OVAL Developer Days

July 18-19, 2005

Introduction

OVAL Developer Days was held on July 18-19, 2005 at The MITRE Corporation in Bedford, MA. The purpose of this event was to focus on discussing, in technical detail, the more difficult issues facing version 4 of the OVAL Schema, and to drive the development of version 5. By bringing together the lead proponents within the OVAL Community, the goal was to derive solutions that would benefit all parties, and continue the development of the language. What follows is a detailed summary of the discussion from this event.

Attendee List

ArcSight	-	Raffael Marty
Bastille Linux	-	Jay Beale
Center for Internet Security	-	Dave Waltermire
	-	Bert Miuccio
Citadel Security Software, Inc.	-	Kent Landfield
	-	David Lewis
Department of Defense	-	Melissa McAvoy
	-	Neal Ziring
DesktopStandard	-	Barry Day
DISA	-	Matthew Kerr
Microsoft Corp.	-	Bashar Kachachi
	-	John Wilson
	-	Mark West
MITRE	-	Jon Baker
		Matthew Burton
	-	Andrew Buttner
	-	John Carlson
	-	Bob Martin
	-	Marion Michaud
	-	David Proulx
		Harvey Rubinovitz
	-	Charles Schmidt
	-	Ingrid Skoog
	-	Dr. Todd Wittbold
	-	Matt Wojcik
	-	Jackson Wynn
	-	Margie Zuk
NetClarity	-	Michael Su
netForensics	-	Anton Chuvakin
PatchLink Corporation	-	Henry Sprafkin
ThreatGuard	-	Rob Hollis
	-	Randy Taylor
TriSixty Security	-	Joe DiPietro
	-	Itamar Heim
	-	Yuval Kashtan

Day One

Session 1

OVAL ID Format

In the current identifier process, if an organization is to produce OVAL content, they are required to obtain unique ids from the MITRE OVAL team for each of their definitions, tests, and variables. Attendees felt MITRE's capacity in this role will only serve to slow down the creation of content and stretch available OVAL resources too thin. A more desirable approach would allow an organization to use a format or id range to assign identifiers with little, to no, coordination with MITRE.

Two separate issues were established in the identifier discussion; how an id should be modified in order to preserve uniqueness within a set of different definitions, and the appropriate amount of information an id should have incorporated in it.

Some proposals for discussion:

```
- currently      OVAL123
- full name      OVAL123ORGANIZATION
- limited name   OVAL123ORG
- hyphens        OVAL-ORG-123
- comment        OVAL_check_password_length_123_ORG
- dns-style      org.cisecurity.windows.xp.123
```

General consensus was the ideal OVAL id format is one of simplicity. An identifier's main purpose should be to guard against collisions and shouldn't have its scope expanded to contain additional information. Although real world scenarios exist, in which customers use an id to tag specific content and to lookup information about it, the OVAL id should not contain a description of the content. Attendees stated the proper source for a definition's descriptive information should be obtained from other fields within the XML.

Were a DNS-style approach used, attendees agreed a great deal of work and discussion would be necessary to clarify what information was held inside an id. Both the DNS-style and comment style options open up the chance for a definition to exist with a specific id that, if the definition was modified, could cause the id to become misleading or incorrect. As an offered example, an id format could include just the Microsoft Windows 2000 platform and be labeled oval:win2k:123. However, if the definition for oval:win2k:123 was altered to also apply to Windows XP, then the id would no longer convey the true nature of the definition.

Another offered solution would have OVAL identifiers handed out in ranges, as is the case with CVE Candidate Numbering Authorities. A noted downside to this approach occurs when the full range has been used and the producer must go back to the id source for more numbers. The MITRE OVAL team wishes to remove this type of dependency. Big ranges handed out to several, separate customers, would also cause the id numbers to grow in length quite quickly. Attendees preferred to keep all ids as small as possible.

It was noted that, another factor that must be considered when defining identifiers is that identifiers should contain information about where to find the content. If OVAL content becomes hosted by multiple sources instead of just the MITRE repository, then a user who is told to look at a particular OVAL definition must be able to find it.

