vFeed & vFeed API

The open source cross-linked local **vulnerability database**

A quick-and-dirty user guide

Version beta

May 2013

NJ OUCHN
@toolswatch
www.toolswatch.org

Target Audience

vFeed is appropriate for:

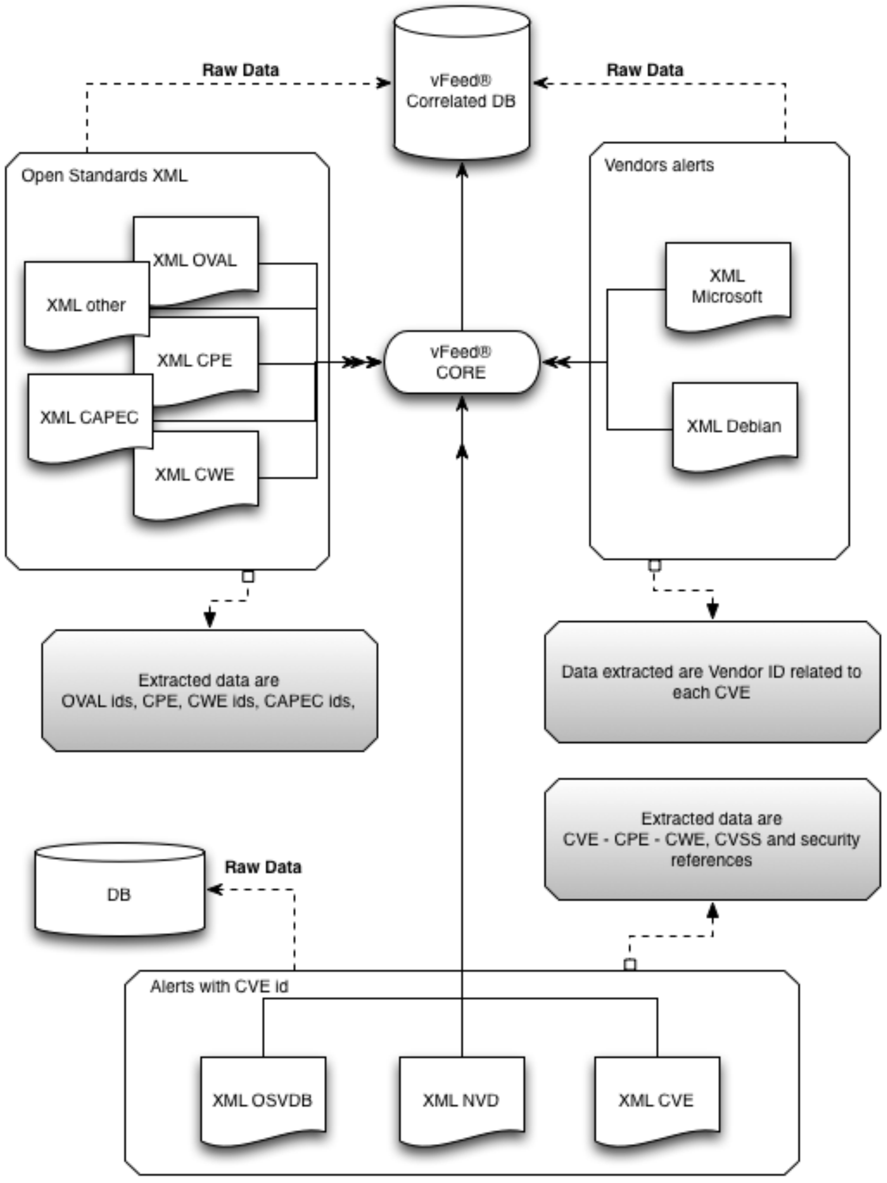
- Penetration testers who want to analyze CVEs and gather extra information to help shape avenues to exploit vulnerabilities.
- Security auditors who want to report accurate information about findings. vFeed could be the best way to describe a CVE with attributes based on standards and 3rd party references as vendors or companies involved into standarization efforts.
- Security tools vendors / security open source developers who need to implement libraries to enumerate useful information about CVEs without wasting time to correlate and to create a proprietary database. vFeed is by far the best solution. Methods can be invoked from programs or scripts with a simple call.
- Any security hacker who is conducting researches and need a very fast and accurate way to enumerate available exploits or techniques to check a vulnerability

Concept introduction

vFeed is an open source naming scheme concept that provides extra structured detailed 3rd parties references for a CVE entry.

