OVAL Developer Days

April 28-29, 2008

Table of Contents

Introduction

The third OVAL Developer Days was held on April 28-29, 2008 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 drive the development of a future Version 6. 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 things that were flagged as needing further discussion, or ideas that need to be further worked through based on the discussions held.

It was noted during the Developer Days overview that even though discussion will be about a new major version, it is anticipated that additional minor releases will be needed before the major release can be completed. An estimated roadmap to a Version 6 was outlined and is included below:

June 16th 2006	**- Version 5.0 released**
April 10th 2008	**- Version 5.4 released**
April 28th 2008	**- OVAL Developer Days**
------------------------------	**--**
summer 2008	**- Version 5.5**
fall 2008	**- draft 1 of Version 6.0**
winter 2009	**- Version 5.6**
winter 2009	**- draft 2 of Version 6.0**
spring 2009	**- release candidate of Version 6.0**
fall 2009	**- final Version 6.0 (3 months after rc)**

Attendee List

Assuria Ltd.	-	Chris Wood
DISA	-	Jason Mackanick
	-	Joe Anthony Mazzarella
DOD	-	Vladimir Giszpenc
HP	-	Gyesi Amaniampong
	-	Jeff Cheng
	-	Todd Dolinsky
	-	Nick Hansen
	-	Pai Peng
	-	Alex Quilter
	-	Yuzheng Zhou
McAfee	-	Don Campbell
	-	Eric Fredricksen
	-	Kent Landfield
	-	David Raphael
	-	Ken Simone
	-	Dick Whitehurst
MITRE	-	Steve Boczenowski
	-	Cleo Casipe
	-	David Mann
	-	Matt Wojcik
	-	John Wunder
	-	Margie Zuk
nCircle	-	Jay Graver
	-	Will Weisser
NIST	-	John Banghart
	-	Harold Booth
	-	David Waltermire
Secure Elements	-	Scott Carpenter
	-	Sudhir Gandhe
Sun Microsystems	-	John Totah
ThreatGuard	-	Robert Hollis
Qualys	-	Holger Kruse
OVAL Team	-	Jon Baker
	-	Andrew Buttner
	-	Bryan Worrell

Day One

Better <affected> Element

The first session focused on addressing a number of known issues with the affected element. The conversation started with an explanation of the purpose of the affected element and a detailed description of the current element.

The current affected element is intended to provide descriptive metadata about the products, platforms, and family that an OVAL Definition applies to. The element details the specific system(s) for which a definition has been written and provides hints for tools using OVAL Definitions. The inclusion of a particular platform or product in the affected element does not mean the definition is checking for the existence of that platform or product. The absence of a platform or product element can be thought of as a definition applying to all platforms or products.

A number of deficiencies were cited as the justification for refactoring the affected element. First, the affected element does not currently support standardized platform names. Second, since affected information is considered metadata, it has a tendency to diverge from the actual criteria of the definition. This divergence leads to confusion and increases the cost to maintain a set of definitions.

Next, a set of requirements for a new affected element was proposed. These proposed requirements were intended to capture the positive aspects of the current affected element while addressing its weaknesses. First, affected information should be tied directly to the evaluated criterion of the definition. Selecting sets of applicable definitions for a host without fully evaluating the definitions must also be supported. The affected element must continue to provide metadata suitable for human consumption. The affected element must support CPE to standardize affected platform names.

A proposal was then made to drop the existing affected element and add a new type of criterion. This new type of 'affected' criterion would look similar to the current extend_definition criterion and allow key items in a definition's criteria to be called as conveying affected information. This proposed element would include a CPE reference for expressing product and platform information. The result of this proposal is that all of a definition's affected information would be embedded into the actual criteria of the definition as references to inventory definitions. This proposal would remove family information from a definition's affected element, and in some cases will result in duplicate criterion.

The conversation then focused on looking closely at the proposal. It was quickly pointed out that the CPE reference in the proposed affected criterion is likely to fall out of alignment with the referenced definition just the current affected element falls out of alignment with a definition's criteria.

It was pointed out that the content migration cost to the proposed affected criterion would be quite high. To properly convert the affected element of 5.x into this proposal would require a complete review and retesting of all existing content. This type of change is counter to the organizational desired to reduce content maintenance costs and progress to highly reliable and stable assessment checks. This led to the suggestion to promote the affected information into an applicability section that would codify all of the applicability information for a given definition at the same level as the current criteria. This applicability section should consider leveraging the CPE Language.

It was suggested that the current affected element or something similar to it should remain in the language to facilitate human consumption of content. Keeping this structure will ensure that definition metadata continues to be readable and will also allow an author to easily codify the intent of the definition. This separation may lead to increased content quality since a light and easy to understand structure will be available for authors to express their intent, and this structure could be then be used to run comparisons with the actual evaluated criteria to find discrepancies.

The discussion then shifted to a discussion about embedding CPE dictionary elements into OVAL Definition documents as a way to allow definitions to reference CPE Names without also referencing OVAL Definition ids. If this type of capability is supported then a definition could leverage a CPE Language based structure for determining definition applicability.

The concept of the ‘OVAL Unit’ was introduced during this discussion. An OVAL Unit would sit between the definition element and the existing criteria element. It would include as children an optional platform specification element (defining applicability) followed by the current criteria structure. Each definition would be allowed to have any number of OVAL Units. In practice this would allow smaller platform specification blocks to be bound to smaller criteria blocks. Preselecting definitions would be achieved be iterating through all OVAL Units and evaluating the platform specific portion only.

```
<definition>
        <metadata> …….. </metadata>
        <oval_unit>
                <platform_specifcation ref=CPE>
                <criteria>
        </oval_unit
        <oval_unit>
                <platform_specifcation ref=CPE >
                <criteria>
        </oval_unit
<definition>
```

The discussion shifted to a conversation focused on the three categories of data that need to be encoded; human readable affected information, machine readable prerequisites or applicability checks, and machine readable evaluation criterion.

It was agreed that the suggestions discussed need to be further considered. It was agreed that the affected element should be refactored and that we need a more formal representation of applicability for an OVAL Definition. Follow up conversations on restructuring the affected element will be held on the OVAL Developer list.

Definitions as the Focal Point

The focal point of the OVAL Language is currently the Definition. The expectation is that this is the unit that external languages (i.e., XCCDF, etc) reference. It has been suggested that we treat all OVAL units (Definition, Tests, Object, States, Variables) the same and allow external references into each. Is this a good idea? Related to this in some way, metadata is associated only with the definition. Should we expand the metadata and associate it with each unit?