In the end, the consensus opinion was a URN style scheme that followed the following form:

```
oval:{trigraph}:{number}
```

Such an approach is a simple, relatively short format that provides global uniqueness and would allow MITRE to give out specific trigraphs to each organization that wants to produce content. Once an organization registers with OVAL and obtains a trigraph, they would be in charge of determining the numbers used for their definitions. Attendees also noted the benefit in matching organizations to their respective OVAL content, should the community wish to approach the respective organization with questions regarding their content. A pointed downfall in this scenario is MITRE's role in deciding who gets what trigraph.

Short Name

Some participants suggested the addition of a short name to a definition to ease a human user's distinction between various definitions. Organizing the work that has been done can pose a problem for the organizations that create OVAL content. These organizations can currently either view definitions by a non-descript OVAL identifier or a by a long description. The advantage of a short name, attendees offered, would be the quick description of the issue in question, and would allow definitions to be easily distinguishable from one another. The short name field could even be thought of as an organizational id; it could be limited in length and exist as an optional field left up to interpretation for each unique creator organization.

Attendees questioned if the short name field should also be forced to be unique, but then rationalized it would be impossible for humans to write meaningful unique names. It is acceptable to have a meaningful name that might collide with a small number of other definitions if the tradeoff is a means to provide some way of distinguishing that definition from all the rest.

Repository

The current repository set up allows organizations to submit any content to the OVAL community for review. This process was originally developed when the OVAL project had the goal of producing a centralized vulnerability database of unique, and accepted, OVAL definitions for each issue. Since that time, OVAL has expanded into policy compliance, patching, and, in fact, the language is now positioned to describe any single machine state. The idea of one accepted definition per issue, to be maintained by the community, came under question. Organizations that create content to express their own proprietary views on computer security don't always want the community to review, change, or accept their ideas. Content open to the community seems to be acceptable, but allowing the community to edit certain definitions won't work. It was decided the nature of the OVAL repository is an issue that needs to be resolved by the OVAL Board.

XML Signatures

In light of an individual organization's ability to create content for the community, there needs to exist a way to verify that content is unaltered and that customers can test for this fact. In response to this need, XML signatures are allowed in version 4.1 of the OVAL Schema, but only at the document level. The question was posed whether signatures need to be allowed for each definition and test. One potential solution would be, to add a specific signature id attribute to each definition and each test that could be referenced by the signature for that item.

It was noted that pieces of the XML can be signed by specifying an XPath to what is being signed. Essentially, the signature applies to a specification within the XPath and that signature can be placed anywhere within the XML document. The task of adding a signature for each definition and test could already be accomplished in version 4.1. Keeping all the signatures at the bottom of the document may improve the readability of the document, but would also make it difficult to process and to maintain.

Attendees also addressed the question of whether a <signature> element should be available inside each definition and test. Many felt that this was a good solution because, with the signature contained within the definition and test, it would allow the item to be reused by someone else, while maintaining the integrity of the original item.

As well as the addition of optional signature elements to the definition and test items, it was agreed there needs to be a document outlining how to use signatures within OVAL. This document would serve as a guide for OVAL content producers.

Combination of Definitions and Tests

There isn't a means in OVAL schema version 4 to reuse definitions and extend existing definitions to create new ones. Community members expressed interest in the ability to use each definition as a building block to create new content, similar to how tests are currently used. One proposed solution was to remove the distinction between definitions and tests, and have a single, logical construct. Attendees agreed that, in either case, more metadata must be added to tests. If the tests and definitions remain separate, then the identifier format issue will need to be extended to tests. If the two are combined, the effects on the criteria section will need to be examined to ensure no loss of existing functionality.

It was agreed that a separate file to hold metadata would be neither beneficial nor desirable. Keeping everything in one XML file is best, since it allows the use of XPath to gather information. The idea of optional metadata items was also deemed as inadequate, as consumers need to know what information they can expect to receive.