While the emergence of the Open Standards helped undeniably to shape a new way to communicate about vulnerabilities[1], the new vFeed is adding an intelligent structured xml feed that provides effective level of information (meta-data) related to vulnerability.

The following schema depicts how the vFeed Database is generated. Indeed, the database is the main asset of this project.



[1] http://measurablesecurity.mitre.org/

Internally, vFeedCore (not published yet) collects the basis xml feeds which are generated by reliable references and correlates it across multiple information sources. Here are examples of 3rd parties sources (just to name a few):

- Security standards
    - CVE (http://cve.mitre.org)
    - CWE (http://cwe.mitre.org)
    - CPE (http://cpe.mitre.org)
    - OVAL (http://oval.mitre.org)
    - CAPEC (http://capec.mitre.org)
    - CVSS (http://www.first.org/cvss)
- Vulnerability Assessment & Exploitation IDs (Metasploit, Saint Corporation, Nessus Scripts, ZDI, Exploit-DB, milw0rm)
- Vendors Security Alerts
    - Microsoft MS
    - Mandriva
    - Redhat
    - Cisco
    - Sun
    - Gentoo
    - Apple
    - …

Key features

- Built using open source technologies
- Fully downloadable SQLite local vulnerability database
- Structured new XML format to describe vulnerabilities
- Based on major open standards CVE, CPE, CWE, CVSS..
- Support correlation with 3rd party security references (CVSS, OSVDB, OVAL…)
- Extended to support correlation with security assessment and patch vendors (Nessus, Exploit-DB, Redhat, Microsoft..)
- Simple & ready to use Python module with more than 15 methods

What is published with vFeed open source package?

- **vFeed.db** : full correlated and aggregated SQLlite vulnerability database.
- **vFeedApi.py** : python library contains methods to generate the vFeed xml format or to be used separately. Sample scripts are attached to help users understanding how it works.
- **vFeedAPI_calls_1.py**: sample python to demonstrate how to call methods from your programs
- **vFeedAPI_calls_2.py :** sample python to demonstrate the usage of vFeed API : calling methods from command line.

What is NOT published (yet)?

- **vFeedCore.py** : python script to extract CVEs and their attributes from NVD and to populate the SQLite database.
- **Library of mappers**: different python script mappers to download, extract, correlate and update accordingly the Sqlite tables with the appropriate references IDs (Microsoft, exploit db, Nessus, OSVDB, OVAL, Redhat, Suse, AIXAPAR ..)

Project dependencies

- vFeedCore (not published)
  - openCVSS.py >> https://github.com/9b/openCVSS library developed by Brandon Dixon
  - Sqlite3
  - Beautifulsoup
  - xml.dom.minidom
  - urllib2

- vFeedApi
  - xml.etree
  - Sqlite3
  - xml.dom.minidom

Project page

http://www.toolswatch.org/vfeed

Project concept idea & author

@toolswatch (http://www.toolswatch.org)

The usage

vFeed project consists of a python library to either use as a module, called from your software, or directly from command line. In addition to the library, an SQLite database that stores different mapping tables is also provided.

Bear in mind that it is always possible to develop your own API in another language. At the time of this writing, the only available library is in python. The reasons for this choice are various:

1. Python is a quite easy to learn and I count on skilled developers and contributors to optimize and to extend the functionalities.
2. Numerous security tools are developed in python or can import adds-on in python. So it will be easy for them to implement the various methods that come with vFeedApi.
3. I'm not a programmer and always hated to write a single line of code. Besides, I always seek for the simplest way to do things with less effort. And python was a great help to make it this way.