Some background information was present as to why the definition has always been the focal point. OVAL started out as an SQL language and back in those days there was only the concept of a single definition (or query for those purists). With the transition to XML, OVAL started to separate things like tests and objects in order to foster reuse. But there was a fear that this breakout might result in confusion if some users were focused on definitions and others focused on tests (or objects, etc.). In an attempt to keep everyone on the same page, the definition was advertised as the external entry point to OVAL. It was the thing that everyone should link to and reference (even if it only contained a single test).

To help encourage this, most of the metadata was only placed on the definition and was not repeated on the other units.

Recently there have been some questions about external entities being able to reference tests, without the overhead of a definition. XCCDF has in the past desired this type of functionality since that content is often just concerned with checking a single registry key or a single file. The needs of XCCDF align very nicely with the test structure. It was pointed out however that there have been many instances where the needs of an XCCDF check have changed over time and the original definition (that might have just included a single test) has been modified and now contains multiple tests. This type of change would have been difficult if XCCDF did not reference the definition structure.

Another point to consider was brought up related to content management and sharing across multiple external repositories. The definition structure gives the community a place to put things like signatures and tracking information.

A second use case related to referencing something other than a definition is the proposed remediation language. Leveraging the OVAL Object structure might be very useful (as opposed to re-inventing the object structure). Currently, the OVAL Language does not promote this type of interaction. Although there is nothing at a technical level stopping this type of interaction.

One option brought to the table is to start by putting some effort into sharing the object structure since that seems to be where the most requests are. If this worked out and requests for additional sharing are made we could always open up more of the language at that time.

In conclusion, the feeling in the room was that it is perfectly acceptable for external schemas to include parts of the OVAL Language (test, objects, etc), but doing so would be done outside of OVAL. The OVAL Community should stay focused on their main mission. There really isn't anything that has to be changed in the OVAL Language to support these external uses. The change would simply be to acknowledge these others usages as valid uses of the constructs defined in the language.

The decision was to continue under on our current path, but understand that these types of uses are out there and try to better understand their needs and desires while at the same time thinking about how this impacts the rest of OVAL.

Reusing Content Across External Repositories

OVAL has always strived to facilitate reuse. Version 5 of OVAL introduced many opportunities for reuse of components within an OVAL Definition document. This reuse made it easier to write new definitions, designed numerous opportunities for optimization into the language, and simplified content maintenance.

Version 5 introduced the notion of an extended definition and utilized this concept to allow existing definitions to be easily built upon when authoring new definitions. As discussed earlier, definitions are intended to be the focal point of reuse across repositories. In Version 5 this type of reuse is achieved by copying the definition, and all of its dependencies, from one repository into the new OVAL Definition document that contains a new definition intended to extend the existing definition. For example, the only way to reuse a CPE inventory definition from repository A in a vulnerability definition from repository B is to copy the inventory definition from repository A to repository B. This requirement to copy definitions results in a content maintenance nightmare. There is no simple way to keep copied definitions up to date with the original definition in the source repository.

This discussion focused on addressing these deficiencies by enabling definitions to reference other definitions without requiring the referencing definition to include a copy of the referenced definition in the same OVAL Definition document. A number of challenges must be overcome to allow this sort of referencing.

The first challenge discussed was that a definition that references another definition is subjected to all of the lifecycle issues of the referenced definition. So if the referenced definition changes the referencing definition may change in meaning as well. In some cases this might be a good thing and improve the quality of the referencing definition. In other cases the referencing definition may no longer have the desired meaning. This is first issue is really related to the fact that for referencing definitions to work you must be able to trust the maintaining authority of the definition. A definition repository's moderator must not allow the meaning of a definition to change. Unfortunately, there is no way to enforce this type of restriction.

Another challenge related to referencing definitions is that it will not always be the case that referenced definitions are available. In situations where network connectivity is down or not available, it will not be possible to evaluate referenced definitions.

Referencing definitions may also not be appropriate in cases where a set of OVAL Definitions must be certified. These challenges lead to the conclusion that we must support the current extended definition capability in addition to any new referencing capability.

The discussion shifted to defining an API or web service interface that all definition repositories should support. Traditionally this has fallen under the auspices of the OVAL Compatibility program. In some cases it might be prohibitive to require such capabilities of a repository moderator.

It was pointed out that in the current environment we are basically requiring everyone that wants to host a repository of OVAL Definitions to write the same application. It would be a huge benefit to the community if an open source repository was made available for all to leverage.

Currently most of the vendor applications are shipped with their own copies of the content. The current applications do not refer out to repositories. They tend to use only trusted content validated by the vendor before it is shipped to a customer and evaluated in a customer environment. This line of discussion lead into a conversation focused on developing solutions and capabilities that would enable efficient content sharing.

It was pointed out that for a content referencing capability to succeed we would need a solution external to an OVAL Definition document that will indicate where to look for a source of definitions. This is not something that can be defined within the language. The language can only add support for a referencing capability.

Another drawback to supporting a referencing capability is the potential impact on the network to support the number of requests required to serve all the referenced OVAL Definitions. A typical set of vulnerability definitions might reference several hundred inventory definitions. Each of these definitions would result in a request to some repository. This type of activity could quickly result in a denial of service issues.

After continued discussion about desirable requirements and capabilities for managing and maintaining OVAL content, it was agreed that developing a definition referencing capability should be further evaluated. Several key items should be considered for a first proposal to address a referencing capability.

First, there must be a way to show the intent of a reference and show result of the reference in the results of an evaluation. This capability will allow authors to indicate whether the current version of a definition can be used or a specific known version must be used.

Second, any sort of referencing capability will need to be considered when the OVAL Capability program is updated for the new version of the language.

Finally, this referencing capability will be new and experimental for many content authors. A new referencing capability should support an xsd:any or similar construct to allow community exploration of this new capability.

Supporting Network Devices

A number of community members have been asking for better support of network devices. This section worked though the current schemas looking for ways to improve on what is being supported, as well as explored possible additions that could enhance the ability to use OVAL for routers, switches, etc.

The discussion was focused around a proposal originally sent out by AlterPoint. The basis of the proposal was to add support for new network devices and to think about changes that would make it easier to add new network device component schemas in the future.

Network Device Defined

The first question asked was, “What is the definition of a network device?” It is hard to answer this question since these devices are constantly changing in their functions and names. For example, multiple hardware modules are being collapsed into a single unit, and often times combined with things that might not have been considered network devices before.

One attempt to define a network device was to say that it is something upon which a user is not allowed to install code. Related to this would be that the device can only be accessed via the network. If the user does have access to the device, or if the user can install code, then that device should be considered a server. It was pointed out that there are a number of newer devices (known as network devices) that run full versions of Linux and even though the vendor says you can’t install something else on the device, with a quick change to a configuration setting the device can be opened up.

Another possible definition was that a network device is a device that you dump the configuration to a flat file in order to read. Unfortunately there are a number of devices out there that dump their configurations to binary files or some proprietary object format.