During the discussion, attendees backed two separate proposals. The first proposal would have a one to one relationship between definitions and tests. In this scenario, when a member of the OVAL community wanted to reuse or extend a definition, they would use the single test associated with its given definition. This case would keep the current distinction between definition and test in tact, which is good, since a definition is what currently gives meaning to the tests. The other proposal was to move the metadata into the test structure and remove the OVAL definitions as they currently exist. This approach would simplify the language, but lose the distinction between definition and test. A decision about which path to follow was not reached and it was decided that this needs to be discussed further over the OVAL Developer list.

Session 2 - Track 1

The discussion for the first track of session two was focused on breaking object sections out from the existing OVAL test structure. Members of the OVAL community wish to express complex objects that can't be tested under the current version 4 schema. Some examples of this need include: gather information about all the users in the Administrator group, and the user FRED, identify all the files owned by a particular account, or identify all the files that a specific user can modify. Another drawback of the current approach is any object that needs to be tested multiple times, has to be restated within each test. An ability to reuse common object declarations could help clean up the OVAL language and simplify its design.

To start the discussion, an overview of the current version 4 architecture was presented. Each OVAL test consists of an object section and a data section. The relationship of the two sections is determined by the check attribute. Possible values are 'all', 'only one', 'at least one', and 'none'. In other words, the check attribute determines how many matching objects defined by the object section must satisfy the data section requirements for the test to return true. The object and data sections were introduced in version 4 to help tools understand what information is used during data collection.

Breaking Out The Objects

The following schema change was proposed as a means to separate the object section from the OVAL tests:

Example of separate object section

```
<oval>
     <definitions>
          ...
     </definitions>
     <objects>
          <object id="obj_1">
               ...
          </object>
     </objects>
     <tests>
          <test obj_ref="obj_1" check="">
               <data>
                    ...
               </data>
          </test
     </tests>
</oval>
```

The objects would be referenced by their respective tests and the check attribute would continue to operate as in the current schema. This proposal solves the issue of object reusability within a definition and also allows for a more complex object. The proposal to break out the object section could provide a lot of power and flexibility to the OVAL language, but attendees pointed out every attempt should be made to keep the complexity hidden from those that don't need it. As a given example, if an organization wanted to write a test to simply check the value of an individual registry key, they shouldn't have to write a complex object declaration to accomplish it.

Another proposal was to keep the object section as is, but add the ability to reference an external object. This could be done via a choice group in the schema that would allow either an <object> element or an <obj_ref> element. Attendees agreed this idea needed more thought and was marked for further discussion over the OVAL Developer list.

There is also a desire in the community to run a given test against multiple instances of a declared object. For example, an Oracle server might have a number of instances running and a tool might wish to test each instance against a particular set of attributes and gather the results of each. In other words, the schema functionality would allow users to obtain the results for each individual instance instead of the result for the collection as a whole. It was determined that this ability is already available with version 4, as the results file will report each matching object using the <tested_object> section. Each of the <tested_object> elements has its own result attribute, which allows an individual to look at the results file and determine the result of each instance. Attendees asked that this functionality be better defined in the documentation for the Result Schema.

As a side note, the current results and system characteristics files would not change with any of the above proposals.

Behaviors

The group regarded a possible behavior element to help define how data collection should proceed. An example of such an element would be a recursive behavior to control the examination of file objects. This behavior would guide the processing of a supplied directory during data collection. Two possible approaches would be, to either look for all files in the directory, or look for files in the directory and its sub-directories.

One proposal was to have a <set> element that would contain a <behaviors> section. This would allow an object declaration to have multiple sets with different behaviors. These sets could then be related through a set operation such as: intersection, union, subtraction. For example:

Example of object behavior

```
<object id="obj_1">
    <set op="intersection">
        <set>
            <behaviors>
                <recurse_direction>down</recurse_direction>
            </behaviors>
            ...
        </set>
        <set>
            <behaviors>
                <recurse_direction>up</recurse_direction>
                <max_depth>4</max_depth>
            </behaviors>
            ...
        </set>
    </set>
</object>
```

It was pointed out that each type of object would probably need different behaviors available to it. For instance, the recurse_direction behavior in the example above might not be valid for a set of ports.

Object Test

An idea was proposed to create a new object test that would allow a content creator to test the makeup of a given object set. For example, does one object set equal another? Is one object set a subset of another? The group consensus was, there are four major things to test for in this case: existence, subset, superset, and equality.

