Speeding up the Vulnerability Checks Development Process

Automated Generation of OVAL® Vulnerability Definitions (Episode1: Microsoft Windows)

Jerome Athias

2017 January – v0.1 - DRAFT

Introduction – What is OVAL

As per The OVAL® Language Specification [0] (Version 5.11.2):

"The Open Vulnerability and Assessment Language (OVAL®) is an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services."

"The OVAL Language provides a common and structured format that facilitates collaboration and information sharing among the information security community as well as interoperability among tools."

It is part of SCAP. [1]

An OVAL Definition is materialized as an XML file, distributed to be used by interpreters such as Vulnerability Scanners.
(*one could think of it as the "YARA for vulnerabilities")*

Use Cases

The OVAL® Language Specification contains various uses cases, here are just two examples for Vulnerability Management:

Use Case Scenario: Collaborating on the Development of a Vulnerability Check

A new critical vulnerability is disclosed by an application vendor and the initial security advisory does not include an authoritative standardized check for the vulnerability. A vulnerability management product vendor quickly develops and distributes an OVAL Definition with a check for the presence of the vulnerability on the platforms the vendor supports for its customers. <snip>

Use Case Scenario: Leveraging a Standardized Patch Check

An operating system vendor releases a new set of patches for its platform and includes standardized patch checks as OVAL Definitions. <snip>

Current State

As of now (over the past few years), only few organizations or individuals have been observed effectively creating and publicly sharing OVAL Definitions.
Even if the language is *(or claimed to be)* supported by the majority of the most well known Configuration & Vulnerability Scanners, its global adoption is still not as significant.

While the OVAL language/model could be seen as quite complex for a new comer (the specification is almost 150 pages), it remains powerful.
Probably due to this 'complexity', only just few tools effectively supports it.
(Efforts from Open Source products such as OVALDi merits to be highlighted.)
Moreover, no really good and user friendly editor is currently available. *

These facts greatly reduce its wide adoption, and do not facilitate the production of OVAL Definitions.
Also, on top of that, needs to be stated that proprietary equivalent to OVAL Definitions are quickly created by commercial vendors of relatively affordable security products (i.e. Vulnerability Scanners).

** Note that XORCISM has been proving to offer a great potential to resolve that*

Current State - Analysis

From this picture and observations made over the past few years, it appears that the number of OVAL Definitions creators/generators/editors is limited to a few.
And **it seems that A LOT of manual tasks still remain involved.**

From my study of the OVAL language (as part of my XORCISM project [2]) and community, I decided to investigate if, and how, it would be technically feasible to speed-up the OVAL content generation process.

A Journey in OVAL (and Microsoft website)

As part of my XORCISM research regarding the creation of an all-in-one Cybersecurity and IT security data model (Ontology+), I came to OVAL, and after a quick review, I decided to create a relational database representation of its model.
This was done relatively quickly.
But to verify that my interpretation and modeling is correct, I use to write code able to parse and entirely ingest original content into my database.
It helps me to fix the database schema, and make sure that I manipulate each piece of information. Basically, if I can recreate the original content purely from my database (after the initial prepopulating of it), I'm confident enough.
Then, the relational representation of it helps my brain better understand the relationships than when reading a Word specification, or XML.

In many cases, it also helped me to discover some hidden or unexpected errors, optimization points, or new potential use cases.

Anyhow, regarding the publication of OVAL content from the community, following the publication of some vulnerability advisories/bulletins, and for the current scope of this article, the Microsoft's ones: I observed a relatively long (from a Cybersecurity point of view) delay between the two.
Without any specific efficient tooling, and quickly understanding that experience with the language is so absolutely critical, I understood that this delay was explainable.
Digging more into it, I also noticed the complex path to access all the information needed to build a new OVAL Definition, from scratch, for Microsoft Patches/Vulnerabilities.