Maybe we should really call what we are talking about “Network Control Devices”, or maybe “Hardware-Based Network Devices”.

In reality, everyone can come up with their own definition for what a network device is. What is important for OVAL is that we have available all the necessary tests. It is not important that we classify things correctly. If a network device is running windows, then we can use the existing registry test, we don’t need a separate schema just because we think of this as a network device. So it was mentioned that we should just focus on devices that aren’t covered by existing OVAL Schema and how to best provide support for them.

Network Device Schema

To help support network devices in general, and to make it easier to add additional component schemas in the future, one proposal is to create a set of general tests that relate to the entire class of (what we consider) network devices. Possible tests might include a device_type_test and a config_file_test. Once this high level network device schema has been created, then individual component schemas can be created to hold tests specific to a certain device.

One additional point was to remember that we must design these schemas thinking about different types of implementations. If we design a test around a certain API and it requires an implementation to log into the box via SSH to call this API, then we are really limiting the types of possible implementation. When at all possible, we need to create these schemas in a way that is implementation agnostic.

It might be also worth consideration to have schemas based on function as those types of devices might share similar tests. For example, maybe a firewall schema or a router schema would be worth adding. This would need some more exploration as no one was sure how this would work. It is possible that this idea is more focused around the need for a test like the family_test, where we could ask if the device is a router, firewall, etc.

The group in attendance was in basic agreement that this is a good place for OVAL to expand into, and further discussion should be had to determine how best to set up the different component schemas and what types of tests should be in them. But breaking out a common network device schema might not be all that useful. There just isn't enough commonality across devices.

Repository and Reference Implementation Transition

The OVAL Repository and Reference Implementation (the OVAL Interpreter) have both undergone major version changes in the past. This discussion hoped to address the concerns of the vendors with respect to the OVAL Language version update from 5.x to 6.0.

Planning for the migration to OVAL 5.0 began in May of 2005. Drafts began to appear in October of 2005 and continued until the first of the release candidates appeared in March of 2006. The release candidate phase continued until June of 2006 when the official OVAL version 5.0 was released on Jun 16th, 2006.

When planning out the migration from version 4.x to 5.0 it was determined that the 4.x content would be archived on the OVAL Repository website and then be updated to 5.0. Content producers were given a month after the release of version 5.0 to begin submitting 5.0 compliant content—during this time MITRE would handle the conversion from 4.x content to 5.0 content.

OVAL Repository Transition

The group was reminded that the purpose of the OVAL Repository is to facilitate community interaction with regards to the OVAL Language. Wide-spread adoption of the standard is reliant on the usage and understanding of the OVAL Language within the community and because of this it is felt that the OVAL Repository is really the "community's repository": MITRE-alone cannot drive the adoption of the standard.

The proposal for the migration of version 5.x to version 6.0 follows a similar roll-out timeline seen during the migration of version 4.x to 5.0. Drafts would likely start to appear during the fall of 2008 followed by a release candidate phase. When the 6.0 launch occurs, the 5.x content would be archived on the OVAL Repository website for community members to access. During the migration the likelihood of several vendors having to update large amounts of content is extremely probable. Because of this we would like to consolidate our efforts by working together on a tool which will assist in the conversion of OVAL content. This may be a stylesheeting exercise but could end up being a fully-function application that requires sophisticated logic. This is an opportunity for the community to work together to determine the best route to go and reduce the cost to convert to version 6.0.

It was pointed out that quality assurance is a concern when undergoing a major version change: there are still semantic errors lingering in the OVAL Repository that were caused by the migration from version 4 to version 5.

The growth that has occurred within the OVAL user-base over the years was brought up. When undergoing the update from version 4 to version 5, the number of tools, repositories, and users, as well as the amount of content was all very small. Because OVAL as a whole was small the past transitions were much easier. Whatever is done in terms of changes to the structure of the OVAL Language should not affect vendors too drastically because vendors will be able to cover themselves; however, the idea of a hard cut-off from version 5 is not realistic.

It was proposed that instead of a hard cut-off date being set early on in the development process, a date should only be established after the drafts have come to a point where there will be no more major changes. Once there are no more major changes scheduled and the conversion tool is developed, the existing content in the OVAL Repository can be converted over and then archived.

It was explained that in the past sample files, along with the drafts and release candidates of the language, were provided to the community allowing vendors something to test their tools against. This enabled vendors to adapt their tools to consume the new features and it opened up a new channel for feedback.

Because OVAL is a Language, it was proposed that version updates should be handled much in the same way that updates to programming languages (such as Java) are. With the exception to Java 1.1 becoming Java 1.2, the programming language attempted to be as backwards compatible as it possibly could through the use of deprecating methods. This allowed developers to use existing applications and an existing code base without having to radically modify either. Instead of breaking applications or existing code, the compiler or consuming application would print out a warning letting the developer know that there is a newer way of accomplishing the same thing. Because of examples like Java, it is felt that OVAL should consider the same behavior.

Another point was brought up regarding the resources to support two different versions of the language and thought was given to what it would take to be able to accomplish this. The question was scoped by stating that whatever is done with the OVAL Repository isn't necessarily what others are going to want to do with their external repositories. This being said, others could state that they want to support two different versions of the language.

In terms of backwards compatibility, it was felt that it isn't just the OVAL Interpreter that should be able to digest older content. Rather the OVAL Schema should also actively support different versions of OVAL content through the use of Schematron, XML Schema, or some sort of XSLT exercise. This means that content authored in OVAL version 5.x should be capable of validation against the OVAL version 6.0 XML Schema.

The requirement to always support backward compatibility across major versions of the OVAL Language may prevent the language from making significant changes in structure, no matter how beneficial they may be. To reiterate, this plays into the definition of a major change versus a minor change: a major change may break existing content. The suggestion being made is really to never again have a major version.

The notion of not upsetting the customers through the use of an immature and unstable content medium was reiterated several times. Because of the large consumer-base, it is felt that there needs to be significantly more planning in this version change relative to previous version changes.

The ability to support multiple versions of the OVAL Language within the OVAL Repository is heavily dependent on the changes that go into the new version. The greater the differences are between the two versions, the more resources that will be required to support the two versions. If the language is wholly backwards compatible, the issue of supporting multiple versions becomes null and void. The content in the OVAL Repository would never need to be converted.

The question was raised, "Why change features in the language if it isn't broken?" It was felt that getting something to just work shouldn't be the goal of OVAL, and that getting it to work well, or better than it does now, should be its goal. Sometimes the improvement of language features will mean the invalidation of previously authored content. This statement could be illustrated through the example of Perl's migration from version 4 to version 5. When version 5 came to be, a lot of functionality found within Perl 4 code was broken. This meant that a developer's version 4 code base could function with some modifications, but any sophisticated code was broken and had to be upgraded to version 5. This being said, can the benefits of new features or restructured areas of the language justify the invalidation of existing code, and if so how should the community get the damage under control?

