.2.1) and returns CVEs affecting the system in two lists:

- `cve_list` – CVEs which have an available advisory to fix the vulnerability. This field is also present in VMaaS.
- `unpatched_cve_list` – CVEs which don't have an available advisory. This field is not in VMaaS, because VMaaS doesn't support unpatched CVE detection.

Whole structure of the response is a following JSON:

```
{
  "cve_list": [string],
  "unpatched_cve_list": [string]
}
```

When the application is started, it loads all data from the SQLite database into Python dictionaries and lists, there is some delay until the application is ready to serve requests. This is the same way how the VMaaS *webapp* works.

When some system profile is sent to the running HTTP server, the evaluation algorithm is following:

1. Validation of the input, the application makes sure a valid JSON is received and is prepared that some fields may be missing.

2. Initialization of two empty sets for resulting vulnerabilities – `cve_list` and `unpatched_cve_list`.

3. Parsing the list of system packages, every package string is separated into a package name, epoch, version, release and architecture. Package names not covered by any OVAL definition are filtered out in this step.

4. Getting the list of candidate OVAL definitions affecting repositories from the system profile. Definitions are obtained using the Repository to CPE mapping and selecting definitions associated with given CPEs. Repositories are defined by labels, and additionally can also be by the `basearch` and `releasever` values in a given system profile (they might not be defined).

   **Note:** If the input repository list is empty, OVAL can't detect vulnerabilities with certainty and the resulting list of CVEs is always empty. This is a same behavior as in VMaaS.

5. Getting the list of enabled modules. They are used later in criteria evaluation for module tests.

   **Note:** Applies only to RHEL 8 system profiles, no other version is using modules currently.

6. For each package, candidate OVAL definitions are found, they are obtained by searching package name associated with `rpminfo` objects, which are linked to tests, which are linked to definitions.

7. Two sets of definitions found in previous two steps are intersected and each of the result set of definitions will be evaluated for a given package.

8. Criteria of all definitions are evaluated recursively.

9. If the result of criteria evaluation is `true`, CVEs associated with the definition are added either to `cve_list` (if the definition type is `patch`) or to `unpatched_cve_list` (if the definition type is `vulnerability`).

10. Result JSON with `cve_list` and `unpatched_cve_list` is returned.

The core of the evaluation is the `rpminfo_test`, which is in most cases comparing the currently installed version (also epoch and release) of a package to the version specified by the test. The package version and release can be basically any string, but usually includes also numbers. To properly compare each pair of matching values the version is parsed to a list of (*number*, *string*) pairs – *number* parts are compared first and if they are equal, then *string* parts are compared. For example the version `1.0.3.beta` is parsed as this list of pairs:

```
[
  (1, ''),
  (0, ''),
  (3, ''),
  (0, 'beta'),
  (-2, '') # indicating end of the list
]
```

Example run:

```
$ python3 -m vmaas_oval.app --help
```

### 9.3.4 Complete workflow example

1. Switch to the working directory.

   ```
   $ cd src/
   ```

2. Install Python dependencies.

   ```
   $ pip3 install -r requirements.txt
   ```

3. Download all metadata files.

   ```
   $ python3 -m vmaas_oval.download_metadata
   ```

4. Initialize and populate the database.

   ```
   $ python3 -m vmaas_oval.initialize_db
   ```

5. Run the HTTP API application in foreground.

   ```
   $ python3 -m vmaas_oval.app
   ```

6. From a different terminal query the evaluation API and get a response.

   ```
   $ curl -s -X POST -d "@./profile.json" http://localhost
       :8000/vulnerabilities
   {
     "cve_list": [
       "CVE-2018-25011",
       "CVE-2020-36328",
       "CVE-2020-36329"
     ],
     "unpatched_cve_list": []
   }
   ```

### 9.3.5 Verbose mode

Each command contains the `--verbose` parameter, which provides additional logs to the standard output. This can be useful for debugging.