Example of object test

```
<object_test id="">
    <operator>subset</operator>
    <base_obj_ref>obj_1</base_obj_ref>
    <testable obj ref>obj 2/testable obj ref>
    <testable_obj_ref>obj_3</testable_obj_ref>
</object_test>
```

Filters

The final topic of the session focused on adding a filter to the object declaration. A filter could refine a set of objects, based on a criterion that is currently considered a data element. In effect, applying a filter would be using the data section of an existing test to define the set of objects to act upon for another test. This could allow even greater flexibility in the creation of object sets to test against.

Example of a filter

```
<object id="obj 1">
    <set op="intersection">
        <behaviors>
            <recurse_direction>up</recurse_direction>
            <max depth>4</max depth>
        </behaviors>
```

```
            ...
            <filter test_ref="wrt-3"/>
        </set>
    </object>
```

Session 2 - Track 2

OVAL Compatibility

The OVAL Compatibility Program is an important piece in promoting the OVAL standard and in ensuring it is implemented within the industry in a manner that assures interoperability. Currently, there are 9 organizations that have declared OVAL compatibility for a total of 19 products. The first draft of OVAL compatibility requirements was released to the OVAL Community on 13 July 2005. The purpose for this track was to present the goals and motivations for the OVAL Compatibility program, review the individual requirements, and gather feedback from those in attendance. The session opened with a quick review of some envisioned uses for OVAL, which have been discussed with various organizations in industry and government and shown on the OVAL web site page "How OVAL Compatibility Improves Vulnerability Management." OVAL use beyond vulnerabilities, for configuration checks and patch applicability checks, was also reviewed.

Currently, the OVAL Compatibility program is viewed as a process of steps, where each subsequent step is somewhat more difficult to accomplish. First, an organization that has a shipping product or publicly available capability declares their intent to be compatible with one or more of the OVAL Schemas. Second, a capability, or set of capabilities, can be declared to support the OVAL Schemas in their currently shipping product or their currently available capability. Third, the organization completes a compatibility questionnaire, detailing the incorporation of the declared OVAL Schema into their capability. Finally, some type of a hands-on evaluation of the capability is conducted to ensure that the OVAL Schema have been incorporated correctly.

Attendees discussed the time, effort, and need for such a program from both sides of the process. From the vendor perspective, there were questions associated with an added effort to provide the necessary compatibility documentation and support, the time-frame of the compatibility process to complete, and the added value of the 'OVAL Compatible' logo for their product. From the evaluator's side, there was concern that the top level of compatibility could become a long, laborious process, for something that doesn't merit such a level of effort. The participants were reminded that the compatibility program is still in the process of being defined, and that the level of effort required to obtain and evaluate compatibility will certainly be taken into consideration, to ensure that the program is reasonable for all involved. The basic approach was described as having enough validation and evaluation to protect the investment that vendors make in incorporating OVAL support and to assure customers that capabilities that claim to support OVAL actually do so.

The question was raised whether there is need to obtain OVAL Compatibility in order to use the OVAL language in a capability. MITRE clarified that no such requirement exists. Participants also questioned whether OVAL might follow the lead of other standards efforts in industry where the use of, and compliance with, a standard is self policing. The argument against this was two-fold; an official compatibility program will better ensure interoperability between tools, and the U.S. Government has started creating contracts that call for the inclusion of standards compatible security tools.

Discussion touched on several aspects of individual requirements. Some items of note included:

- Attendees felt compatibility testing for OVAL Content Producers to be easier to conduct than for Consumers, as there is more opportunity to automate tests used to examine how content has been produced.

- Participants agreed compatibility for OVAL should be driven by the different use cases for the language within industry, rather than borrowing an evaluation approach from other initiatives.

- If an organization obtains compatibility, but does not stay up-to-date with revisions to the OVAL Schema, at what point should they no longer considered OVAL Compatible? There was consensus that the market would drive this issue, but compatibility should require organizations to state their policy for updates.