It was brought up that many enterprise-level application developers rejected Perl 5 due to the lack of stability within the language so a question was raised regarding the importance of a language's stability over its feature set (capability): which is more important? The answer seems to fall in line with stability and consistency through the OVAL Language such that a schema document could validate multiple OVAL versions. This being said, producing a schema document that validates both OVAL version 5 and OVAL version 6 content is possible, but the schema will likely become less readable and more difficult to maintain.

Having to throw out content and throw out software and rebuild whenever a new version of the language arrives hinders the adoption and growth of the language. This point demonstrates that any major change that is made needs to be well gauged.

Reiterating the notion that the version update process needs to be well thought out, well tested, and aptly timed, an important question was raised: Just how much time is needed? Unfortunately, this is open to too many variables and cannot be gauged at this point.

While going through the development phase, a feature freeze is necessary to allow significant testing by vendors. If the content is constantly changing while vendors are attempting to develop support for OVAL vendors will be more resistant to adopt the standard.

While it is possible that the next version of the OVAL Language ends up being only a minor release, it is felt that the changes being proposed during Developer Days would require a major version change. The ideas derived from this discussion will help to determine if and how these changes will occur and be incorporated.

OVAL Interpreter Transition

Since version 5.3 build 40 of the OVAL Interpreter, it has been hosted on SourceForge.net. Hosting it on SourceForge.net has given MITRE the ability to take advantage of their monitoring, bug tracking, and repository management capabilities. Since then we have been able to take collaborating to a new level with source code being sent in to us and included in the OVAL Interpreter.

It was proposed that when 6.0 is in development the OVAL Interpreter will branch off (or tag) the current state of version 5 and continue to develop 6.0 off of the trunk. While developing 6.0 the interpreter will see a feature freeze but will still keep up to date on 5.x releases as well as bug fixes. The OVAL Interpreter would then be released with the OVAL Language once it arrived at version 6.0.

OVAL Objects

This session of OVAL Developer Days focused on the OVAL Object structure and ways that it could be improved to make the language more powerful and flexible. Two proposed changes were discussed as outlined below.

Stand-Alone Objects

In version 5 of the OVAL Language, objects were split out from the test structure and allowed to be reused across multiple tests. The way this was done was to have a new object type for each test type. Two problems have surfaced with the current model. The first is that certain objects end up having similar constructs (e.g. file paths) and if changes to these constructs are to be made, they must be made in each object that uses the construct. The other issue is that behaviors that are related to a given construct must also be repeated for each object that uses the given construct.

To solve the two problems above, a proposal was made to allow objects to exist independent of the test structures. Currently there is a one to one mapping that is enforced between tests and objects. Although in reality this would probably still be the case given this proposal, the idea would be not make this mandatory. So if two tests use the same object (e.g file attributes and file permissions) then we don't have to repeat the object definition in the schema.

Discussing this, we realized that in reality, most of the current objects are unique in some way. For example the textfilecontents_objects uses the same constructs as the file_object, but adds a pattern entity to define a specific block of text in the file. Getting away from the one to one mapping probably would not be something we can achieve.

Instead, objects should be allowed to extend other objects. For example, the textfilecontents_test could extend the existing file object (pulling in the path and filename entities) and add an entity to specify the block of text to test. By using the properties of extension, the behaviors associated with the file object could also be carried over and used with the new object.

One big advantage of the extension approach is that these changes to the schema might not result in any change to an XML instance document. The textfilecontents_object would still be written the same way with the same entities (meaning the instance would still validate under version 5 schemas). The only change would be how that object is defined in the schema files.

Further discussion brought out the idea that if we can add more integrated object reuse, tools might be able to adjust their implementations to realize gains in efficiency. For example, if a file_object is defined and then used by both an attribute test and a text file contents test, then an implementation could possibly perform both operations at the same time (only pulling the file across the network once), thus saving steps in the processing of the test.

By the end of the discussion the group had narrowed done the proposed change to a decision of either composition or extension. In composition, an object embeds existing objects (via a reference to them) in order to leverage that existing object's structure. (e.g. a textfilecontents_object would refer to an existing file_object and a textpattern_object). In extension, an object would be derived from other objects and then would add additional entities. Both philosophies have advantages and disadvantages over the other. One big difference is that with extension, a tool would need to have knowledge of the schema to determine relationships, while with composition the relationships are embedded in the content. Of course the composition approach (with its additional references) might make the content more difficult to understand and write.

One final note was brought up ... the advantages of composition would have to outweigh the cost of changing existing content, since the extension approach would probably not result in a change to existing content.

Choice Structure

For certain objects, there is a need to have different ways of identifying them. For example, Windows accounts need to be identified by both name and SID depending on the use. Another example is with the path and filename entities that sometime need to be referenced as a single entity. The current OVAL Object structure does not allow this. This discussion centered on the idea of introducing a choice structure into the schema definition of the object to allow these different approaches to instantiating the object.

It was asked how a tool would deal with this and which definition a tool would use. The first point to understand is that in order for this approach to work, each choice would have to be a unique way of identifying the object. In addition a tool would have to understand how to search for an object based on all defined approaches.

Windows SIDs

The discussion turned into a related discussion about the choice of Windows SIDs vs. Names. OVAL has struggled over this choice in the past and currently has different tests for each approach.

A different approach to this instead of offering a choice is to treat the SID/Name issue like a function and have it resolved before the object is evaluated. In other words, the schema might define the object to use a SID, but if a user supplied a name then that name would just be resolved before passing the object to the interpreter. This was seen as a bit unnatural in relation to the way OVAL approaches everything else.

Another option would be to have a behavior that resolves built-in trustee names to their actual account. The benefit here is that a user could refer to the account by its common name (e.g Administrator) and OVAL will know to look for the built-in account (by its well know SID) even if the name has been changed. Of course with the choice structure in place, a user could just use the well known SID in the first place instead of using the trustee name. This behavior is something that could be added to a minor release and should be something considered for Version 5.5.

If the desire is to allow the content writer to refer to an account by the name but underneath use the well known SID, then maybe a 3rd choice option (in addition to SID and Name) is needed for Well Known Sid. With this, an object would be written using the common name of the well known SID, but any tool would link this to the SID in order to avoid any pitfall due to a name change. This of course requires tools to hard code the mapping between Well Known Sid and Name.

File Path

The other example discussed was related to file paths. Currently the file_object is defined using a path entity and a filename entity, but often this information is supplied as one combined path. By utilizing the choice structure, both options could be allowed. The room was pretty much all in agreement about this being a good thing for inclusion in OVAL.

