[Type the company name]

Security Automation Developer Days – OVAL Discussion

June 8 – 12, 2009

Introduction

The fourth OVAL Developer Days was held in conjunction with the first-ever Security Automation Developer Days from June 8 - 12, 2009 at The MITRE Corporation in Bedford, MA. The purpose of this event was to discuss, in technical detail, some of the more difficult issues facing Version 5 of the OVAL Language, and to explore new use cases for OVAL. By bringing together the lead proponents within the OVAL Community, the hope was to derive solutions that would benefit all parties, and continue the community-led development of the language. What follows is a detailed summary of the discussions from the event.

In addition to a summary of the discussions, a list of action items has been recorded at the end of this document. These items represent topics that were flagged as needing further discussion, or ideas that need to be further worked through based on the discussions held.

Table of Contents

Attendee List

Belarc	-	Richard Defuria
		Gary Newman
		Peter Tracy
BigFix Inc	-	Ben Hobbs
Booz Allen Hamilton	-	Michele Ellison
		Conrad Fernandes
		Jason Smith
		Michael Sor
		Colin Troha
CA	-	Michael Petersen
Calwater	-	Ramesh Dhullipaua
Center for Internet Security	-	Blake Frantz
		Steven Piliero
Concurrent Technologies Corporation (CTC)	-	Kirk Johnson
DISA FSO	-	Jim Govekar
		David Hoon
		Paul Inverso
		Terry Sherald
		Ricki Vanetesse
DOE	-	Peter Wettergreen
DSCI	-	Vladimir Giszpenc
		Leon Sung
EWA-Canada	-	Dawn Adams
Fortinet	-	Raymond Chan
G2, Inc	-	Melanie Cook
		Matthew Kerr
		George Saylor
		Shane Shaffer
		Greg Witte
Gideon Technologies	-	Kyle Key
		Dragos Prisaca
Guidsoft	-	Bill Ren
Hewlett-Packard	-	Pai Peng
		Peter Schmidt
HQDA	-	Lilliam Cruz
Juniper Networks	-	Stephen Hanna

Lockheed Martin	-	Jim Hartig
		Wesley Snell Jr
McAfee, Inc.	-	Kent Landfield
MIT Lincoln Laboratory	-	Stephen Boyer
MITRE	-	Jon Baker
		Steve Boczenowski
		Steve Boyle
		Andrew Buttner
		Maria Casipe
		Michael Chisholm
		Daniel Haynes
		Robert Martin
		Brendan Miles
		Linda Morrison
		Lisa Nordman
		Charles Schmidt
		Larry Shields
		Glenda Turner
		Matthew Wojcik
		Bryan Worrell
		John Wunder
		Margie Zuk
Modulo Security Solutions	-	Marlon Gaspar
nCircle	-	Timothy Keanini
	-	Natalia Smishko
Netiq/Attachmate	-	William Graves
NIST	-	John Banghart
		Paul Cichonski
		Timothy Harrison
		Christopher Johnson
		David Waltermire
NSA	-	Mike Buckley
		Jason Hage
		Joseph Wolfkiel
OpenPages Inc	-	Gary Zakon
Prism Networks Pvt. Ltd.	-	Maneesh Jolly
ProSync Technology Group, LLC	-	Joe Wulf
Qualys, Inc.	-	Parminder Singh

Secure Acuiity, LLC	-	Andrew Bove
Sparta	-	Jim Ronayne
SPAWAR | SAIC | EMA	-	Jeff Cockerill
Symantec	-	Jim Boyles
		Chris Coffin
		Jason Meinhart
Telos	-	Sudhir Gandhe
		Peter Smith
ThreatGuard, Inc.	-	Robert Hollis
Tripwire, Inc	-	Rob Etzel
		Robert Huffman
		Adam Montville
		Claudia McNellis
Unified Compliance Framework	-	Dorian Cougias
Other		Karen Dixon-Brugh

Day 1

Deprecation Policy Review

A review of the new deprecation policy started the OVAL discussions since having a solid understanding of the policy will help in discussing many of the topics at this year's event. Prior to April 2009, the deprecation policy used by the OVAL Language was not adequately defined, and there was no documentation that described how language constructs could become deprecated, what happened to a construct once deprecated, or how a deprecated language construct could be identified. The new "OVAL Language Deprecation Policy" was developed to address these issues and more.

The creation of the deprecation policy was motivated by the needs of the OVAL Community for a defined process. As OVAL continues to mature, and become more widely adopted, it is important to ensure that the process by which the language may change is well documented and that reasonable conventions are in place for the community to support.

Several key points of the new policy are outlined below:

- All OVAL Language constructs must be deprecated before being removed.
- The duration of deprecation will be in terms of releases.
- Language constructs will remain deprecated for at least one release.
- Deprecated constructs will be marked with machine readable indicators.
- Prior to any release, candidates for deprecation will be announced and discussed
- Prior to any release, deprecated constructs will be discussed on a case by case basis for removal.
- There is no requirement that a deprecated construct be removed from the language after any number of releases.

Following the overview of the deprecation policy, several items were discussed and clarified. These discussion topics are summarized below.

Nominating Language Constructs for Deprecation and Removal

The question of how a given language construct is nominated for deprecation, and how a deprecated item is later nominated for removal, was raised. Under this policy, the OVAL Moderator will work towards a consensus within the community before any construct is deprecated, or removed, from the language. It was clarified that there is no requirement to remove a construct that has been deprecated. However, it is undesirable to have the language become completely bloated with deprecated constructs. A balance between the desire to support deprecated constructs for long periods of time, and the desire to allow the language to evolve free from the burden of carrying year's worth of deprecated constructs, needs to be achieved.