The following sample scripts will highlight the way you should use vFeedApi library. For more information about the available methods and what results they return, please refer to "***vFeed API Method***s" chapter. It discusses the requirements in detail.

Note: For this first beta release, the only parameter provided is the CVE ID that facilitates the usage of the API.

API methods calls

The **vFeedAPI_calls_1.py** demonstrates the ability to call a method, which will be querying the data from your own programs by importing the appropriate library. Let's see how it operates through some examples.

Ex 1: Checking for CVSS v2 scores

```
 (1)  import vFeedApi

 (2)  CVE_from_A_SCAN= "CVE-2007-6439"


 (3)  cvssBase,cvssImpact,cvssExploit = vFeedApi.checkCVSS(CVE_from_A_SCAN)

print '\t [cvss_base]:', cvssBase
print '\t [cvss_impact]:',cvssImpact
print '\t [cvss_exploit]:',cvssExploit
```

(1) Import the library vFeedApi to load the appropriate methods.

(2) This is the CVE you need to check CVSS scores for

(3) Checking CVSS is mapped in vFeedApi to `checkCVSS()` method call. This returns 3 values: `cvssBase`, `cvssImpact` and `cvssExploit`. Therefore, you can use these values henceforth in your script.

The values should be as following:

```
cvssBase = 6.1

cvssImpact = 6.9

cvssExploit = 6.5
```

Ex 2: Checking for Exploit-DB IDs.

Let's say we want to enumerate the PoC that we can leverage to exploit « SQL injection vulnerability in forums.php in CMScout 1.23 a ». Any decent scanner will probably flag this vulnerability as **CVE-2007-3812**

```
(1)  import vFeedApi

(2)  CVE_Report_from_a_VAT= "CVE-2007-3812"

(3)  cveEDB_id,cveEDB_file = vFeedApi.checkEDB(CVE_Report_from_a_VAT)

(4)  for i in range(0,len(cveEDB_id)):
         print '[edb_id]:', cveEDB_id[i]
         print '[edb_exploit]:', cveEDB_file[i]
```

(1) Import the library vFeedApi to load the appropriate methods.
(2) This is a random CVE. Let's assume it has been reported by any vulnerability scanner (or found during an assessment)
(3) Invoke the method *CheckEDB* which returns 2 values: `cveEDB_id` and `cveEDB_file` . They are the Exploit-DB ID and the URL link to download the exploit.
(4) This method may return a tuple containing one or multiple values. Therefore, `cveEDB_id` and `cveEDB_file` are lists. So, you have to loop until you extract all values.

The result may look like

```
[edb_id]: 4182
[edb_exploit]: http://www.exploit-db.com/exploits/4182
```

Call from command line

vFeedAPI_calls_2.py lists and tests the available methods. It could be used as command line to check for CVE attributes and meta-data or to export information into the vFeed XML format (*refer to the next chapter*)

```
[ver] vFeed Beta 1.0
[info] usage: vFeedAPI_calls_2.py <API Method> <CVE id>

[info] available API methods:
checkCVE | checkCPE | checkCVSS | checkCWE | checkReferences | checkRISK
checkOVAL | checkNESSUS | checkEDB
checkMS | checkKB | checkAIXAPAR | checkREDHAT | checkSUSE
exportXML (for exporting the vFeed XML file)
```

The script is very easy to use. 2 parameters are mandatory: the **method** and the **CVE ID** you want to analyze.

To enumerate the numerous methods already implemented, just type the script name without parameters as shown in the syntax listing above. For now, I have implemented 15 methods. The following example will help you to better understand how it works.

Ex 1: Let's check information for CVE-2007-5200 and then grab the more attributes (CPE, CVSS, risks …) as possible. For this, we will leverage different methods.

```
(1) $ python vFeedAPI_calls_2.py checkCVE CVE-2007-5200
(2)[cve_description]: hugin, as used on various operating systems including SUSE openSUSE
10.2 and 10.3, allows local users to overwrite arbitrary files via a symlink attack on
the hugin_debug_optim_results.txt temporary file.
[cve_published]: 2007-10-14T14:17:00.000-04:00
[cve_modified]: 2008-11-15T00:00:00.000-05:00
```

(1) We call the script with the valid syntax. In this case: the `checkCVE` method and the CVE `CVE-2007-5200`
(2) Returned values are Description, Published and modified date. In fact, this method returns 3 values. Refer to chapter "vFeed API Methods" for more information about the methods.