- With respect to the draft requirement 'The tool MUST be able to consume all definition content that properly validates against the schema for which the tool is stating compliance' – the question was raised whether all types of definitions are the same (e.g. vulnerability, patching, policy). Currently, all definition types will be treated the same. However, as the language evolves, this may not be the case and should be considered in the Compatibility Program.

- The draft requirement 'Duplicate definitions MUST be resolved through interaction with the Review Authority' raised the issue of a central repository for all OVAL content. This issue was touched upon during a number of sessions and is an issue that will carefully be considered by the OVAL Board. Attendees agreed under ideal circumstances, such a repository would exist. However, such a repository may not be feasible as a universal practice. Another possible solution would consist of several central repositories of "common definition building blocks."

- Participants discussed requirements governing the interface between producers and consumers. Consensus agreed such requirements as out of OVAL's scope.

Based on the comments and feedback from this session, the OVAL Team plans to revisit the Compatibility Program requirements and to produce an updated draft of the document for posting on the OVAL Web site as an initial public draft.

Session 3

Patching Definitions

Interest has been expressed in using OVAL to support patching. While vulnerability definitions often include tests to see whether a patch has been applied, a different use case is for OVAL definitions which describe on which systems a patch could or should be installed. The "patch" definition class exists for this purpose, but it has never been fully considered or defined. The purpose of this track was to present proposals for the meaning, structure, and limits of patch definitions, and to identify any necessary language additions.

The relationship between vulnerability definitions and patch definitions can be misunderstood. While patches and vulnerabilities are often linked, an OVAL vulnerability definition ultimately tests for a security flaw on a system. It does not prescribe a fix—it is fundamentally about a problem, for which there might be several possible fixes (or none). A patch definition would instead be about a fix, and would test for systems to which the fix might be applicable. The fix may also address multiple problems, some or all of which might not be security-related.

Proposals were presented about several aspects of patch definitions, including their overall meaning, categories of tests, and superseding information.

Overall Meaning

If a patch definition is true on a system, it means that patch is appropriate for that system—appropriate in a technical sense, not in terms of policy. A patch definition is descriptive, and not prescriptive: it indicates on what systems the patch could be applied, but doesn't assert that the patch should be applied in a particular real-world situation. There

might be issues outside the scope of OVAL such as policy directives, interaction with homegrown applications, etc. that affect the decision of whether to actually install the patch on a given system.

(The use of the term "appropriate" was questioned, as it is likely to have different implications for different people. There does not seem to be any term which would not have the same problem, however.)

This understanding of a patch definition implies pre-install tests. To support the proposed semantics, the definition would include checks run before any patch installation would be attempted. An open question is whether an OVAL patch definition should have a section for post-install tests, perhaps to check for successful installation. Also, some patches require operations (either pre- or post-) outside of the installation itself, for example a reboot. Should the definition somehow record those? "Reboot required" might be a simple flag in metadata, but that approach may not be general enough for other cases.

The issue of side effects was raised. Some patches may change an application's version or behavior. Perhaps the definition should also somehow indicate side effects. How to express side effects is unclear: unstructured text, for human consumption? Some kind of organized metadata? Machine state expressions in OVAL (a new application for "tests") indicating how the system would be changed?

Another question is how to address local requirements around a patch. For example, local policy might be to never install Patch A on systems which have Application B version 3 installed, perhaps because operational testing is incomplete. Should OVAL facilitate this more directly than it does currently? This may be a non-issue if general support for extending definitions is added in OVAL Version 5.

Categories of Tests

Patch definitions may benefit from different categories of tests than the <software> and <configuration> criteria sections supported in OVAL Version 4. Those groupings were originally conceived for vulnerability definitions, and while they also apply fairly well to compliance definitions, they may be inadequate for patch definitions. Proposed categories include:

- Relevance: tests to determine if software is installed on the machine for which the patch is intended. If the relevance section is true, there is installed software which contains a flaw that the patch is meant to fix. If the relevance section is false, there is likely no need to consider any of the other information in the definition.

- Pre-requisites: tests to determine if all pre-conditions for patch installation are met. These could include any precursor patches (if there is patch interdependency), necessary software upgrades, or other configuration settings. Arguably, checks for sufficient disk space could be in this section, though perhaps that would be better dealt with in metadata, or is even an application issue beyond the scope of the OVAL definition. There may be other types of pre-requisites still unidentified. It should also be recognized that even if a pre-requisite is not met, the patch may still be desirable!