# 10 Testing

This section describes testing of the HTTP application. Several simple system profiles were prepared to tests specific conditions and compare the output to the VMaaS public instance[1)].

Downloaded OVAL metadata from *2022-05-09* used for these tests are attached to this thesis, but the public VMaaS instance can return more vulnerabilities for the same package versions in the future.

## 10.1 Compare script

Script `vmaas_compare.py` accepts a single mandatory argument – path to the system profile JSON file. When executed, the script sends the system profile to the local HTTP evaluation server and to the public VMaaS instance. Then prints to the standard output following information:

- Number of CVEs returned from the local server.
- Number of CVEs returned from the public VMaaS instance.
- Number of CVEs without a patch returned from the local server.
- List of CVEs returned from the local server but not from the public VMaaS instance.
- List of CVEs returned from the public VMaaS instance but not from the local server.

Usage:

```
$ python3 vmaas_compare.py ./testing_profiles/profile.json
```

[1)] https://console.redhat.com/api/vmaas/v3/vulnerabilities

## 10.2 Scenarios

### 10.2.1 RHEL 7 system, old kernel

*System profile*

```json
{
  "package_list": [
    "kernel-3.10.0-1160.53.1.el7.x86_64"
  ],
  "repository_list": [
    "rhel-7-server-rpms"
  ]
}
```

*Compare script output*

```
Number of CVEs returned from localhost: 11
Number of CVEs returned from VMaaS: 11
Number of CVEs without a patch returned from localhost: 316
CVEs returned from localhost but not from VMaaS: []
CVEs returned from VMaaS but not from localhost: []
```

***Result:*** Both this application and VMaaS returned exactly same fixable CVEs. These is another 316 known CVEs for the RHEL 7 kernel without an advisory.

### 10.2.2 RHEL 7 system, latest kernel

*System profile*

```json
{
  "package_list": [
    "kernel-3.10.0-1160.62.1.el7.x86_64"
  ],
  "repository_list": [
    "rhel-7-server-rpms"
  ]
}
```

*Compare script output*

```
Number of CVEs returned from localhost: 0
Number of CVEs returned from VMaaS: 0
Number of CVEs without a patch returned from localhost: 316
CVEs returned from localhost but not from VMaaS: []
CVEs returned from VMaaS but not from localhost: []
```

***Result:*** Both this application and VMaaS returned zero fixable CVEs. These is another 316 known CVEs for the RHEL 7 kernel without an advisory.

### 10.2.3 RHEL 8 system, postgresql 12, module enabled

*System profile*

```
{
  "package_list": [
    "postgresql-12.7-1.module+el8.4.0+11288+c193d6d7.x86_64"
  ],
  "repository_list": [
    "rhel-8-for-x86_64-appstream-rpms"
  ],
  "modules_list": [
    {
      "module_name": "postgresql",
      "module_stream": "12"
    }
  ]
}
```

*Compare script output*

```
Number of CVEs returned from localhost: 2
Number of CVEs returned from VMaaS: 2
Number of CVEs without a patch returned from localhost: 0
CVEs returned from localhost but not from VMaaS: []
CVEs returned from VMaaS but not from localhost: []
```

***Result:*** Both this application and VMaaS returned exactly same fixable CVEs. Vulnerabilities only for PostgreSQL 12 were returned, not e.g. for PostgreSQL 13, which would be returned if the modularity test didn't work correctly.

### 10.2.4 RHEL 8 system, postgresql 12, incorrect module

*System profile*

```json
{
  "package_list": [
    "postgresql-12.7-1.module+el8.4.0+11288+c193d6d7.x86_64"
  ],
  "repository_list": [
    "rhel-8-for-x86_64-appstream-rpms"
  ],
  "modules_list": [
    {
      "module_name": "postgresql",
      "module_stream": "999"
    }
  ]
}
```

*Compare script output*

```
Number of CVEs returned from localhost: 0
Number of CVEs returned from VMaaS: 0
Number of CVEs without a patch returned from localhost: 0
CVEs returned from localhost but not from VMaaS: []
CVEs returned from VMaaS but not from localhost: []
```

***Result:*** Both this application and VMaaS returned zero fixable CVEs. This is correct, because the package can't be identified.