Day Two

Agility

New features are regularly being proposed for inclusion within the OVAL Language. This section hoped to address methodologies for helping the language respond to feature additions while addressing the possibility of schema changes as well as how to mitigate the inclusion of schema changes required by new content.

Several initial suggestions pointed towards a modularized solution—keeping point releases separated from the stable schema releases is an example of this. It was stated that keeping point releases separate from the stable schema documents might warrant insufficient testing of new features. Rather than separating the new features by file, it was proposed that a "sandbox" working area could be constructed through the use of namespaces.

Keeping new features within a new, experimental namespace provided vendors with the capability of recognizing new features, allowing them to respond to these features as they saw fit. While this meant that experimental data could remain in the OVAL Repository as production-grade data, it was felt that this would lead to fragmentation within the OVAL Repository and the exposed feature-set of the language.

A new feature would need to be properly evaluated before it was declared stable and pushed out of the experimental namespace. A concern was brought up that suggested it was bad practice to force features to remain in an experimental phase for an amount of time prior to including it in the stable schema. It was stated that if a new vulnerability was found in the field and the OVAL Definition required an adjustment to the schema, that data could not be used by vendors for the experimental phase time period. However, it was stated that as long as it was flagged as experimental through the use of the namespace tools could use the data or ignore it. It is architectural decisions made by vendors which allow them to compete with one another—if a vendor can digest experimental features well and respond to new data and feature sets quickly and elegantly it will cause others to follow.

Allowing the use of "script tests" could, in theory, allow flexible creation of OVAL Definitions but in practice has raised security concerns. Even though it was stated that script tests could increase the speed in which OVAL Definitions were authored, it was felt that this was a bad method for authoring content and opened the door to security problems. However, this mentality proved hypocritical when it was mentioned that allowing SQL queries could cause security issues—many database management systems allow queries to call out to executable code found on the host operating systems file system. Also, SQL contains certain functions such as "DROP" or "SELECT INTO" which could be used in a malicious manner. Because of the concerns surrounding SQL query execution it was felt that annotations should be added to the schema indicating what level of access the executing user should have or other policies that should be in place prior to execution of the OVAL consuming application. Such an annotation could indicate that the mounted file system be read-only. By forcing the file system in which the database lies to be mounted as read-only or forcing the queries to be executed by a user with read-only access during execution of the OVAL consuming application we eliminate the issue of having data altered maliciously.

During the last few minutes of the session the notion of vendors creating OVAL content was reiterated. The community is here to help the vendors—the vendors need only participate or place a “stamp of approval” on content authored by another party.

Future of OVAL Compatibility

This discussion focused on outlining the current OVAL Compatibility program and then discussing the plan to transition testing of compatible products to NIST and the National Voluntary Laboratory Accreditation Program (NVLAP).

The current OVAL Compatibility program was developed to establish consistency in the usage of the OVAL Language. The program outlines a set of guidelines that help enforce a standard implementation of the language and allows users to easily distinguish between compatible products and know that their implementation coincides with the standard.

At a high level the compatibility program follows a five step process. The first step begins with an organization seeking compatibility sending and email to oval@mitre.org and requesting a declaration form. The second step is simply the submission of the completed declaration form. Once this form is reviewed the organization’s declaration to support the standard is posted on the OVAL web site. This posting includes organizational information, the product that intends to become compatible, how the product will use OVAL, and the status of the capability. The third step is the submission of a completed compatibility questionnaire. This questionnaire requires that the organization state specific and verifiable details about how it has satisfied the compatibility requirements. At this time, MITRE will review the responses to the questionnaire and notify the organization of any potential areas of concern. Once both MITRE and the organization are satisfied with the questionnaire, MITRE will post the questionnaire on the OVAL Web site. The publication of the organization's questionnaire on the OVAL Web site allows end users and prospective customers to compare how different products satisfy the requirements and decide which are best. The fourth step in the process is correctness testing. This step usually involves bringing the product into a MITRE run lab and validating that the product has in fact followed the intent of the OVAL Compatibility program for the specific type of declared capability. The final step in the compatibility process is to grant compatibility to products the successfully complete the correctness testing phase.

After the overview of the current compatibility program, MITRE’s compatibility transition plans were presented. MITRE has been looking for an organization to take over the correctness testing portion of the compatibility program for years.

NIST established SCAP Validation program to test and certify that tools are correctly using SCAP. A portion of the program includes testing some aspects of OVAL compatibility. This overlap has led NIST to request to run an OVAL Validation program as well. MTRE thinks this is a good idea as it will allow MITRE to focus on the advancement of the standard while NIST leverages NVLAP to deliver a high quality validation program for OVAL.

NIST is working on a first draft of OVAL Validation testing requirements. These requirements will be based on the revised OVAL Compatibility Requirements document that the community developed in the fall and winter of 2006 (See the oval-developer-list archives for a copy of the draft). MITRE's current intent is to defer the testing portion of the compatibility program to NIST when they are ready. MITRE will look to the OVAL Board to make the determination as to when to transition to the NIST OVAL Validation program for testing.

As a reminder the draft of the compatibly program reorganizes the categories of compatibility. The current program focuses on the notion of consumers and producers of content written against each of the three major components of the OVAL Language; OVAL Definitions Schema, OVAL System Characteristics Schema, and the OVAL Results Schema. The new program will define 5 main roles that more closely align with how the standard is used today. The Definition Evaluator role is filled by a product that uses an OVAL Definition to guide evaluation and produces OVAL Results (full results) as output. The System Characteristics Producer role is filled by a product that generates a valid OVAL System Characteristics file based on the details of a system. The Definition Repository role is filled by a repository of OVAL Definitions made available to the community (free or pay). The Authoring Tool role is filled by a product that aids in the process of creating new OVAL files (including products that consolidate existing definitions into a single file). Finally, the Results Consumer role is filed by a product that accepts OVAL Results as input and either displays those results to the user, or uses the results to perform some action. (remediation, sim, etc.)

The proposed OVAL Compatibility process will roughly follow the existing process with the major change being that the fourth step in the process, "Validation Testing", will be run by NIST through the NVLAP program.

At this point in the discussion NIST described the SCAP Validation program and how NIST plans to run the validation program. It was clarified that NIST does not actually do any testing. NIST accredits laps through the NVLAP program to do the testing. (a detailed description of this process is available in the oval-developer-list archives) The desire to ensure that this transition is an open collaborative process was stressed. NIST will be actively seeking review and comments on the drafts of OVAL Validation requirements. (the first draft is available in the oval-developer-list archives)

A question about validation and how it progresses with versions of the language was asked. Currently products are required to revalidate with each major version of the language. NIST is looking into how validation will follow the versions of the standard. It was pointed out that under the proposed NIST validation program multiple version of the language could be supported by the program and the current thinking is to require tools to revalidate annually regardless of any changes to the standard.