Requiring Vendor Support for Deprecated Constructs

Next, the issue of requiring support for deprecated constructs was raised. Currently, there are numerous tests and other language constructs that have been deprecated. Should the vendors be required to support all of these deprecated constructs?

Historically, deprecated constructs have been considered part of the OVAL Language and therefore considered to be required for a vendor to implement. Products still need to support deprecated constructs to ensure that content, which utilizes deprecated constructs, will be properly supported. Content authors need to be aware of deprecated constructs and avoid using them. Vendors that are implementing support for OVAL must also support deprecated constructs.

With the introduction of machine readable deprecation flags, it is now much easier for users of OVAL to make informed decisions about deprecated constructs.

Deprecations Occurs with New Releases

The question of when a language construct can be deprecated was brought up. Basically, the group wanted to better understand when an item could be deprecated?

A language construct may be deprecated only with a release of the OVAL Language. Constructs will not be deprecated without a new release of the OVAL Language. Similarly, deprecated constructs may only be removed with a new release and they cannot be removed at any other time.

Deprecation Information for Existing Deprecated Items

Given that this deprecation policy was published in April, and that there has not been a release of the OVAL Language since the fall, will this new deprecation information be added to the existing deprecated constructs?

The most significant change of Draft 1 of Version 5.6 was the addition of deprecation information to all of the existing language constructs that have been deprecated.

Schematron Usage in OVAL

Schematron has been used in OVAL since Version 5.0. Compliance with the Schematron rules has been considered optional since its introduction. Schematron is used in conjunction with XML Schema to express data validation constraints that XML Schema alone cannot express. Schematron leverages XPath expressions to define constraints and relationships within the OVAL Language. Schematron can report both warnings and errors during validation.

At this point, Schematron validation can be prohibitively slow. With large XML documents, Schematron validation often takes several minutes. This has been a large factor in considering the Schematron rules to be optional.

Currently, Schematron is optional for all use cases. It is left up to organizations to determine how, or if, they will support Schematron. The OVAL Repository currently requires all submissions to comply with the Schematron rules because it will help ensure that the best content is available in the repository.

This discussion is intended to serve as an opportunity to consider how OVAL should use Schematron moving forward.

Requiring Schematron

The discussion was started by considering the implications of requiring Schematron validation. It was acknowledged that Schematron validation may be more important when producing content than when consuming content. For vendors that consume content, it is probably less important that all imported content is validated against the Schematron rules with the assumption that the content author has already done this validation, and that the tool follows the spirit of the Schematron rules.

It was agreed that moving forward it is important for all published content to comply with the Schematron Rules. This will ensure that high-quality content is available as well as place the burden of content validation largely on the content producer rather than each and every downstream consumer of that content.

Changing Schematron Rules May Break Backwards Compatibility

Currently, Schematron rules are created and modified with each version of the OVAL Language. These changes are typically done to add additional restrictions to the valid structure of OVAL definition documents. This flexibility with Schematron rules helps to ensure that increasingly higher-quality content is produced. However, this maybe concerning to some vendors as it will result in documents that were considered valid in one minor version of the OVAL Language to become invalid in a subsequent minor version of the language.

In general, it was agreed that the ability to add new data validation rules through Schematron, as the OVAL Language matures, is desirable. As long as the changes coincide with OVAL Language releases, the rules should continue to be matured over time.

Validate the Rules or Follow Their Spirit

The discussion shifted to validation and whether or not tools should be required to actually evaluate the Schematron rules, or if the requirement should be that the tools must follow the intent of the rules. Given that there is a strong desire not to force implementations on the vendors, requiring all the vendors to specifically support Schematron may not be the right approach.

The consensus was that it is important to follow the spirit of the Schematron rules. It does not matter if content producers and consumers actually utilize the Schematron rules. The Schematron rules are simply an expression of requirements for OVAL content. A content producer or consumer should be free to use the rules through Schematron or to review them, understand them, and ensure that the rules are followed.

Schematron Validation Requirements

From a product validation standpoint, it is important that a content consumer can report errors in any content that is being imported. However, it is not required for tools to do this all the time, and it is not required that an XML Schema or Schematron rules are used. The intent of the rules must simply be verified.

Conclusion

The consensus was that for OVAL, compliant content should be considered to be any content that complies with the XML Schema and the requirements expressed in Schematron. With this agreement, the discussion was then summarized as follows:

- Official content must be compliant with the XML Schema and Schematron rules.
- Any output from a tool must be compliant with the XML Schema and Schematron rules.
- Any tool must be able to detect noncompliance with the XML Schema and Schematron rules.

Moving forward these items will be captured in the OVAL and SCAP Validation programs. The OVAL documentation will be updated to reflect the requirement that states compliant content must satisfy the XML Schema and Schematron statements.

In order to ensure that the highest quality OVAL content is available, Schematron rules will continue to be allowed to become increasingly stringent with each release of the OVAL Language.

Element Name Reconciliation

In naming and managing OVAL Language constructs, the following guidelines have been used:

- Make element names as intuitive as possible.
- Reduce schema bloat when possible.
- Utilize consistent naming patterns.

As the OVAL Language evolves, these guidelines begin to contradict each other. For example, if a typo is found in an element name that makes the element slightly less intuitive, but, fixing the typo in a new release will add to schema bloat since the misspelled element cannot be removed. As another example, when creating a new test, it may be possible to utilize an existing state. In this case, reusing an existing state will reduce schema bloat, but it will also reduce readability. Due to these inherent contradictions, the naming pattern of a test aligning with object, state, and item names has broken down and element names have diverged.

This discussion focused around a proposal to bring all test, object, state, and item names into alignment. Along with the effort to realign the element names, a convention would be established that all names will align. This convention could be automated to ensure that element names do not diverge again.