- Conflicts: tests to look for any counter-indications for patch installation. For example, if there is a known incompatibility with a different critical application, this section could check for its existence.

An outstanding question with all of these sections is whether they are granular enough. The lack of sufficient disk space may be actionable or reportable in itself, which could be difficult to support if it's just reflected as a test or tests in a general pre-requisites section.

There are some implications of the relevance and pre-requisite sections which may not be obvious, and may be debatable. For example, Patch A is released, and the vendor states that it addresses a flaw present in versions 6 and 7 of the target application. However, the patch cannot be directly applied to version 6; an upgrade to version 7 is required first. For Patch A, the relevance section would contain a test for version 6 OR version 7. The pre-requisite section would include a test for version 7. The logic is: if the system has either version 6 or version 7, there's a flaw present that Patch A is intended to fix. However, unless version 7 is installed, the patch cannot be applied because a pre-requisite has not been met.

This approach allows for more meaning than a single block of tests. It is unclear how far the relevance section should be pushed, however. If a flaw is present in an old, unsupported version of an operating system, and a patch is only available for the current version, it may be unreasonable to identify the patch as relevant to the old version and include an operating system upgrade as a pre-requisite, because of the level of effort and the impact of performing that kind of upgrade.

It was pointed out that this structure means that the definition as a whole would not have a simple true or false result. One result could be that a flaw exists but it cannot be fixed immediately (relevance section true, pre-requisites not met or conflicts exist). Another result would be that a flaw exists and the patch could be applied (relevance true, pre-requisites met, and no conflicts). Another would be that the patch is not applicable to the system (relevance false). This is true, and a primary motivation for having these categories.

Superseding Information

Any patching process, manual or automated, must take superseding or obsolescence relationships into account. If Patch C is released and supersedes Patch A, it may be a waste to consider Patch A for installation. It seems OVAL should support tracking these relationships.

One question is whether to maintain "superseded by" or "supersedes" information (or both). For example, should Patch A's definition indicate that it has been made obsolete by Patch C, or should the definition for Patch C state that it makes Patch A obsolete? These relationships evolve over time, of course. It could be much easier to track "supersedes" than the inverse. It may also be possible to derive one from the other.

Issues remain regarding how to record this information in a definition: via tests? In the metadata, perhaps as a list of patch identifiers?

New Tests

If patch definitions are going to be fully realized in OVAL Version 5, any new test types necessary to deal with patches must be identified. For Microsoft Windows platforms, .MSI files and the .NET framework have been suggested as areas that may require new tests. Further investigation is needed, for Windows and other platforms.

Other Issues

Areas for further consideration include:

- Patch identifiers: how should an OVAL definition uniquely identify a patch? Vendor names for patches as published in advisories or bulletins are not always unique, sometimes requiring (for example) information on the target platform or the date of the patch release for true uniqueness. Should patch identifiers be standardized?

- Platform and application metadata: indicating the target platform and/or application in the definition's metadata could be very useful. Should these be standardized?

- Descriptions: should patch definitions contain a textual description? What sort of information should be included?

- Issues addressed: should there be an indication of what the patch is intended to fix? For security-related patches, a list of CVE identifiers in the metadata would seem natural. Is there any analog for non-vulnerability fixes? Is it desirable to structure this data, or does it belong simply in a textual description of the patch?

- Cross-platform issues: are the semantics described applicable to multiple platforms? This work was done primarily in the context of core Microsoft Windows operating system patches. Different operating systems' and major applications' approaches to patching need to be considered as well.

Day Two

Session 4

The OVAL results schema has expanded as new requirements were identified. It originated as a simple format to report the results of an OVAL vulnerability analysis, but has since grown to support a variety of other needs. These modifications to the results schema prompted attendees to raise concern over its verbosity and effectiveness.