A concern was raised at this point about a required annual revalidation when the underlying standard my not have changed. One of the reasons for this annual revalidation was to allow the program to mature. It was acknowledged that this may not be appropriate for OVAL and that revalidating with minor version changes to OVAL may also not be appropriate.

Another positive aspect of the transition to NIST run oval validation is that a product may be simultaneously tested for SCAP, OVAL, and other standards. Basically, it should be possible to validate bundles of standards all once time. The bundling will be left up to negotiations between vendors seeking testing and the lab doing the testing.

When talking about compatibility we need to make sure that we are defining testing and validation procedures that are widely recognized by a broad audience, as OVAL adoption increases this will be increasingly important. NVLAP and the NIST validation program have a far more broadly recognized name that will help with this need.

A concern was raised over the change from a free program to a fee based validation program. This change may result in fewer compatible products. This concern was not really addressed because the conversation shifted to a discussion of validating repositories of content.

Repository compatibility is a difficult challenge. Traditionally the notion of repository validation has focused on making sure the repository can reliably produce valid content written in the OVAL Language. There may be a need for semantic validation of repositories. Due to the difficulty of semantic validation of a repository, and the implied meaning of compatibility, it may be worth looking into developing a program that is less weighty than compatibility to convey the simple syntactic validation that is currently in place.

The closing point of the conversation was that both NIST and MITRE want any compatibility transition to be very open and both organizations will be relying on the community for support and guidance throughout the transition.

Regular Expression Syntax

The question of how to represent regular expressions within the OVAL Language has been an ongoing topic on the OVAL mailing list. The question of whether or not POSIX 1003.2 was the correct choice in regular expression standards has been debated throughout the community ever since the OVAL Language added regular expression support. This discussion hoped to describe the benefits and drawbacks of the different regular expression standards proposed on the OVAL mailing list.

When regular expression support was included in the OVAL Language it was decided that POSIX 1003.2 was the most appropriate standard to use. This was because it had a limited feature-set, yet it had regular expression constructs needed by OVAL, and also because POSIX is a widely recognized body of standards found mostly within the realm of UNIX.

POSIX regular expression syntax, as it turns out, is not widely used outside a smattering of tools within the world of UNIX as well as a few Windows applications. The absence of development libraries and full support within common development framework regular expression engines made the use of POSIX a challenge. Because it is difficult to implement programmatically, and because certain syntax constructs are unfamiliar to the majority of people who use Perl-esque regular expression syntax, and because the POSIX does not support Unicode, the question was raised: Should we change the regular expression standard used within OVAL?

With many regular expression flavors in the wild such as Perl, Python, Ruby, XML Schema, XPath 2.0, etc., deciding on the one true standard to be represented within OVAL has proven challenging. This notion is only reiterated when the feature sets of each of these commonly used regular expression standards are not symmetrical. This can be illustrated by looking at any regular expression that uses anchors to mark the beginning or end of the lines: these anchors hold no special meaning to XML Schema regular expressions. With all of these standard regular expression flavors being used in the wild, a discussion was held about the set of requirements for OVAL: hopefully to help determine what standard (or what subset of an existing standard) was the best fit for the OVAL Language.

The requirements for a regular expression standard proposed were as follows: it needs to have a small set of features, it needs Unicode support, it needs programming language independence, and it would be nice if it were an established standard.

Various regular expression standards have been proposed on the OVAL mailing list in the past: Unicode (henceforth known as UTS), PCRE, and XPath 2.0 (PCRE with the most support). Each of these, with the exception of UTS, has their own unique strengths and weaknesses.

Despite UTS being described as a viable option for the OVAL Language to adopt, UTS is not actually a regular expression standard. As it turns out, UTS is a technical standard which describes how a regular expression engine should consume and interpret Unicode as it parses the regular expression. Because it is not an actual flavor of regular expressions, it was thrown out as a viable solution.

Perl Compatible Regular Expressions (henceforth known as PCRE) seemed to have won over the majority of those on the OVAL mailing list. The fact that Perl regular expressions are used heavily in the programming and systems administration world makes it an attractive option. While Perl regular expressions are a part of the Perl programming language, PCRE has allowed the use of Perl regular expressions in C and C++. PCRE does not fully support Perl regular expressions as there exists a small set of Unicode character classes that PCRE dislikes and thereby ignores.

It was noted at this point that Java has an option to support Perl 5 regular expressions wholly and that Ruby has wrappers for PCRE. On a side-note, the documentation that supports the claim of full Perl 5 support in Java has proven elusive at best. On the contrary, the Pattern class Javadoc explicitly denies full Perl 5 support, stating that there are Perl 5 constructs not supported by Java and vice versa.

The size of the Perl 5 regular expression standard makes it less than appealing—a standard representing a subset of Perl 5 regular expressions would be nice, if not ideal. Because of the requirement to have a small set of features, the notion of XPath 2.0 seemed appealing.

The XPath 2.0 regular expression standard is built on top of small regular expression standard used by XML Schema. XML Schema is tiny when compared to the functionality found within Perl and, in fact, is too small and lacks functionality found beneficial to the OVAL Language. XPath 2.0 takes what XML Schema has to offer, such as Unicode support, and then adds functionality found in most modern regular expression engines such as lazy quantifiers, anchors, grouping and back-referencing.

It should be noted that when discussing XPath 2.0 regular expressions, the use of regular expression matching was described through the interface of XPath 2.0 queries and not the use of the entire XPath 2.0 language.

Because developers working within the OVAL Language need to be able to either produce or consume XML, the idea of utilizing functionality already built into XML processing libraries seemed appealing. Calling an XPath 2.0 query is all that is needed to execute a regular expression match or grouping—something that is built into any XML processing library that supports the XPath 2.0 language. Because the matching functionality falls on XML and is independent of the programming language environment, XPath 2.0 seemed like a good choice to the OVAL team at MITRE. However, as with all other regular expression standards, it too has drawbacks.

It was stated that the largest drawback to the use of XPath 2.0 is that it is a new technology. XPath 2.0 was introduced to the world of XML in January of 2007 and has slowly been adopted by the software development community. Because of its youth, major development frameworks such as .NET 3.5 and the core Java libraries do not support XPath 2.0 fully and may lack functionality needed to utilize the regular expression matching functions, though they are working to incorporate the technology in future releases. However, Saxon has come forward to provide XPath 2.0 to Java and .NET.

Saxon is a library built for .NET and Java that provides full XPath 2.0 query support. The XQilla project is an open source C and C++ development library built on top of the Apache Xerces-C project which includes XPath 2.0 query support.

Others expressed concerns for having an unfamiliar standard used within the OVAL Language. OVAL is typically used by computer security engineers whom are mostly familiar with Perl 5 regular expression syntax. It is this present expertise that should be leveraged instead of having to force engineers to learn a separate, isolated flavor of regular expressions.