Impact of the Proposal

If accepted, this proposal would introduce new tests, objects, states, and items where ever there was a misalignment of elements names. In the process of renaming elements, any incorrectly-named elements would be deprecated. This change would not invalidate any existing content, but it would add schema bloat.

Benefit of the Proposal

If accepted, the proposal would ensure that a constant naming pattern is followed for all future changes. Establishing a constant naming pattern will simplify some implementations since the relationship between an item in the system characteristics schema will always align in name prefix with the corresponding test, object, and state in the OVAL definitions schema. The proposal would also begin the process of removing all misnamed elements by deprecating them according to the deprecation policy.

Proposal Discussion

When considering a change to OVAL, the development burden for making a change should be minimized when possible. People, when given proper documentation, can learn to overcome changes and inconsistencies. There has been concern about the large size of XML documents and thought should be given to making changes to the OVAL Language to reduce instance document bloat.

As the discussion progressed, two different conventions were considered. The first was the proposed convention of ensuring that all element names align regardless of potential schema bloat. The second convention was to lean in favor of reducing schema bloat and reusing elements whenever possible. This second convention would suggest that element names do not need to be maintained to always align, but, when originally created, the names should align. Then, after being created, the names should be allowed to diverge if the result of the divergence will be to reduce schema bloat.

During the course of the discussion, it was pointed out that there is no documented or machine-readable mapping between tests, objects, states, and the corresponding item in the system characteristics schema. It was agreed that this is something that should be documented, and clarified, with the next release to ensure that there is no ambiguity in this relationship. Currently, the mapping is assumed to be based on an element name prefix. Encoding the mapping in element name prefixes may lead to trouble down the road.

Concerns about the way in which OVAL Language schemas use namespaces to differentiate versions were raised. It was suggested that some of the challenges of managing the language over time might be easier to address if the schema namespace included both the major and minor version of the language.

Conclusion

It was concluded that leaving typos in the language is not a good idea and that any of the naming inconsistencies that are due to typos should be addressed.

Keeping the mapping between items, tests, objects, and states as implicitly defined, is not desirable. This is something that should be corrected, and an explicit mapping should be created.

At this time, aligning the names of tests, objects, states, and items is not worth the implementation effort. In the future, perhaps with a major revision, element names should be brought into alignment.

As long as an explicit mapping between tests, objects, states, and items is provided, an effort should be made to reduce schema bloat and reuse existing elements when possible in the future.

Day 2

Choice Structure on Objects

At OVAL Developer Days 2008, the notion of introducing a choice structure into objects in the OVAL Language was discussed and agreed to. Since then, a proposal has been made for how to actually implement this choice structure. What remains to be discussed is whether or not this new structure can be added into OVAL Version 5.6, or if this structure introduces too large of a change for a minor revision.

Issue and Proposal Review

As background for discussion, a review of the issue and the standing implementation proposal was held. That proposal can be found here: http://oval.mitre.org/community/archives.html#nabble-td1485589.

In the course of reviewing the issue, the following example was used:

```
<file_object id="oval:sample:obj:1" version="1" xmlns="…">
    <path>c:\windows</path>
    <filename>foo.exe</filename>
</file_object>
```

Here the current file_object is presented. Currently, a file can only be identified by a combination of the path and filename element as seen above. It is not possible to have a single string representing a complete file path today. This becomes an issue when the complete file path is not available in any other form. On Windows systems, the registry frequently holds complete paths to files that need to be examined. It is also common to store complete paths to binaries in configuration files on other platforms as well. In OVAL, these other sources of information are often queried to determine the location of a file that is then examined with a file test or file permission test. In the current version of OVAL, there is no way to support these subsequent tests because files can only be identified by a separate path and filename.

The proposed solution to this issue would allow the following two methods for representing a file_object in OVAL:

```
<file_object id="oval:sample:obj:1" version="1" xmlns="…">
    <path>c:\windows</path>
    <filename>foo.exe</filename>
</file_object>
```

OR

```
<file_object id="oval:sample:obj:2" version="1" xmlns="…">
    <filepath>c:\windows\foo.exe</filepath>
</file_object>
```

Basically, a file_object could be expressed as it is today with the path and separate filename, or with a combined filepath element.

The XML Schema for a file_object currently looks like this:

```
<xsd:sequence>
    <xsd:element name="behaviors" type="win-def:FileBehaviors" minOccurs="0"/>
    <xsd:element name="path" type="oval-def:EntityObjectStringType"/>
    <xsd:element name="filename" type="oval-def:EntityObjectStringType" nillable="true"/>
</xsd:sequence>
```

The proposed XML Schema would look like this:

```
<xsd:sequence>
    <xsd:element name="behaviors" type="win-def:FileBehaviors" minOccurs="0"/>
    <xsd:choice>
        <xsd:sequence>
            <xsd:element name="path" type="oval-def:EntityObjectStringType"/>
            <xsd:element name="filename" type="oval-def:EntityObjectStringType" nillable="true"/>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:element name="filepath" type="oval-def:EntityObjectStringType"/>
       </xsd:sequence>
    </xsd:choice>
</xsd:sequence>
```

The important change to notice is that an xsd:choice has been introduced. The xsd:choice would allow for two different sequences of elements. Introducing this xsd:choice structure would be a distinct change from the very consistent pattern in OVAL of only having one possible sequence of child elements for any given object or state.

It is also important to note that the example above is based on a file_object and that there are several other places in OVAL that could benefit from this same xsd:choice structure. If it is agreed that this choice structure on objects should be introduced, a similar pattern will be implemented to support choice structures on the other objects too.

As previously stated, the notion of introducing a choice structure was agreed to at OVAL Developer Days 2008. This is now being considered for inclusion in Version 5.6 because it does not break backwards compatibility, and addresses a known deficiency that is preventing certain types of tests from being developed.