The Challenge

From "*CVE-2017-001*", generates automatically an accurate OVAL Definition XML file. *(in less than 60 seconds)*

Getting information about the Products impacted

CVEs and CPEs

A CVE provide a list of CPEs. Or, in fact, some kind of "CPE patterns".
This statement is because if you would look for the CVE's CPEs into the main CPE database, (Official CPE Dictionary) chances are that you would often not find them directly.
What the CVE provides is something like "CPEs starting with".

With that said, an analysis of the CPE database will tell you that you can obtain a Title for the CPEs.

Examples:

```
<cpe-item name="cpe:/a:microsoft:outlook:2010:sp1:~~~~x64~">
  <title xml:lang="en-US">Microsoft Outlook 2010 sp1 x64 (64bit)</title>
<cpe-item name="cpe:/o:microsoft:windows_server_2008:-">
  <title xml:lang="en-US">Microsoft Windows Server 2008</title>
<cpe-item name="cpe:/o:microsoft:windows_server_2008:-:sp2">
  <title xml:lang="en-US">Microsoft Windows Server 2008 Service Pack 2</title>
<cpe-item name="cpe:/o:microsoft:windows_server_2008:-:sp2:x64">
  <title xml:lang="en-US">Microsoft Windows Server 2008 Service Pack 2 x64 (64-bit)</title>
```

While interesting, this won't really be of great help in our challenge while these titles include the products names with the versions… which would make it difficult to use in our below "matching function" with the OVAL Inventory Definitions.

OVAL Inventory Definitions

An OVAL Inventory Definition, or an OVAL Definition of class "inventory", "describes an OVAL Definition that checks to see if a piece of software is installed on a system. An evaluation result of '*true*', for this class of OVAL Definitions, indicates that the specified software is installed on the system."

This is an important ingredient for the cooking of our Final OVAL Definition.
While later on we will focus on how to check if a specific file (i.e. .dll or .exe) is present on a system, and analyze its version, we would have first to check if the Product (piece of software) is present on the system.

The matching function

At this point, we would know:

- The CVE
- The CPEs (patterns) listed in the CVE
- The real CPEs corresponding to the CPEs patterns (by simply searching with StartsWith() into the central CPE database (Official Dictionary))

So now we need a way to:

- 1) Identify to what Product a CPE corresponds to
- 2) Obtain an OVAL Inventory Definition for this Product(s)

So for 1):
I tried using the CPE.Titles, but it was not really efficient (or too difficult, or not generic enough), due to the difficulty to predict what information, and how it would be represented, regarding the product's "version" part in the CPE.Title.
Examples: Gold or(Gold) Edition, initial release, R2 SP2 vs SP2 R2, etc.

I went to just using the "product" part of the CPEName.

```
[part="a",vendor="microsoft",product="internet_explorer",version="8\.0\.6001",update="beta"]
```

It appeared more efficient, especially while using some hardcoding later on in my code to enhance this simple information with things like "Service Pack 1" (or SP1 – depending the use case), "Release Candidate" (or RC – depending on the use case), x86/x64, R2, Enterprise/Professional, etc. in a specific order.

For 2):
No magic here.
I mean, I don't think there would be an automated, generic way, of creating checks like "is product X installed" by a computer. (homework for IA maybe)
But,
Luckily enough, a couple of cool guys (ladies and gentlemen) (probably waiting for some drinks beers/vodka) already created manually a quite long list (hundreds) of OVAL Inventory Definitions for Microsoft products.

Examples:

OVALDefinitionIDPattern	OVALDefinitionTitle
oval:org.cisecurity:def:698	Microsoft Windows 10 Version 1511 (x64) is installed
oval:org.cisecurity:def:699	Microsoft Windows 10 Version 1511 (x86) is installed
oval:org.cisecurity:def:1377	Microsoft Windows 10 Version 1607 (32-bit) is installed
oval:org.cisecurity:def:1379	Microsoft Windows 10 Version 1607 (64-bit) is installed

So now that we know (in fact assume) that we can find an already created OVAL Inventory Definition for a Microsoft product – and – that we learned how to reconstruct a Microsoft product name (let's say compatible with the OVAL Inventory Definitions names) automatically from a CPE, we can proceed writing our, now infamous, "matching function".