Because the utilization of XPath 2.0 queries is dependent on the existence of external libraries, the footprint of OVAL-compliant tools will increase. To go even further with this point, it was mentioned that to utilize Saxon outside of a Java or .NET application, an end user would still need the Java Runtime Environment installed. This would greatly increase the footprint of an OVAL-compliant tool whereas PCRE is a simple set of dynamically linked libraries.

At this point the requirement for a limited set of features within the regular expression standard used by OVAL was questioned. Previously, it was felt that some of the functionality found within OVAL could be accomplished through the use of sophisticated regular expressions. This behavior is undesirable because it would lead to a lack of coherence and maintainability within OVAL content. By limiting the set of features used by a regular expression a developer is forced to use constructs defined by OVAL.

While the use of XPath 2.0 regular expressions does have its merits, it was felt that the community already knows Perl 5 regular expressions and that this knowledge should be leveraged by OVAL. Though interpreters processing Perl 5 regular expressions may not always return the same results, it was felt that inconsistencies will likely occur infrequently and would be manageable.

Because the existing OVAL repositories all contain content which utilizes POSIX regular expressions, they will have to be modified to support PCRE. This may cause headaches for the repository managers, but the benefits in the long run justify the work needed. Also, the change from POSIX to PCRE would warrant the need for a major version change as it would certainly break existing content.

An option that was suggested on the OVAL mailing list that came up during the discussion was to include some sort of annotation along with the regular expression which identifies what flavor of regular expression it is. This would require extra logic in a consuming tool to identify how to rebuild the regular expression, if necessary. While this does offer a flexible solution, it was felt that it overly complicated the issue at hand and led to fragmentation as well as unwanted overheard processing by the tools consuming the data.

Overall the discussion led to a conclusion that PCRE should be explored more fully and that the work to incorporate the new regular expression standard into existing repository content is justified.

XML Footprint

As the OVAL Language has grown in both power and flexibility over the past few years, the XML footprint has also grown. This increase is in both the number of characters that are used, as well as the number of elements and attributes. Is this a concern of ours? Would a 10% reduction in size provide any benefit?

Everyone in attendance was not concerned with the file size of our OVAL files. Compression utilities do a great job of condensing these XML files.

Content creation on the other hand is something that could be made easier. The amount of references that currently must be followed make scrolling through a document a huge challenge. What is really wanted when viewing an OVAL document is a way to put everything inline and in essence remove all the references. Editors can be used to do this.

Break-In the State

The proposal discussed was about the OVAL State and if it really needs to be broken out from the test. Having used Version 5 for the past two years now, what we are finding is that states are not reused all that often. This is especially true when we are dealing with variables since each test is associated with its own variable and so a new state (that refers to this variable) is needed each time. By having the state broken out, we are increasing both the complexity and size of the XML.

Note that filters still require the state structure in some way. We would need to think about how breaking-in the states affects filters. One option will be to embed the state into the filter as well.

Another option would be to use the choice structure to allow states to be either inline or pulled out and referenced. This would help maintain backward compatibility. But if the state is rarely if ever reused, having the option just adds complexity. Also, if states are allowed to be referenced, many implementations will need to track those IDs even if the reuse functionality is not taken advantage of. That said, it was agreed that there is a real advantage to not invalidating existing content for the sake of this change.

By allowing states to be broken out, we are increasing the number of things that must be tracked and versions. Management of OVAL content would be easier if this was not allowed and everything was inline.

Might this cause an unnecessary revision as what was once not nested, later needs to be shared? We could just add an ID to a nested sated if it needs to be shared in the future. This has the advantage of not forcing a chunk of XML to be copied when it needs to be reused. But this does not seem right as it can be misleading as the shared object is now out of scope.

As a community we could try to better encourage the reuse of states. Instead of always looking to change the state when new information is available, maybe we should all be looking to reference another existing state or create a new state. This approach would work for our typical vulnerability definitions that do not extensively leverage variables, but when considering our typical compliance definitions this approach won't help.

One final point made was that OVAL is a language and it probably is not a good idea to remove functionality or features just to make content management easier, especially when tools can be created to alleviate this burden. But really what is being discussed is what is best for humans vs. what is best for machines. We are trying to find the best balance. It was noted that a tool really can deal with either approach, yet given that reuse of states is not being seen, there is a huge benefit for humans to move it back inline.

Moving forward we could work toward the inline approach and deprecate the referencing of external states. New tests that are created in the schema would not need to create the additional state structure. This goes back to the debate about cleanliness vs. backward compatibility.

This discussion would probably benefit from the creation of a pro – con list for each approach and then to have more discussion about this. There doesn't seem to be one approach that outweighs all other approaches.

Remediation Language

One area for improvement that has been identified is in standardizing how one expresses a remediation or change once an assessment has been made. This session explored this idea and discussed some of the basics about how such an extension to OVAL would be shaped. Where should such an extension live? Should it be part of OVAL similar to the way the system characteristics and results schemas are? Or should it live on its own outside of OVAL?

A separate mailing list has been created for continued work in this area (please visit http://oval.mitre.org for information about how to sign up) and the first topic for discussion was trying to figure out a definition of remediation. This question was posed to the group at OVAL Developer Days. Note that we were looking for a definition in terms of the work we are looking to do. Maybe the word 'remediation' is not the right word, but the important thing is to understand what it is we are trying to accomplish.

It was noted that the timing for this discussion is right due to the ongoing interest from the OVAL Community. This has been backed up by support for the concept exploration from both the OVAL Board and the Government sponsors. OVAL is also at a maturity state that it is feasible to start thinking about possible extensions to fill in gaps that have been identified. Work on a Version 6 of the language makes this a natural time to start this work.

An initial goal of this work has been stated as enabling IA tool vendors to consume authoritative remediation statements as well as define their own remediation statement that other IA tools can act on.

Use Cases

The definition thrown out to the mail list was: "the act or process of correcting a fault or deficiency". Should this definition be expanded to in allow for any change to the state of a system? Maybe updating software, or starting a service? This exercise was an attempt to bring out the different use cases for a remediation language as part of OVAL.

It was brought up that initial definition dealt exclusively with things that are done directly to a system. Often remediation is performed by blocking certain ports at a firewall or not allowing specific traffic into a network. Normally this type of action (outside the system being assessed) has been outside the scope of OVAL. This might be a good place to draw the line with remediation as well.

Maybe the most common use case in the OVAL Community is that of vulnerability management. In this, a user performs an assessment of a system using OVAL Definitions. The results of this assessment identify some number of vulnerabilities that were found on the system. The next logical step is to make a change on the system (often install a patch) to remove that vulnerability. So there needs to be a way to express what changes can be made to fix a vulnerability that an OVAL Definition has described.