Proposal Discussion

Looking at how tests have evolved in OVAL to date, the lack of support for a choice structure has been worked around by simply creating new tests. This can be seen in the user_test and the user_sid_test. Ideally, a choice structure could have been used there to allow the original user_test to support identifying a user by username or by SID. This workaround caused the OVAL Community to learn these two different methods for looking up a user. It would have been cleaner, and resulted in less schema bloat, if a choice structure had been introduced instead of a new test.

In surveying the vendors in attendance, it was agreed that this addition would not be a significant burden to support. In fact, the benefits of adding this choice structure far outweigh the implementation cost of supporting the structure.

Conclusion

A proposal will be created and distributed to the oval-developer-list for adding the proposed choice structure into the current file_object. A list of other objects that would also benefit from this structure will be included in the proposal, and assuming agreement over the oval-developer-list, those changes can also be made in Version 5.6.

Supporting N-Tuples in OVAL

The OVAL Language currently supports several data repositories that may return query results as n-tuples. However, it does not currently provide a mechanism for representing these n-tuples, or for checking the state of a result set that contains n-tuples. OVAL has solid support for result sets with single value results and arrays of single values, but leaves something to be desired when working with WMI, SQL, and XML where n-tuples common.

This discussion focused on reviewing the issue and considering three different proposals for addressing the issue. While the examples and the discussion was in the context of WMI, the understanding was that the issue is common to several data stores that the OVAL Language currently supports and that a common solution based on the proposals will be developed. Given the increasing demand for adding this capability to OVAL Language, the group considered whether or not this feature could be added to the OVAL Language in Version 5.6.

Background

The two examples below highlight the current OVAL capability, and the desired capability. The first example is a WQL used to query WMI that selects the ‘ScreenSaverTimeOut’ field from each ‘Win32_Desktop’ class in WMI.

```
SELECT ScreenSaverTimeOut FROM Win32_Desktop;
```

The second example selects the ‘Name’ and ‘ScreenSaverTimeOut’ fields from each ‘Win32_Desktop’ class in WMI. This example will select 0 – n pairs of results where it is important to maintain the relationship between the ‘Name’ and ‘ScreenSaverTimeOut’ fields.

```
SELECT Name, ScreenSaverTimeOut FROM Win32_Desktop;
```

Using the second example, it is possible to know what a specific user’s ‘ScreenSaverTimeOut’ is set to. With the first example, it is only possible to examine all the ‘ScreenSaverTimeOut’values on the system, and not individual user’s values.

It is also important to note that this capability has not been supported in the OVAL Language because there has never been a solution that did not introduce an entirely new structure for developers to implement and content authors to learn. The group was reminded that a primary goal in the development of Version 5.0 of the OVAL Language was consistency in order to ensure that once a single test in OVAL was understood all other tests would be easily understood. This was also done to reduce the implementation burden for developers.

Current WMI State

As background, the current win-def:wmi_state and win-sc:wmi_item were reviewed before any proposals were presented. Below is an example of a win-def:wmi_state that highlights the result element as it is in Version 5.5.

```
<wmi_state id="oval:sample:ste:1" version="1" xmlns="…">
    <result datatype="string" operation="equals" >user2</result>
</wmi_state>
```

Below is an example of the win-sc:wmi_item as it is in Version 5.5.

```
<wmi_item id="1" xmlns="…">
    <namespace>root\CIMV2</namespace>
    <wql>SELECT Name FROM Win32_Desktop</wql>
    <result>user2</result>
    <result>user1</result>
</wmi_item>
```

Proposal One – 'record' Datatype

The first proposal considered introduces a new 'record' datatype that would allow an entity to have child 'field' elements. The proposed changes to the win-def:wmi_state are shown below:

```
SELECT Name, ScreenSaverTimeOut FROM Win32_Desktop;

<wmi_state id="oval:sample:ste:2" operator="AND" version="1" xmlns="…">
  <result datatype="record" operation="equals" entity_check="all">
    <field name= "Name" datatype="string" operation="equals">user</field>
    <field name= "ScreenSaverTimeOut" datatype="int"
           operation="less than">600</field>
  </result>
</wmi_state>
```

Under this proposal, the current 'result' element would remain unchanged and a new 'record' datatype would be introduced. This datatype would allow mixed content and define a 'field' element. Each 'field' element would have a unique 'name' attribute that would distinguish one 'field' from another. Field elements would have their own datatype and operation to allow for different datatypes and operations to be specified for each 'field' as seen in the example above. Field elements would also support 'var_ref', 'var_check', and 'entity_check' attributes in the same way that other standard entities in the OVAL Language support these attributes.

When considering this proposal there are a few issues that should be weighed:

1. This proposal will keep the result entity closely aligned with other entities.
2. However, close alignment will leave several unneeded and unused attributes on the 'result' element. For example, the 'result' element would allow for a 'var_ref' attribute which would not really have any meaning.
3. The 'datatype' attribute, rather than an element name, is being used to indicate that the element will have child elements. This is different than most of the other constructs in the OVAL Language and is not considered good XML Schema practice.
4. Adding in the 'record' datatype will change the nature of the result element such that it does not align with any of the other entities in the OVAL Language.