As you may notice, the method shows a description about SUSE vulnerability by using `checkCVE`. Let's verify the vulnerable targets. For this, we will use `checkCPE` method.

```
$ python vFeedAPI_calls_2.py checkCPE CVE-2007-5200
[cpe_id]: cpe:/o:novell:opensuse:10.3
[cpe_id]: cpe:/o:novell:opensuse:10.2
[stats] CVE-2007-5200 has 2 CPE
```

It's confirmed, 2 OpenSuse targets are prone to this vulnerability. Let's go deeper and see how we can test the validity of this CVE. We will leverage the `checkNESSUS` to enumerate the appropriate Nessus scripts (if any). Thus can further be used to automate the scan of a large network.

```
$ python vFeedAPI_calls_2.py checkNESSUS CVE-2007-5200

[nessus_id]: 27807
[nessus_file]: fedora_2007-2807.nasl
[nessus_name]: Fedora 8 : hugin-0.6.1-11.fc8 (2007-2807)
[nessus_family]: Fedora Local Security Checks
[nessus_id]: 28154
[nessus_file]: fedora_2007-2989.nasl
[nessus_name]: Fedora 7 : hugin-0.6.1-11.fc7 (2007-2989)
[nessus_family]: Fedora Local Security Checks
[nessus_id]: 29231
[nessus_file]: gentoo_GLSA-200712-01.nasl
[nessus_name]: GLSA-200712-01 : Hugin: Insecure temporary file creation
[nessus_family]: Gentoo Local Security Checks
[nessus_id]: 27268
[nessus_file]: suse_hugin-4518.nasl
[nessus_name]: SuSE Security Update:  hugin creates fixed-name file in /tmp (hugin-4518)
[nessus_family]: SuSE Local Security Checks

[stats] CVE-2007-5200 has 4 Nessus testing script(s)
```

Bingo, the method returns 4 Nessus scripts alongside with ids, files, names and families. You have everything to tune your scanner.

```
[nessus_id]: 27268
[nessus_file]: suse_hugin-4518.nasl
[nessus_name]: SuSE Security Update:  hugin creates fixed-name file in /tmp (hugin-4518)
[nessus_family]: SuSE Local Security Checks
```

The customers need to understand how their systems are breakable (or not) but also the best recommendations regardless how to fix the weaknesses. Good news, vFeed comes with a set of methods to extract and correlate the patches from different sources. Let's verify this with our example.

For now, the patch methods are 5 `checkMS | checkKB | checkAIXAPAR | checkREDHAT | checkSUSE`

We will leverage `checkSUSE` and see what it says.

```
$ python vFeedAPI_calls_2.py checkSUSE CVE-2007-5200

[SUSE_id]: SUSE-SR:2007:020

[stats] CVE-2007-5200 has 1 SUSE id(s)
```

Great, Suse has issued a patch to fix CVE-2007-5200 "**SUSE-SR:2007:020**"

More methods to get information about the vulnerability are listed below.

```
$ python vFeedAPI_calls_2.py checkCWE CVE-2007-5200

[cwe_id]: CWE-59

[stats] CVE-2007-5200 has 1 CWE

$ python vFeedAPI_calls_2.py checkCVSS CVE-2007-5200

[cvss_base]: 3.3
[cvss_impact]: 4.9
[cvss_exploit]: 3.4

$ python vFeedAPI_calls_2.py checkRISK CVE-2007-5200
[cve_severity]: Low
[cve_isTopVulnerable]: False
[cve_pcistatus]: Passed
```

vFeed xml sample

The XML format is the flagship feature of the vFeed concept. In fact, it's the "raison d'être" of the project. Once you invoke the `exportXML` method, the library will gather automatically every piece of information regarding the submitted CVE id.