Of course OVAL is not just focused on vulnerabilities and as pointed out by the group, maybe at the same level as the previous use case is one focused on configuration management. Similar to the vulnerability use case, an OVAL Definition is used to evaluate a system and determine if the system is compliant with a specified configuration setting. Once the result has been gathered, the next step is to bring that system into compliance by changing the setting to a desired value.

A final use case discussed was software management. In this use case, a tool evaluates a system for the versions of certain software, and if it finds out of date or unsupported version it can update that software to the newer / older version as necessary. Oval remediation could be used to encode the type of information necessary to perform such an upgrade. It was quickly pointed out to be careful to talk about “automated” and not “automatic”. It was also noted that certain remedies often require certain software or service packs to be installed and it may be necessary to bring a system up to a system level before installing a patch.

Community Proposals

To give a more complete picture of the current landscape, a couple of vendors were asked to present proposals that they had on this topic and see if any of it could be used as a foundation for bringing remediation into OVAL.

Hewlett-Packard Development Company Proposal

HP has been working for over a year now providing their customers with OVAL Content to help them determine if a system is vulnerable. Unfortunately automation stopped at the assessment and couldn’t be used to help the user fix the issue. Their need aligns closely with the vulnerability management use case.

Since HP was already writing content under the OVAL framework, they figured it would be efficient for them to extend OVAL to accomplish their needs in remediation. During the content creation process, HP has realized that most of the information needed for remediation is already looked at and all that is needed is a way to represent it.

Three main points were brought up as design principles that were important to HP:

1. backward-compatibility with the current OVAL schemas
2. ease of understanding for those familiar with the current language
3. ability to extend as new types of remediation are needed

The basis of their proposal was to add a <remediation> element as a child to a <criteria> element. This would allow a content writer to add one or more <rmdcriterion> elements to describe specific remediations that could be performed relative to that parent <criteria>. Each <rmdcriterion> would be associated with a specific <remedy>, similar to the way a normal <criterion> is associated with a <test>.

```
<definition>
        <metadata> …….. </metadata>
        <criteria operator="AND“ comment="Win2K,SP4">
                <extend_definition comment="Win2K is installed" definition_ref=" def:229"/>
                <criterion comment="mqrt.dll version is less than 5.0.0.805“ test_ref=" tst:6814"/>
                <remediation>
                        <rmdcriteria order_number=”1”>
                                <rmdcriterion remedy_ref="rmd:0001"
                                            order_id="1“
                                            comment="install patch"/>
                        </rmdcriteria>
                </remediation>
        </criteria>
<definition>
```

One of the important parts of this proposal was the fact that remediations are not established as a 1-to-1 mapping with the definition. In HP's experience, there are often different patches for different types of systems (e.g. a win2k patch and a winxp patch) and the definition criteria already expressed this breakdown. By nesting the remediation elements inside the criteria, we can take advantage of the OVAL's present breakout of logic.

ThreatGuard Proposal

In ThreatGuard's case, compliance management presents their biggest need for a remediation language. To achieve this, the first question they had to answer was when to apply a remediation. It only made sense to use the test that was already in use and add remediation logic to it.

As with the previous presentation, ThreatGuard pointed out that it was important to leverage the existing OVAL structure both to ease content creation, but also tool development. Due to ThreatGuard's focus on compliance content, the solution proposed was to use the existing test structure and add a <fix> element that would point to a specific <_fix> that provided details related to the given test type. By using this test structure, the identifiers for the original test and object were accessible and allowed for additional information to be gathered if necessary.

```
<registry_test id=" tst:2" version="1" comment="Registry key Limitblankpassworduse=1"
                check_existence="at_least_one_exists"
                remediate="all">
          <object object_ref=" obj:1" />
          <state state_ref=" 120" />
          <fix fix_ref=" fix:123" />
</registry_test>


<registry_fix id=" fix:123" version="1" test_comment="Registry key Limitblankpassworduse=1">
          <value variable_context="true" operation="equals" datatype="REG_DWORD">
                    <options>
                              <option context="fdcc">1</option>
                              <option context="default">1</option>
                    </options>
          </value>
</registry_fix>
```

One of the challenges that ThreatGuard faced was that there is no notion of priority within an OVAL criteria. If three different tests were OR'd together and all three came back as not compliant, then which fix should be performed? Similarly, there may be multiple fixes associated with each test and it is unknown which one to execute.

As for generating this type of remediation content, challenges were when the values in OVAL were open-ended. For example, when a test checks that a value does not equal 0. What value should be set during remediation?

Development Ideas

The rest of the discussion was focused on ideas and past experience related to remediation and factors that OVAL should consider as it goes down the remediation path.

Maybe the most fundamental point is that remediation is a matched set. As a remediation tool, you can’t just blindly perform remediation and hope that it works. Rather it is important that the tool really knows that the system needs the remediation. Because of this, it is important that remediation logic inside of OVAL be closely related to the testing logic.

Another important note for remediation tools is that they cannot trust results coming from a 3rd party. The reason for this is that time will have passed it in theory the issue could have been resolved, or software updated such that a given remediation is no longer applicable. This makes it necessary for a remediation tool to run some amount of testing logic to verify the applicability of a remediation before applying it.

Even though the tests and remediation need to be accessible at the same time, separation of the testing logic from the remediation logic was also brought up as important. The reason for this is to keep the content clean enabling easier validation of the content.

The ability rollback a specific remediation was mentioned. The group seemed to split on it viability, let alone its importance. It was mentioned that it was not possible to do this correctly and often this type of feature is kludged together to satisfy a marketing need. A couple of vendors in the room mentioned that they do indeed have rollback working in their products. This was brought up as it should be something for OVAL to consider in the remediation space.

There needs to be a way to group or relate like items. For example, if you enable a specific setting, then other changes must also be performed.

Really what we are doing is developing a system manipulation language.

There is a lot of work to be done relative to a remediation language and this discussion has been a great start and will really help get the work started.

Wrap-Up

A huge THANK YOU goes out to all the community members that attended OVAL Developer Days and shared their opinions about the topics being discussed. It is this community participation that drives the development of OVAL and enables us to progress down the standards path.

Action Items

- further development of the <oval_unit> and applicability ideas
- determine how best to support a definition referencing capability
- explore the creation of an open source repository application for organizations to leverage
- explore the creation of a network device schema – what types of tests would be included?
- set up a sandbox area for experimental new schema features
- look into limiting the SQL test to just simple SELECT statements
- look further into switching regular expression support to PCRE
- outline the pros and cons of composition vs. extension for the proposed object change
- look into adding a ‘resolve well known sid’ behavior
- outline the pros and cons of breaking-in the state
- create a draft remediation language and submit to the community for comments / suggestions