### 10.2.5 RHEL 7 system, dnsmasq, OVAL detects more

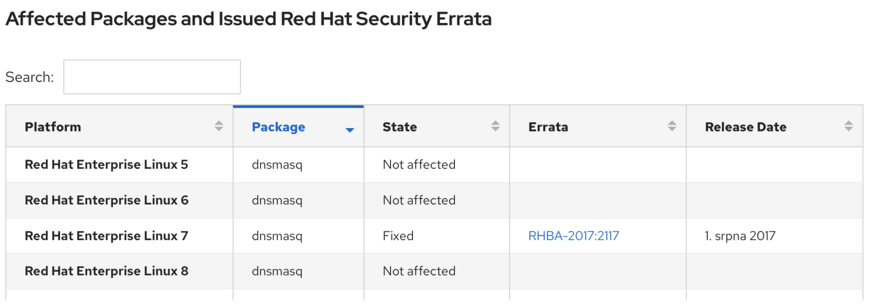
*System profile*

```json
{
  "package_list": [
    "dnsmasq-2.76-1.el7.x86_64.rpm"
  ],
  "repository_list": [
    "rhel-7-server-rpms"
  ]
}
```

***Compare script output***

```
Number of CVEs returned from localhost: 11
Number of CVEs returned from VMaaS: 10
Number of CVEs without a patch returned from localhost: 11
CVEs returned from localhost but not from VMaaS:
  ['CVE-2019-14513']
CVEs returned from VMaaS but not from localhost: []
```

***Result:*** This application returns one more CVE than VMaaS – *CVE-2019-14513*. After examining Red Hat metadata it's clear that the CVE is detected correctly. VMaaS doesn't detect the CVE, because it's not linked from the advisory, probably due to an error. This is a case when OVAL can provide a better detection than VMaaS.

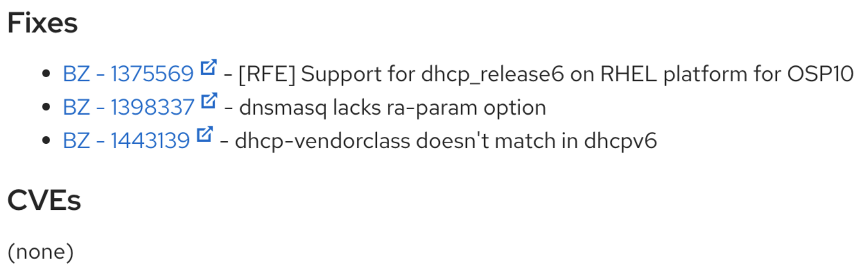


Affected Packages and Issued Red Hat Security Errata

Search:

| Platform | Package | State | Errata | Release Date |
|---|---|---|---|---|
| Red Hat Enterprise Linux 5 | dnsmasq | Not affected | | |
| Red Hat Enterprise Linux 6 | dnsmasq | Not affected | | |
| Red Hat Enterprise Linux 7 | dnsmasq | Fixed | RHBA-2017:2117 | 1. srpna 2017 |
| Red Hat Enterprise Linux 8 | dnsmasq | Not affected | | |

Fig. 10.1 CVE page mentions the advisory [36]



Fixes

- BZ - 1375569 - [RFE] Support for dhcp_release6 on RHEL platform for OSP10
- BZ - 1398337 - dnsmasq lacks ra-param option
- BZ - 1443139 - dhcp-vendorclass doesn't match in dhcpv6

CVEs

(none)

Fig. 10.2 Advisory page doesn't mention the CVE [37]