The current version 4 schema has a <generators> element to holds information about the OVAL definition and system characteristics files used to do analysis. The results schema has a <system_info> element to hold information about the analyzed system, a <definitions> section to outline the definitions used for analysis and reporting, and a <tests> section to outline all tests used for analysis and to report each corresponding result. The outline of each used test contains a <definition_object> section to restate the object declaration from the original test. Each outline also holds a <definition_data> section to restate the configuration items examined by each test. Every outline consists of one or more <tested_object> sections to describe each matching object and the result of each object, as it relates to the <definition_data>.

Verbosity in the Results Format

Participants were concerned that the average results file is too verbose: packed with repeated information that already exists in the system characteristics and definition files. Several attendees pointed out that the information needed by a results file would differ with each use case, and that, at least two different formats might be needed. One possible format would be a full version with all the repeated information for certain tools generating in-depth reports. Another format would be a slimmed down version that only contained the false results. Any slim version considered by the OVAL community would have to be a subset of the full version.

The proposal of two different formats led the audience to question whether OVAL should have two separate schemas to validate these different formats, or whether stylesheets could be used to transform a single format, such as the full version, into small files aligned with each specific use case. An attendee proposed having OVAL result producers always create the full results format and OVAL results consumers use stylesheets to tailor the results to their individual needs. Some draw backs with this approach include; in many instances the entire results file would be transmitted over the network, when only a slim version was needed, and tools that do not generate a system characteristics file may not be able to generate the full results file.

Attendees discussed the option for a consumer to send a stylesheet to the producer. This stylesheet would trim the results file to exactly what the consumer would need, and could give control to the consumer as to what information would be sent over the network. This approach would still require the producer to create the full format, however, as well as developing the need to establish a communications path between two tools.

Some participants offered to have a results format that only reported failures during an analysis. This modification could greatly cut down the verbosity of the results file. This change could cause problems in future analysis of results files when a new vulnerability is discovered and a user wants to examine the file to see if they are vulnerable. By reporting only failures, the necessary information would not be available for such a determination.

Participants tried to meet on some common ground by discussing the idea of having a single format, but making a number of elements optional, thus allowing for a slimmed version file to be produced. A problem with this

approach, again, was proper communication and sharing of information between OVAL producers and consumers. With optional elements, the consumer would not know the specific information contained within the file, meaning the benefits of having a common, standard format would be lost.

Consensus was that, at minimum, two different OVAL results formats are necessary, but the specifics of these modifications will require further thought and discussion.

Archiving

For archiving purposes, certain attendees felt that having repeated information about the definition used, and the system information collected, allowed one to go back at a later date, and verify a result for auditing purposes, even if the OVAL definition used in the test has changed in the meantime. A timestamp could be added to the results file that would provide a link back to the original set of definitions used to generate the result file. The drawback is that vendors would have to rely upon external content producers to support their archiving capability. Another proposed solution would be to archive the definition file and system characteristics file used to generate the results file.

Use Cases

The following use cases were listed in reference to potential results format changes:

```
Initial OVAL Result Schema use cases

1) Policy Compliance - reportable results implies details
2) Auditing - full (?) system info plus what was tested
3) SIM or other aggregation - lower network traffic, option for full details
```

Note: this list is not considered to be the definitive set of result file use cases. Based upon further discussion on the OVAL Developer mailing list, a more complete set of use cases will be generated, and used to drive the requirements for the OVAL Results Schema.

Results for Multiple Systems

A main concern within the OVAL community is the inability to support analysis done on multiple hosts. Currently, if 1000 systems are analyzed, then 1000 system characteristic files and 1000 result files are generated. If these 1000 files were to be combined into a single file, there would be a vast amount of duplicated information. To avoid this repetition, a complex set of referencing could be added to reduce the size, which in turn, would increase the complexity of processing.

If multiple files were to be combined into a single format, would the structure be grouped by the different systems, or would the structure be grouped by different definitions? Both choices would produce a unique set of unavoidable redundancy within the file. If multiple hosts were to be included in a single file, the files used to do the analysis could not be assumed to be the same for each system and would have to be reflected in the <generators> section. A participant also pointed out the benefit of being able to use XPath within one single file of 1,000 systems, versus the separate 1,000 single files. The XPath benefit would allow a user to uniquely identify any desired element within a given document.