In fact, you have 2 ways to do so. Either invokes the method from your program as explained previously by using the following code

```
import vFeedApi

Your_CVE= "CVE-2007-3091"
vFeedApi.exportXML(You_CVE)
```

or from the command line (using the example program `vFeedAPI_calls_2.py)` by issuing the syntax

```
$ python vFeedAPI_calls_2.py exportXML CVE-2007-3091
[info] vFeed xml file CVE_2007_3091.xml exported for CVE-2007-3091
```

Here you are. The file has been generated as **`CVE_2007_3091.xml`**

Browse it. It's self-explanatory.

```
<?xml version="1.0" ?>
<vFeed xmlns="http://vfeed.toolswatch.org/0.1" xmlns:meta="http://vfeed.toolswatch.org/0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://vfeed.toolswatch.org/0.1 http://vfeed.toolswatch.org/vfeed.xsd">
  <!--#####################################-->
  <!--Generated by vFeedApi.py-->
  <release>
    <name>
      vFeed XML for CVE-2007-3091
    </name>
    <version>
      vFeed Beta 1.0
    </version>
    <author>
      NJ OUCHN
    </author>
    <url>
      http://www.toolswatch.org/vfeed
    </url>
    <date_generated>
      Thu, 16 May 2013 20:28:54
    </date_generated>
  </release>
  <!--#####################################-->
  <!--Entry ID-->
  <entry exported="CVE_2007_3091.xml" id="vFeed-2007-3091">
    <date modified="2012-10-30T22:37:14.983-04:00" published="2007-06-06T17:30:00.000-04:00"/>
    <summary>
      Race condition in Microsoft Internet Explorer 6 SP1; 6 and 7 for Windows XP SP2 and SP3; 6
and 7 for Server 2003 SP2; 7 for Vista Gold, SP1, and SP2; and 7 for Server 2008 SP2 allows remote
attackers to execute arbitrary code or perform other actions upon a page transition, with the
permissions of the old page and the content of the new page, as demonstrated by setInterval
functions that set location.href within a try/catch expression, aka the &quot;bait &amp; switch
vulnerability&quot; or &quot;Race Condition Cross-Domain Information Disclosure
Vulnerability.&quot;
    </summary>
    <cve_ref>
```

```
      http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3091
    </cve_ref>
    <!--#####################################-->
    <!--The Vulnerability References-->
    <references>
      <source id="CERT" reference="http://www.us-cert.gov/cas/techalerts/TA09-160A.html"/>
      <source id="CERT-VN" reference="http://www.kb.cert.org/vuls/id/471361"/>
      <source id="VUPEN" reference="http://www.vupen.com/english/advisories/2009/1538"/>
      <source id="BID" reference="http://www.securityfocus.com/bid/24283"/>
      <source id="MS" reference="http://www.microsoft.com/technet/security/Bulletin/MS09-
019.mspx"/>
      <source id="XF" reference="http://xforce.iss.net/xforce/xfdb/34696"/>
      <source id="VUPEN" reference="http://www.vupen.com/english/advisories/2007/2064"/>
      <source id="BUGTRAQ"
reference="http://www.securityfocus.com/archive/1/archive/1/470446/100/0/threaded"/>
      <source id="SECTRACK" reference="http://securitytracker.com/id?1018192"/>
      <source id="SREASON" reference="http://securityreason.com/securityalert/2781"/>
      <source id="SECUNIA" reference="http://secunia.com/advisories/25564"/>
      <source id="OSVDB" reference="http://osvdb.org/54944"/>
      <source id="OSVDB" reference="http://osvdb.org/38497"/>
      <source id="MISC" reference="http://lcamtuf.coredump.cx/ierace/"/>
      <source id="FULLDISC" reference="http://archives.neohapsis.com/archives/fulldisclosure/2007-
06/0026.html"/>
    </references>