### 10.2.6 RHEL 7 system, java, repository-CPE mapping inconsistency

***System profile***

```
{
  "package_list": [
    "java-1.8.0-ibm-1:1.8.0.7.0-1jpp.1.el7.x86_64"
  ],
  "repository_list": [
    "rhel-7-server-rpms"
  ]
}
```

***Compare script output***

```
Number of CVEs returned from localhost: 9
Number of CVEs returned from VMaaS: 0
Number of CVEs without a patch returned from localhost: 0
CVEs returned from localhost but not from VMaaS:
  ['CVE-2021-35550', 'CVE-2021-35603', 'CVE-2022-21248',
   'CVE-2022-21293', 'CVE-2022-21294', 'CVE-2022-21340',
   'CVE-2022-21341', 'CVE-2022-21360', 'CVE-2022-21365']
CVEs returned from VMaaS but not from localhost: []
```

***Result:*** In this case the java package update is not available from currently enabled repository *rhel-7-server-rpms*, but it is available from *rhel-7-server-supplementary-rpms*. However, both of these repositories are mapped to the same CPE label. As a result, the application acts as the correct repository would be enabled.

This is a case only for a few repositories, which are distributed together, and to cause any potential problems it requires unusual repository configuration on the system. But it can be considered as a bug in the Repository to CPE mappings.

# CONCLUSION

This thesis compares a vulnerability detection method using metadata of linux software repositories in VMaaS to a detection method using OVAL metadata.

The theoretical and analytical part examined how the OVAL works, what are the existing tools, and what are the differences to the currrent detection method. It also designed how the potential VMaaS extension could look like and which OVAL parts makes sense to implement and which not.

The practical part presented a standalone application capable of downloading metadata from Red Hat public OVAL repository, parsing and importing them to a SQLite database and exposing them in a form of the HTTP API. The implemented HTTP API is capable of full system profile evaluation and accepts the same input format as the VMaaS API. The API was tested using several sample input system profiles and results were compared with the public VMaaS API. The goal was to prove the functional parity, extra features provided by the OVAL metadata and test potentially problematic parts. The implemented application is using a compatible architecture to VMaaS and can be integrated to it with minor modifications.

The OVAL metadata are convenient for vulnerability detection. Their biggest advantage is that they contain also rules for detecting vulnerabilities which do not have an advisory – this is not possible to find out from repository metadata. Also, in practice, they can detect some vulnerabilities which are not linked from repository metadata due to an error.

In the end, OVAL vulnerability detection can fully replace the method using repository metadata. OVAL metadata are represented by a lot smaller amount of data and the application can populate all metadata in matter of minutes. This is an improvement to populating all metadata from repositories, which takes hours.

However, repository metadata still provide some metadata, which OVAL doesn't, e.g. information about individual packages contained in the advisory or in the repository. This is not relevant for the vulnerability detection itself, but can provide more context to the application consumers (users, automated tools). Extra information about the exact package version included in the advisory, or the information in which repositories the advisory can be found, is useful during installation of the fixed package for a given CVE.

Also, there are notable differences between repositories and CPEs, they do not always map one to one, as a result, OVAL may report some extra vulnerabilities with an advisory, but the advisory installation on the system needs additional repository to be enabled on the system.

## REFERENCES

[1] *MITRE: Terminology* [online]. [cit. 2021-02-18]. Available from: `https://cve.mitre.org/about/terminology.html`

[2] *MITRE: Frequently Asked Questions* [online]. [cit. 2021-02-18]. Available from: `https://cve.mitre.org/about/faqs.html`

[3] *About RPM* [online]. [cit. 2021-03-07]. Available from: `https://rpm.org/about.html`

[4] *Why Red Hat's new dnf package manager is not "just another 'yum'"* [online]. [cit. 2021-04-01]. Available from: `https://developers.redhat.com/blog/2016/08/30/why-red-hats-new-dnf-package-manager-is-not-just-another-yum-2/`

[5] *Zypper* [online]. [cit. 2021-04-01]. Available from: `https://en.opensuse.org/Portal:Zypper`