Different use cases will be gathered through the OVAL Developer mailing list, and these will be used to help formulate a new solution.

Session 5

Pass/Fail Logic

The current valid result flags within a result file are 1 (pass), 0 (fail), and -1 (unknown). Attendees decided there needs to be a second field in the XML to give further meaning to the unknown cases. Various OVAL users want to know why the result is unknown. Was the test or definition is not applicable to the system being analyzed? Was there is an error collecting the needed data? This sort of information will be very beneficial to tools during the analysis process.

The discussion then switched to procedural logic within OVAL. Participants asked if there is a way to declare certain sections of definitions or tests to be evaluated only if specific criteria are met. For example, could a user evaluate one test, when analyzing a Windows system, and evaluate a different test when analyzing a UNIX system? Under the current model, a definition could have a test that checks one setting if looking at a Windows system and a different setting if analyzing a UNIX system. In the event that this test is run on a Windows box, but the setting is incorrect, the Windows side of the test would fail, causing the UNIX test, which is inappropriate for this case, to be incorrectly used for evaluation. If logic exists to short circuit the test after the Windows section, the UNIX test would not be considered, and the overall process would run more efficiently.

Participants offered the idea of an ONPASS operator to try to meet the logic issue half way. Being an operator, it wouldn't necessitate a significant change to the current design of OVAL. The operator's implementation would be defined by looking at the first test in the list, evaluating it, and, if passed, evaluating the rest of the tests. If the first test failed, then the remaining tests would be skipped. This approach would only allow a single test to be used in the 'if' clause. Some attendees felt uneasy about allowing such logic into OVAL. They felt that such modifications would introduce procedural concepts into a declarative language, although they could not produce concrete examples at the time to show why such procedural inclusion would not be a good idea.

The namespace associated with most tests could answer some of these concerns. As an example, when a tool running on Windows encounters a test from the UNIX namespace, it will know that the test is 'not applicable.' Unfortunately, this would not help distinguish tests that are within the same namespace; for example, tests for a specific Window service pack, or tests based on an application version.

Session 6

The OVAL Team has developed a suite of tools to assist vendors to incorporate OVAL into their tools. The current tools include; the Definition Interpreter, which conducts a vulnerability assessment of your system using solely OVAL Vulnerability Definitions, the Definition Writer, which assists in the creation of OVAL Definitions, and OVAL Web Services, which provide applications with an API into the OVAL content repository at MITRE.

Definition Interpreter

The OVAL Team asked what could be done to the Definition Interpreter to make it more useful to the OVAL Community. One attendee proposed to make the interpreter more modular so developers could incorporate individual components more easily. While some found the tool useful as an initial introduction to OVAL, the consensus was that there is little use for the current interpreter in their own products. Participants felt the interpreter is more oriented toward a niche audience.

Definition Writer

The MITRE internal OVAL definition writer is a php-based tool, with a mySQL backend. Attendees felt this set up is not ideal for most organizations; however, a more lightweight version could be useful. As an example, a writer

that would rely solely upon XML files, rather than a database, with its sole output OVAL XML document. There is no current requirement to use a specific definition writer in the generation of OVAL content.

OVAL Web Services

The current, and proposed, set of Web Services focuses on functions to search, manipulate, and retrieve definition related data from the OVAL Repository at MITRE. Since the role of a central repository is in question, therefore the utility of the OVAL Web Services is also in question. Attendees saw little use of the Web Services to commercial developers.

Community Feedback

When the OVAL team asked for feedback on tools or support they could provide to assist in the adoption, development, and implementation of the language, the overwhelming participant response was in the area of content generation and management. In addition to a lightweight definition writer, documentation outlining best practices for generating OVAL Definitions, and assistance to incorporate OVAL into a tool were requested. Attendees felt without necessary documentation to guide a new developer through the OVAL process, the initial barrier to enter into the community may appear too much.

Other suggestions for support include:

- A test suite of definitions, tests, and corresponding results to help developers validate the results generated by their tools.

- A notification or subscription service to warn tools when a definition or test has changed.

- A tool to check for the use of best practices when creating new OVAL Definitions.