    <!--#####################################-->
    <!--Vulnerable Targets according to CPE-->
    <vulnerableTargets>
      <cpe id="cpe:/a:microsoft:ie:6"/>
      <cpe id="cpe:/o:microsoft:windows_2003_server:sp2::itanium"/>
      <cpe id="cpe:/a:microsoft:ie:6:sp1"/>
      <cpe id="cpe:/o:microsoft:windows_2003_server:sp1::itanium"/>
      <cpe id="cpe:/o:microsoft:windows_vista:-:-:x64"/>
      <cpe id="cpe:/o:microsoft:windows_vista::sp1:x64"/>
      <cpe id="cpe:/o:microsoft:windows_server_2008:-:sp2:x64"/>
      <cpe id="cpe:/o:microsoft:windows_server_2008:-:-:x32"/>
      <cpe id="cpe:/o:microsoft:windows_vista:-:sp2"/>
      <cpe id="cpe:/o:microsoft:windows_server_2008:-:-:x64"/>
      <cpe id="cpe:/o:microsoft:windows_vista"/>
      <cpe id="cpe:/a:microsoft:ie:7.0"/>
      <cpe id="cpe:/o:microsoft:windows_server_2008:-:sp2:x32"/>
      <cpe id="cpe:/o:microsoft:windows_2003_server:sp2::x64"/>
      <cpe id="cpe:/o:microsoft:windows_xp::sp2"/>
      <cpe id="cpe:/o:microsoft:windows_2003_server:sp2"/>
      <cpe id="cpe:/o:microsoft:windows_vista:-:sp1"/>
      <cpe id="cpe:/o:microsoft:windows_xp::sp2:professional_x64"/>
      <cpe id="cpe:/o:microsoft:windows_server_2008:-:sp2:itanium"/>
      <cpe id="cpe:/o:microsoft:windows_2000::sp4"/>
      <cpe id="cpe:/o:microsoft:windows_xp::sp2:professional"/>
      <cpe id="cpe:/o:microsoft:windows_2003_server:sp1::x64"/>
      <cpe id="cpe:/o:microsoft:windows_2003_server:sp1"/>
    </vulnerableTargets>
    <!--#####################################-->
    <!--Risk Scoring Evaluation-->
    <riskScoring>
      <severityLevel status="High"/>
      <cvss base="7.1" cvssvector="not_defined_yet" exploit="8.6" impact="6.9"/>
      <topVulnerable status="False"/>
      <topAlert status="not_defined_yet"/>
      <pciCompliance status="Failed"/>
    </riskScoring>
    <!--#####################################-->
    <!--Patch Management-->
    <patchManagement>
```

```
      <patch id="MS09-019" reference="microsoft"/>
    </patchManagement>
    <!--#####################################-->
    <!--Attack and Weaknesses Categories. Useful when performing classification of threats-->
    <attackPattern>
      <source id="CWE-362" standard="CWE - Common Weakness Enumeration"
title="not_implemented_yet"/>
    </attackPattern>
    <!--#####################################-->
    <!--Assessment and security Tests. The IDs and source could be leveraged to test the
vulnerability-->
    <assessment>
      <check file="http://oval.mitre.org/repository/data/getDef?id=oval:org.mitre.oval:def:6041"
id="oval:org.mitre.oval:def:6041" type="Local Security Testing" utility="OVAL Interpreter"/>
      <check family="Windows : Microsoft Bulletins" file="smb_nt_ms09-019.nasl" id="39341"
name="MS09-019: Cumulative Security Update for Internet Explorer (969897)" type="Remote Security
Testing" utility="Nessus Vulnerability Scanner"/>
    </assessment>
  </entry>
</vFeed>
```

vFeed API methods

The methods, except those for exporting meta-data to files, begin with "check" then followed by a more explicit name (CVE, CPE, CVSS and so on). `checkCWE,` you got it, will be leveraged to enumerate the CWE ids.

vFeedApi available methods are

- `checkCVE`
- `checkCVSS`
- `checkReferences`
- `checkCWE`
- `checkCPE`
- `checkRISK`
- `checkMS`
- `checkKB`
- `checkREDHAT`
- `checkSUSE`
- `checkAIXAPAR`
- `checkOVAL`
- `checkNESSUS`
- `checkEDB`
- `exportXML`

All the following methods accept the CVE id as the ONLY argument.