Proposal Two – 'resultset' Entity

The second proposal considered introduces a new 'resultset' element that would allow an entity to have child 'field' elements. The proposed changes to the win-def:wmi_state are shown below:

```
SELECT Name, ScreenSaverTimeOut FROM Win32_Desktop;

<wmi_state id="oval:sample:ste:2" operator="AND" version="1" xmlns="…">
  <result datatype="string" operation="equals" >user2</result>
  <resultset entity_check="all">
    <field name= "Name" datatype="string" operation="equals">user2</field>
    <field name= "ScreenSaverTimeOut" datatype="int"
           operation="less than">600</field>
  </resultset>
</wmi_state>
```

Under this proposal, the current result element would remain unchanged and a new 'resultset' element would be introduced. This element defines a child 'field' element and supports the 'entity_check' attribute. The 'entity_check' attribute would allow for assertions to be made about multiple 'resultset' elements. Similar to the first proposal, each 'field' element would have a unique 'name' attribute that would distinguish one 'field' from another. Field elements would have their own datatype and operation to allow for different datatypes and operations to be specified for each 'field' as seen in the example above. Field elements would also support 'var_ref', 'var_check', and 'entity_check' attributes in the same way that other standard entities in the OVAL Language support these attributes.

When considering this proposal there are a few issues that should be weighed:

1. The 'resultset' entity is unlike any other entity in the OVAL Language and diverges from previous efforts to ensure that all entities are similar in nature.
2. The proposal would result in the creation of an oval-def: ResultSetType that would be reused in other logical locations in the OVAL Language.
3. This new element would have only the needed attributes which would simplify it, but also make it even more different than other elements in the OVAL Language.

Proposal Three – Sequential Result Entities

The third proposal considered introduces several new 'result' elements where each new element is sequentially named. The proposed changes to the win-def:wmi_state are shown below:

```
SELECT Name, ScreenSaverTimeOut FROM Win32_Desktop;

<wmi_state id="oval:sample:ste:2" operator="AND" version="1" xmlns="…">
  <result datatype="string" operation="equals" >user2</result>
  <result_1 datatype="string" operation="equals" >user2</result>
  <result_2 datatype="int" operation="equals" >333</result>
</wmi_state>
```

Under this proposal, the current 'result' element would remain unchanged and several new 'result' elements would be introduced. Each of these new 'result' elements would be just like the existing element except that they would allow for tuples with a few more elements to be represented.

When considering this proposal, there are a few issues that should be weighed:

1. This solution will support tuples that are limited in size by the number of sequentially named ‘result’ elements. This will address some cases, but there will always be a request for one more ‘result’ element.
2. This proposal does not allow the complete set of elements in the tuple to be treated as a unit. The other proposals consider the complete tuple to be the unit for comparisons.
3. This is not a complete solution, but merely a workaround that remains consistent with the other structures in the OVAL Language.

Discussion

In the follow-up discussion for proposal two, the suggestion of using the choice structure that was discussed in the previous session was made. A choice structure here would allow a state to have either a traditional ‘result’ element or the new proposed element. This would ensure that this new element would not be used with the existing ‘result’ element.

It was pointed out that the ‘field’ elements from proposal one and two need to also support unnamed fields. For example, a user might want to use ‘SELECT * FROM Some_Table’. This would select all of the fields in that table without naming them. The first two proposals must support this to be effective. The suggestion was made allow the children to be either ‘named_field’ elements or ‘anonymous_field’ elements where the anonymous version would have a sequence attribute to uniquely identify one field and the named version would have a ‘name’ attribute to uniquely identify one field.

In proposal two, it would not make sense to use both the proposed ‘resultset’ element and the existing ‘result’ element in the same state. This might be justification for using a choice structure here or at least using Schematron rules to prevent this.

In proposal one and two, it is important to note that the child ‘field’ elements would all be logically and’ed together during evaluation. So in the examples for proposals one and two, the evaluation results from each field are being and’ed together to determine the overall evaluation result for the entity.

It was also pointed out that when possible, Schematron rules should be avoided. The first preference should be to define language constructs through XML Schema. Then, if all else fails, use Schematron. This will make the language simpler to follow and avoid some of the challenges of Schematron.

These proposals do not address what the corresponding system characteristics items would look like. The proposal assumes that the items will have a parallel structure. When the final proposal is made to the oval-developer-list, an example showing the item in the system characteristics file is needed.

Another option would be to simply overload the existing ‘result’ element by inserting comma separated values. This would allow for some improvement over the existing capability, but would not allow per field comparisons with different datatypes and operations.

RDF may offer some assistance in supporting n-tuples. Perhaps utilizing RDF in this particular structure is worth exploring. The issue is that this seems like a very large change for OVAL. In the near-term, RDF is not likely a good solution, but should be considered for a major revision or as a long-term solution.

This discussion is assuming that all of the data returned is tabular. In the future, OVAL may need to support data that is returned as a graph. In fact, because XML is hierarchical, there may already be a need to support non-tabular data. How would this proposal support querying XML where node sets are returned? At the moment, it is not clear how OVAL would support doing further comparisons on XML data that can be stored in any number of different locations. The solution is not just simple XPath statements to retrieve single values or n-tuples.

This proposal, and the notion of adding RDF to OVAL, led to the suggestion to allow for a 'development' branch, or similar, to run in parallel to the current official branch. This discussion led to a general consensus in the room that an experimental branch is really needed for this and many other reasons.

Given that this need to support n-tuples is really a new area for the OVAL Language to support, it might be better to define this as an entirely new construct and not attempt to fit the solution into the existing entity structure. It may actually be harder for users to learn the new structure if it is not clearly broken out as a new structure. Given the benefit of this capability, it is well worth the cost to teach new users how to use it as a new unique construct.

Regarding the third proposal, we should ensure that we develop a good generally-applicable solution. Simply developing a workaround will not solve the entire problem.

The discussion shifted to whether or not this capability could be introduced in a minor version. One perspective was that as long as backwards compatibility is maintained it is acceptable to introduce capabilities like this in a minor release. To others, this seems like a major release type of capability since there is a fairly high impact on introducing it. It will introduce an entirely new construct to several tests in the OVAL definitions schema and items in the system characteristics schema.