[6] *RPM Packaging guidelines* [online]. [cit. 2021-04-02]. Available from: `https://docs.fedoraproject.org/en-US/packaging-guidelines/`

[7] *RPM Packaging Guide* [online]. [cit. 2021-04-08]. Available from: `https://rpm-packaging-guide.github.io/`

[8] *RPM Repositories* [online]. [cit. 2022-03-04]. Available from: `https://en.opensuse.org/openSUSE:Standards_Rpm_Metadata`

[9] *Common Platform Enumeration (CPE)* [online]. [cit. 2022-04-20]. Available from: `https://redhat-connect.gitbook.io/partner-guide-for-adopting-red-hat-oval-v2/determining-common-platform-enumeration-cpe`

[10] *Understanding Red Hat security ratings* [online]. [cit. 2022-03-04]. Available from: `https://access.redhat.com/security/updates/classification/`

[11] *Red Hat - CVE-2019-11043* [online]. [cit. 2022-03-04]. Available from: `https://access.redhat.com/security/cve/cve-2019-11043`

[12] *Common Vulnerability Scoring System v3.1: Specification Document* [online]. [cit. 2022-03-30]. Available from: `https://www.first.org/cvss/v3.1/specification-document`

[13] *Exploits: What You Need to Know* [online]. [cit. 2022-04-19]. Available from: `https://www.avast.com/c-exploits`

[14] *Red Hat - Product Errata* [online]. [cit. 2022-03-20]. Available from: `https://access.redhat.com/errata/#/`

[15] *OVAL Content Creation Tutorial* [online]. [cit. 2022-03-21]. Available from: `https://ovalproject.github.io/getting-started/tutorial/`

[16] *Repository / Oval Repository* [online]. [cit. 2022-03-23]. Available from: `https://oval.cisecurity.org/repository/download`

[17] *Red Hat and OVAL compatibility* [online]. [cit. 2022-03-25]. Available from: `https://access.redhat.com/articles/221883`

[18] *Technical Guidance on adopting Red Hat OVAL v2* [online]. [cit. 2022-03-25]. Available from: `https://redhat-connect.gitbook.io/partner-guide-for-adopting-red-hat-oval-v2/faqs`

[19] *Red Hat OVAL v2 streams* [online]. [cit. 2022-04-25]. Available from: `https://redhat-connect.gitbook.io/partner-guide-for-adopting-red-hat-oval-v2/red-hat-oval-v2-streams`

[20] *Open Vulnerability and Assessment Language - Element Dictionary* [online]. [cit. 2022-03-26]. Available from: `https://oval.mitre.org/language/version5.11/ovaldefinition/documentation/oval-common-schema.html`

[21] *Other Repositories of OVAL Content* [online]. [cit. 2022-03-28]. Available from: `https://oval.mitre.org/repository/about/other_repositories.html`

[22] *OpenSCAP - Tools* [online]. [cit. 2022-04-12]. Available from: `https://www.open-scap.org/tools/`

[23] *OpenSCAP User Manual* [online]. [cit. 2022-04-12]. Available from: `https://http://static.open-scap.org/openscap-1.2/oscap_user_manual.html`

[24] *Scanning the system for configuration compliance and vulnerabilities* [online]. [cit. 2022-04-19]. Available from: `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardening/scanning-the-system-for-configuration-compliance-and-vulnerabilities_security-hardening`

[25] *What is Clair?* [online]. [cit. 2022-04-19]. Available from: `https://www.redhat.com/en/topics/containers/what-is-clair`

[26] *VMaaS* [online]. [cit. 2022-04-30]. Available from: `https://github.com/RedHatInsights/vmaas/blob/master/README.md`

[27] *VMaaS Webapp - Swagger UI* [online]. [cit. 2022-05-02]. Available from: `https://console.redhat.com/api/vmaas/v3/ui/`