Method	Purpose	Returned values
checkCVE()	Checks for the CVE id validity	3 values PublishedDate: Publication date ModifiedDate: Modification date vulnDescription: CVE summary (source NVD)
checkCVSS()	Retrieved the pre-computed CVSS v2.0 scores	3 values cvssBase: CVSS v2.0 base score cvssImpact: CVSS v2.0 impact score cvssExploit: CVSS v2.0 exploit score *Note: scores were calculated using openCVSS.py[2] library*
CheckReferences()	Gather the references that come with a CVE (OSVDB, secunia ….)	2 values cveRef_id: The reference name cveRef_Link: the reference link
checkCWE()	Gather the CWE (Common Weaknesses Enumeration) related to the CVE.	List of values cveCWE_id: List of CWE values
checkCPE()	Gather the CPE (Common Platform Enumeration) regarding a CVE	List of values cveCPE_id: List of CPE values
checkRISK()	Evaluate security risk about a CVE. Risk is the severity level (high, moderate or low), checks either the vulnerability is not at the highest level (which means all CVSS scores at 10) and just for fun trigger a PCI metric (according to PCI CVSS calculation)	3 values levelSeverity: high,moderate or low isTopVulnerable: True or False PCIstatus: Passed or Failed
checkMS()	Lists all the MS Patches issued by Microsoft to fix the current checked CVE	List of values cveMS_id: the MS Microsoft ID (ex: MS09-019)
checkKB()	Lists all the MS Bulletin KB released by Microsoft to describe the issue regarding a CVE	List of values cveKB_id: the KB Microsoft ID
checkREDHAT()	Lists all the security bulletins or patches references issued by Redhat to fix the current checked CVE	List of values cveREDHAT_i: the Redhat ID (ex: RHSA-

[2] https://github.com/9b/openCVSS

		2013:0710)
checkSUSE()	Lists all the security bulletins or patches references issued by Suse to fix the current checked CVE	List of values cveSUSE_id: the SUSE ID (ex: openSUSE-SU-2013:0374)
checkAIXAPAR()	Lists all the security bulletins or patches references issued by IBM to fix the current checked CVE	List of values cveAIXAPAR_id: the AIX APAR ID (ex: IX80470)
checkOVAL()	Gather the OVAL ids that could be leveraged to check the state of the CVE. (refer to oval.mitre.org)	2 Lists of values cveOVAL_id:The OVAL id (ex. oval:org.mitre.oval:def:14287) cveOVAL_file: the link to download the file
checkNESSUS()	Gather the Nessus IDs and NASL files that could be leveraged to assess the CVE in the perspective of security testing	4 Lists of values cveNESSUS_id: the Nessus Plugin ID cveNESSUS_file: the Nessus Plugin NASL file cveNESSUS_name: The plugin name cveNESSUS_family; the family name
checkEDB()	Gather the Exploit-DB PoC that could be used to exploit the vulnerability reported with the current checked CVE	2 lists of values cveEDB_id: the EDB id cveEDB_file: The link for downloading the exploit
exportXML	Generate the vFeed XML format file as described previously. This is the core feature of the vFeed concept. The XML contains all available data related to the current CVE. It's somehow the results of exporting all the previous methods.	Export an XML file basd on this format **CVE_2007_3091.xml**

License

vFeed concept & vFeed API are released under the BSD License.

The vFeed Concept & vFeed API is provided under the 3-clause BSD license above.

This license does not apply to the following components:

- openCVSS.py library used to calculate the scores within vFeedCore (not distributed yet)

Last but not least, feel free to do whatever you like with vFeed/vFeedApi as long as you give credit for the author. As reward, you still can offer me a book or just a kind word thanking me for spending my nights and weekends doing this while you were enjoying barbecues & fresh beers.