Conclusion

When the discussion concluded, it was agreed that a case could be made for including this capability in OVAL Version 5.6, or deciding that this capability could only be introduced with a major revision. The consensus was that this capability should be further discussed and explored before adding it to Version 5.6. A discussion will be started on the oval-developer-list to continue the dialogue on this topic.

Regarding the merits of the specific proposals, there was strong support for proposal two. Since the structure is really representing an entirely new concept in the OVAL Language, it is at least okay to be inconsistent, and perhaps should be inconsistent, with the other entities. Also, there was no desire to develop a partial solution to this issue.

During the course of the discussion, it was generally agreed to that OVAL does not currently allow the community to easily explore new solutions. There is a strong desire to setup a 'development' branch, or similar, to allow new ideas to be explored and vetted before they are potentially integrated into the official OVAL Language. Most participants agreed that this 'development' branch would be a great place to explore solutions to this issue.

Pattern Match on Enumerations

The ability to use the pattern match operation on enumerations has been an open issue for the OVAL Language since Version 5.0 was released. Enumerations were added to the OVAL Language in order to restrict constructs to specific values, ensure tool interoperability, and increase content readability. As a result, the pattern match operation could not be used on constructs that used enumerations. The deficiency caused by this restriction can be easily demonstrated with the following example.

```
<auditeventpolicy_state id="oval:sample:ste:1" version="1" xmlns="...">
  <account_logon datatype="string" operation="pattern match">
    AUDIT_(SUCCESS|SUCCESS_FAILURE)
  </account_logon>
</auditeventpolicy_state>
```

Currently, this is not a valid auditeventpolicy_state because the string 'AUDIT_(SUCCESS|SUCCESS_FAILURE)' is not included in the enumeration that restricts that allowed values of the account_logon entity. In order to correctly perform this check, two tests would have to be created that check for the value AUDIT_SUCCESS and AUDIT_SUCCESS_FAILURE, and the two tests would have to be embedded in a criteria construct which uses the OR operator. This is much more verbose, and most likely doesn't align with the content developer's thought process.

In Version 5.3 of the OVAL Language, there were Schematron rules that restricted the operations permitted on enumerations to just 'equals' or 'not equal' as the 'pattern match' operation on a finite set of strings did not make sense. Additionally, allowing the 'pattern match' operation undermines the reasoning for having enumerations to begin with. However, in Version 5.4 of the OVAL Language, these rules were accidentally dropped making a workaround possible.

Pattern Match on Enumerations Workaround

The workaround can be implemented by using a variable reference for an entity's value and then specifying the regular expression in that variable. This work around is demonstrated below.

```
<auditeventpolicy_state id="oval:sample:ste:1" version="1" xmlns="...">
  <account_logon datatype="string" operation="pattern match"var_ref="oval:sample:var:1"/>
</auditeventpolicy_state>
```

```
<constant_variable id="oval:sample:var:1" version="1" datatype="string" ...>
  <value>AUDIT_(SUCCESS|SUCCESS_FAILURE)</value>
</constant_variable>
```

In this discussion, the decision to be made is should this workaround be embraced and accepted in the OVAL Language, or should the rules that prohibit this workaround be put back in the OVAL Language for Version 5.6.

If the rules are dropped, and not put back in the language, it would allow the opportunity to close a long outstanding issue. However, this could also be very dangerous because it would allow content developers to place whatever values that they wanted in making it much more difficult to ensure that content is valid, and as a result could cause issues with vendor tools.

Conclusion

After some discussion, a consensus was reached that in Version 5.6 of the OVAL Language the rules should be left out and that the workaround be accepted as part of the language. It was made clear that the documentation should specify that any regular expression used in the workaround should matched against the enumerated values already defined in the OVAL Language. It was also recommended that content developers be encouraged to anchor their regular expressions. Otherwise, unexpected results with vendor tools could occur.

Tests Reference Multiple States

The capability to have a test reference multiple states has been a topic of discussion for many years and has been requested by many members of the OVAL Community. The major motivation behind this capability is that content developers would be given the ability to specify an acceptable range of values as well as produce content that is much more readable. Additionally, during the 2008 OVAL Developer Days Conference (Pg. 21), there were discussions regarding whether or not states should be put back inside tests for Version 6.0 of the OVAL Language. During these discussions, the OVAL Community expressed that a change of this magnitude which would require all of the existing content to be re-written was not in the best interest of the OVAL Community, and should not be pursued any further. However, if members want to reconsider this issue for Version 6.0 of the OVAL Language, the topic can be discussed in more detail.

Proposal

This proposal would grant content developers the ability to reference multiple states in a single test allowing for the use or ranges in a clear and succinct manner. A simple example that can easily demonstrate the value of this capability can be seen below.

```
<min_passwd_len datatype="int" operation="greater than or equal">8</min_passwd_len>


<min_passwd_len datatype="int" operation="less than or equal">16</min_passwd_len>
```

With this new capability, a content developer would be able to make a single test that references one state that evaluates to true if the min_password_len is greater than or equal to 8 and another state that evaluates to true if the min_password_len is less than or equal to 16. This would allow an author to easily write a single test to ensure that the minimum password length was between 8 and 16. Currently, the method of performing this same check would require the author to create two separate tests, one for each allowed state. This is cumbersome, often counter intuitive, and it diminishes the readability of the OVAL content.

Impact of Change

This change would require the addition of a new attribute ‘state_operation’ on the oval-def:TestType as well as changing the multiplicity of each test’s state element to unbounded. An example of this change can be seen below.

```
<xsd:element name="state" type="StateRefType" minOccurs="0" maxOccurs="unbounded"/>
```

The new attribute ‘state_operation’ would be based on the oval-def:OperatorEnumertion datatype which would allow the operations AND, OR, XOR, and ONE to be performed on the states in order to logically combine them.