[28] *Red Hat CVE Database Revamp* [online]. [cit. 2022-04-27]. Available from: `https://access.redhat.com/blogs/product-security/posts/2066793`

[29] *Open Vulnerability and Assessment Language - Linux Definitions Schema* [online]. [cit. 2022-04-30]. Available from: `https://oval.mitre.org/language/version5.11/ovaldefinition/documentation/linux-definitions-schema.html`

[30] *Open Vulnerability and Assessment Language - Unix Definitions Schema* [online]. [cit. 2022-04-30]. Available from: `https://oval.mitre.org/language/version5.10/ovaldefinition/documentation/unix-definitions-schema.html`

[31] *How to accurately match OVAL security data to installed RPMs* [online]. [cit. 2022-05-03]. Available from: `https://www.redhat.com/en/blog/how-accurately-match-oval-security-data-installed-rpms`

[32] *The Python Standard Library* [online]. [cit. 2022-05-07]. Available from: `https://docs.python.org/3/library/index.html`

[33] *Welcome to AIOHTTP* [online]. [cit. 2022-05-07]. Available from: `https://docs.aiohttp.org/en/stable/`

[34] *Requests: HTTP for Humans™* [online]. [cit. 2022-05-07]. Available from: `https://docs.python-requests.org/en/latest/`

[35] *About SQLite* [online]. [cit. 2022-05-07]. Available from: `https://www.sqlite.org/about.html`

[36] *CVE-2019-14513* [online]. [cit. 2022-05-18]. Available from: `https://access.redhat.com/security/cve/cve-2019-14513`

[37] *RHBA-2017:2117 - Bug Fix Advisory* [online]. [cit. 2022-05-18]. Available from: `https://access.redhat.com/errata/RHBA-2017:2117`

## LIST OF ABBREVIATIONS

| | |
|---|---|
| CCE | Common Configuration Enumeration |
| CPE | Common Platform Enumeration |
| CVE | Common Vulnerabilities and Exposures |
| CVSS | Common Vulnerability Scoring System |
| NEVRA | Name, Epoch, Version, Release, Architecture |
| NIST | National Institute of Standards and Technology |
| NVD | National Vulnerability Database |
| ORM | Object–relational mapping |
| OVAL | Open Vulnerability and Assessment Language |
| RHEL | Red Hat Enterprise Linux |
| RPM | Red Hat Package Manager |
| VMaaS | Vulnerability Metadata as a Service |