- See the source code.

At this point, we are able, in an automated way, starting from just a CVE number as an input: to identify the Products/Platforms (OS) impacted and use OVAL to check that.

Getting information about the Files impacted/updated

So, my good Mister Watson Sherlock Holmes, the 1M$ question today is:
How automatically retrieve The file impacted by a vulnerability/updated by a patch from a CVE number?

> Getting, downloading and installing the Patch/KB and checking the system pre/post state?
> Well, yeah maybe for grandma playing bindiff/IDA/MSF (this in another Episode maybe) but that would require Swordfish in French-like bandwith and storage on the moon.
>
> *No well, little padawan, look,* Microsoft is smart enough to provide this detailed information *(sometimes for free…) (through various channels…) (into various formats…).*
> *Where is that? Deathstar?*
> Just on Microsoft.com, in the MS- and/or multiple KB pages.
> *Ready for some scraping kungfu?*
> *Let's do this. Just do it (or I will)*

So, if we look at a CVE, it comes with References.
In the case of a Microsoft product, it would come with a Reference (link/URL) to a **MS- webpage**.

> *Ok, easy, IE/wget/Lynx will do it!*
> *Well, not so easy young padawan (remember? Wr talkin bout Microsoft), so – not so fast…*
> *So yeah, the MS- page won't give you The golden file&version.*

You will **get a link to a KB page** (a big one. *Remember? Wr talki…) (ah and no chocolate. sorry, it's patented. Legal story…)*

And the KB will give you… *no idea?*...
many links! *(Wow, how cool is that.)*
to many KBs. *(so, repeat after me: scrap, scrap, scrap, and more scrap)*

> *Ok boy, I'm coding for few centuries now, so few links === piece of cake, != battle vs Eminem.*
>
> *Yeah sure Ruby Yoda, scrap, parse and regex dance party. Fun*
> *But u now what? You Master, with your HTML v0.1betadraft parsing certification, did you heard about this thing: JS, Ajax and co.? funny stuff (eheh Houston… we got a problem)*
> *So listen Billy the Kid, how we gonna get this fu**ing content?*
> *Well, my kid once told me about that toy, **SeleniumHQ [3]**… (we'll learn later it's not a new drug)*

Then,
One would realize *(while browsing through the KBs on its iPad… come on, I know that's what u do when your wife is out) that*

for accessing some KBs (let's say like ones for Windows 10, or Windows Server 2016…) **that would take a Microsoft Account** *(no wait, not that one! A Business/Education Microsoft Account. Aaaah uh yeah, euh, I see… (and if you don't, go get an evaluation one on Office365))*

Anyhow, when one finds it way through the KB pages to the File information section, with the list of updated files, he will find:
The list of modified files (with info like Date and Version) **in an HTML table**
or
The list of modified files (with info like Date and Version) **in a csv file**

- Parsing these lists is straightforward.
- Sorting them by dates also.
- Now being sure that our program will always retrieve automatically The file we would be looking for, each time, is less easy. (This is not covered here for now, while, luckily enough, during testing, it has been the case most of the time.)

Baking the Cake

Now that we do have automatically collected our ingredients (CVE, impacted Platforms and Products versions, OVAL Inventory Definitions for them, Files impacted/updated with their Versions), we can finally put our cake in the oven.
We'll just make sure to reuse (if applicable) or create some OVAL variables, file_state and file_object for our FILE_TEST.
Mix all of that and that's it basically. Voila!

Check in 60 seconds.

References

[0] The OVAL® Language Specification

[1] SCAP
https://scap.nist.gov/

[2] XORCISM
https://github.com/athiasjerome/XORCISM

[3] SeleniumHQ Browser Automation
http://www.seleniumhq.org