Lastly, these changes would not invalidate any existing content because it allows content developers to reference multiple states in a single test object instead of just one state object as currently defined. The new attribute ‘state_operation’ would not break backwards compatibility because it would have a default value of AND, and if a content developer is referencing only one state, it would impact the evaluation of the test. It is also important to note that historically test-level attributes have been added to the OVAL Language in minor revisions. However, the same cannot be said for changing the multiplicity of an object.

Benefits of Change

The introduction of this proposal into the OVAL Language would allow ranges of values to be specified for the values of states in OVAL definitions. As a result, it would simplify content authoring because it would remove the extra criteria, tests, and states necessary to perform this same functionality.

Conclusion

As a result of the discussion, it was determined that extending the multiplicity of states to unbounded as well as adding a ‘state_operation’ attribute would be beneficial to the OVAL Language because it would simplify content authoring, increase the readability of OVAL definitions, and allow for ranges of values to be specified in state objects. It was also decided that this change would be acceptable for Version 5.6, and that a proposal would be sent out to the oval-developer-list for further discussion by the OVAL Community.

Introduce PCRE Based Pattern Matches

This topic was originally discussed on the oval-developer-list and at the 2008 OVAL Developer Days Conference (Pgs. 21-23) which was focused on driving Version 6.0 of the OVAL Language. It came to the attention of the OVAL Community that the majority of the existing OVAL content was utilizing the PCRE regular expression syntax even though the POSIX regular expression syntax was defined in the OVAL Language. As a result of these discussions, it was decided that the change from POSIX to PCRE in the

Version 6.0 release of the OVAL Language would be beneficial. However, more discussion was needed to make the final decision.

Proposal

As a result of the discussions mentioned above, a proposal was introduced that would require the addition of a new operation called 'pcre pattern match' in the OperationEnumeration datatype as well as the deprecation of the operation 'pattern match' in the OperationEnumeration datatype. This proposal favors the addition of a new value in the OperationEnumeration datatype rather than adding an additional attribute that specifies the regular expression syntax because it follows how things were previously done with other OVAL Language structures. The major question regarding this proposal was whether or not the change fit in the OVAL Version 5.6 release.

Impact of Change

The major impact of implementing this proposal is that it would introduce the 'pcre pattern match' operation and would deprecate the 'pattern match' operation which was specified to use the POSIX regular expression syntax. Due to the conditions of the OVAL Deprecation Policy, these changes would not invalidate existing OVAL content in the next release. Additionally, this change would suggest that the OVAL Language supports both POSIX and PCRE until the POSIX-based 'pattern match' operation is officially removed from the OVAL Language.

Benefits of Change

The primary benefit of making this change is that it would help ensure a standard meaning for all OVAL content. Currently, content authors are using PCRE syntax instead of POSIX out of habit. Users are accustomed to PCRE syntax and implementers are supporting PCRE syntax. For the most part, POSIX is not being used. Creating a 'pcre pattern match' operation would allow vendors and users of OVAL to declare that a given string is a PCRE-based regex and not a POSIX-based regex. This would ensure that all regular expressions are evaluated in their intended syntax. The change would allow much of the existing content to be corrected and brought into alignment with the regular expression syntax of OVAL.

Major Arguments for the Proposal

The first major argument that arose during the discussion was that it would be very unrealistic for the OVAL tool vendors to implement multiple regular expression syntaxes and it would potentially hinder the adoption of the OVAL Language by new vendors. Another argument for making this change is that most OVAL compatible tools, and OVAL content, are already using the PCRE regular expression syntax. Along the same lines, the majority of tool and content developers are already comfortable with the PCRE regular expression syntax and it is counterproductive to have to train developers to use a different regular expression syntax. Additionally, most regular expression syntaxes, Python, Java, Visual Basic, and Perl to name a few model the same syntactical behavior as PCRE. Lastly, OVAL has made it a priority to promote interoperability and reduce vendor burden in order to further the adoption of the OVAL Language. The addition of many regular expression syntaxes would increase the burden on the developers and reduce the potential for interoperability.

Major Arguments Against the Proposal

The overwhelming argument against the proposal is that by specifying a particular regular expression syntax it would be effectively limiting the capabilities of the OVAL tool and content developers, and would not necessarily solve every vendor's needs. Essentially, in the end, it should be up to the developers to make the decision about which syntax is best for them as this flexibility would help expand the community. Also, instead of choosing a single regular expression syntax, a subset of all of the regular expression syntaxes could be used to ensure that every syntax is compatible. The final major argument against making the switch to the PCRE regular expression syntax is that it does not support internationalization and it would alienate many communities that may be interested in becoming involved in the OVAL Community.

Conclusion

The two most notable options that developed out of this discussion are listed below.

1) Add a new value 'pcre pattern match' to the OperationEnumeration datatype and deprecate the value 'pattern match' from the OperationEnumeration datatype. The value 'pattern match' would then be removed at a later time as specified by the OVAL Language Deprecation Policy.

2) Change the documentation in the OVAL Language to specify the PCRE regular expression syntax as the syntax of choice rather than the POSIX regular expression syntax.

As a result of the discussion, the group came to a consensus that, regardless of the option selected, the change should be made in the Version 5.6 release of the OVAL Language, and that these options would be presented to the OVAL Community on the oval-developer-list for further discussion.

OVAL for System Querying

An emerging use case for the OVAL Language, which has been requested by multiple members of the OVAL Community, is the ability to obtain data from the system without making an assertion about its state. Currently, the OVAL Language provides a framework for performing a system inventory with respect to some pre-defined state whether it be in a compliance, inventory, patch, vulnerability, or miscellaneous definition. This new use case would allow OVAL content authors to request all of the items on a particular system using existing OVAL objects. The major questions considered during this discussion include:

1) Should we support this capability in the OVAL Language?