## LIST OF FIGURES

# LIST OF TABLES

## LIST OF APPENDICES

# APPENDIX I. DATABASE RELATIONSHIP DIAGRAM

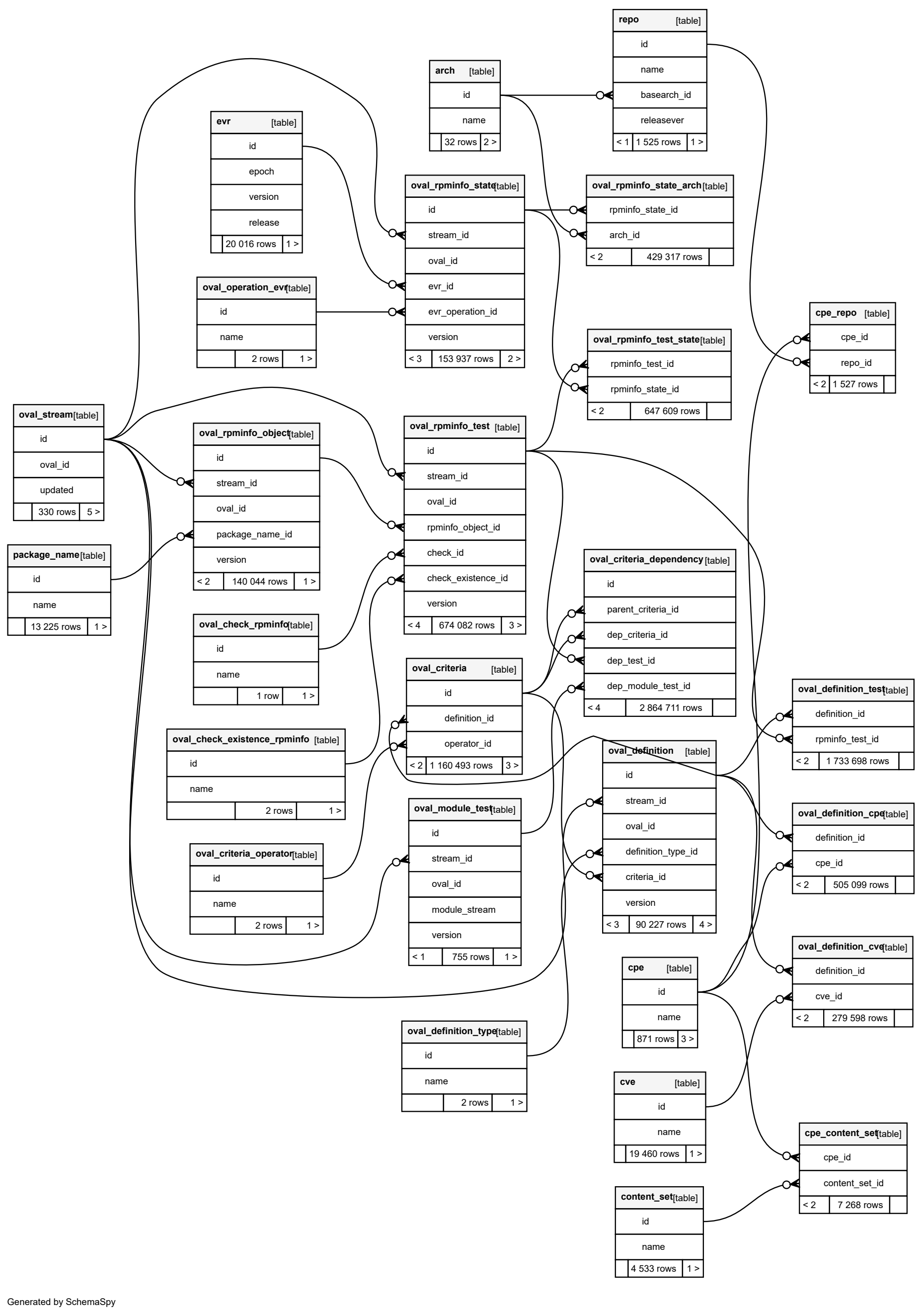


Fig. 1.1 DB diagram generated by SchemaSpy 6.1.0

## APPENDIX II. ATTACHED CD CONTENT

```
$ tree -L 2
.
|--- database_documentation
|    |--- anomalies.html
|    |--- anomalies.js
|    |--- bower
|    |--- column.js
|    |--- columns.html
|    |--- constraint.js
|    |--- constraints.html
|    |--- database.sqlite.database.xml
|    |--- deletionOrder.txt
|    |--- diagrams
|    |--- favicon.png
|    |--- fonts
|    |--- images
|    |--- index.html
|    |--- info-html.txt
|    |--- insertionOrder.txt
|    |--- main.js
|    |--- orphans.html
|    |--- relationships.html
|    |--- relationships.js
|    |--- routines
|    |--- routines.html
|    |--- routines.js
|    |--- schemaSpy.css
|    |--- schemaSpy.js
|    +--- tables
|--- fulltext.pdf
|--- README.txt
|--- src
|    |--- database.sqlite
|    |--- downloaded_metadata
|    |--- LICENSE
|    |--- README.md
|    |--- requirements.txt
|    |--- testing_profiles
|    |--- vmaas_compare.py
|    +--- vmaas_oval
+--- thesis
     |--- graphics
     |--- latexmkrc
     |--- prace.tex
     |--- tex
     +--- zadani.pdf
```