2) Is there enough support to do the work to implement this capability?

Major Discussion Points

First the issue of whether or not the OVAL Community should embrace the use case for system querying in the OVAL Language was raised. As a result, the Open Checklist Reporting Language, also known as OCRL was mentioned. OCRL is an emerging language specification for collecting a system's state and generating human-readable reports. This then raised the question of whether or not this capability was needed in the OVAL Language, and if so, is a new specification even necessary? The group decided that, before attempting to create an entirely new specification, they would like to see if the capability could be built into the OVAL Language. However, before this use case could be built into OVAL, it was proposed that a model should be built to define what it means to collect the data from the system, and how the data should be represented such that it can be understandable by a human. At that point, it could be determined whether or not this capability is outside the scope of the OVAL Language, and if so, would be best left to a new specification like OCRL. Next, a concern was raised about the privacy and security implications of being able to collect information about a particular system. It was then clarified that system querying would only pertain to authenticated systems meaning that unauthorized users would be unable to collect information about any particular system. Another concern about implementing this capability is that you would now be dealing with large sets of data and it would be advantageous if you could keep a local cache and only retrieve the differences between the cache and the system. It also might be beneficial to have higher level constructs that allow you to filter the data sets. This use case could also be useful in assisting interviewers that need to answer OCIL questions. Lastly, it was mentioned that the OVAL system characteristics file was not the answer to implementing this use case because it cannot convert the system characteristics data into CCEs, CPEs, or CVEs.

Conclusion

It was the general consensus of the group that they liked the idea of having support for system querying in the OVAL Language**.** However, it seems that efforts might be better suited improving OVAL's ability to make assertions about machine state, and that it would be great if the OVAL Community could present some proposals on the oval-developer-list to facilitate additional discussion to decide if it makes sense to put this capability in OVAL (e.g. does it undermine existing goals of the OVAL Language?), or if it would make more sense to add an additional specification such as OCRL.

OVAL Repository Considerations

The key considerations for this topic focus on answering the following two questions:

1) Should inventory definitions be required to have a CPE name?

2) Should compliance definitions be required to have CCE IDs?

These questions suggest that if an inventory definition cannot obtain a CPE name or if a compliance definition cannot obtain a CCE ID their class would be changed to 'miscellaneous'. That is not to say that if an inventory definition or compliance definition are candidates for CPEs and CCEs respectively, but they are unable to immediately obtain these names and identifiers, that they would automatically become members of the 'miscellaneous' class. This notion of putting definitions in a 'miscellaneous' class is reserved for definitions that cannot obtain their respective names or identifiers because they do not actually qualify for them as defined in the corresponding specifications, not because they simply don't have the identifiers or names assigned yet. Two examples of definitions that would go into the 'miscellaneous' class under this proposal can be seen below.

1) A definition which determines if Microsoft Windows XP SP2 or later is installed is not an inventory definition

2) If a product XYZ reaches the end of life it is not a vulnerability definition because it is not a candidate for a CVE ID.

A major discussion point regarding this topic was how would the introduction of a 'miscellaneous' class affect the production and consumption of content in the OVAL Repository. The general consensus was that it would be useful to make the change and clarify the distinction of what can really be an inventory or compliance definition. It was also mentioned that it would not drastically affect the production and consumption of the OVAL Repository content. Another key discussion point was that it would be beneficial to establish a convention on how to handle compliance definitions and formalize what it means to be a compliance definition. Lastly, a question was raised as to whether or not the mapping of definitions belong in OVAL or CPE and CCE where the majority of the work is being done, and whether or not it makes sense to extend CPE and CCE to define where the definitions should exist. The problem with this is that all of these specifications CCE, CVE, CPE, OVAL, etc. have grown independently and it would need to be determined who would be responsible for taking on this task.

Conclusion

By the conclusion of this discussion, there was agreement that it would be beneficial to require inventory definitions to have CPEs and for compliance definitions to have CCEs as it would make the OVAL Repository consistent with vulnerability definitions that are required to have CVEs. As a result, it was decided that this discussion would be continued on the oval-developer-list for further consideration.

Action Items

Deprecation Policy Clarification

In discussing the deprecation policy, and later OVAL's use of Schematron rules, it became clear that the policy needs to be clarified to state that Schematron rules are not subjected to the same policy.

Supporting Multiple Versions in Parallel

A recurring theme in many of the discussions was a strong desire for OVAL to support multiple versions in parallel. The idea is to allow the Version 5.x line to continue to run and be supported, and also support a new major revision that will, after time, allow us to drop support for Version 5.6. This idea needs further consideration.

Defining a Major Version

As the discussion progressed through a number of proposals for possible inclusion in Version 5.6, it became clear that the notion of a major version of the OVAL Language is not adequately defined. It is not entirely clear when a change to the language should be considered a candidate for a minor or major version. The criteria for a major version change in OVAL needs to be defined and documented.

OVAL Specification

There were several requests for an OVAL Language specification to be created. This is something that has been asked for on several occasions over the years.

OVAL Language Versioning Process & Implementation Review

The initial discussion about the OVAL Language Deprecation Policy brought up many questions about the way versioning is implemented in the OVAL Language. The OVAL Language versioning methodology can be found here:

http://oval.mitre.org/language/about/versioning.html

This document describes the versioning methodology including the way in which versions of the language are differentiated. However, the “Differentiating Language Versions” section needs to be expanded.

Once this document is updated, a teleconference should be held to discuss potentially changing the versioning implementation. There continues to be a strong desire to include the major and minor version in the namespace of all language constructs.