The MITRE Corporation

Security Automation Developer Days – OVAL Discussion

June 14 – 16, 2010

Introduction

The fifth annual OVAL Developer Days was held in conjunction with the Security Automation Developer Days from June 14 - 16, 2010 at The MITRE Corporation in Bedford, MA. The purpose of this event was to discuss, in technical detail, some of the more difficult issues facing Version 5 of the OVAL Language, and to explore new use cases for OVAL. By bringing together the lead proponents within the OVAL Community, the hope was to derive solutions that would benefit all parties, and continue the community-led development of the language. What follows is a detailed summary of the discussions from the event.

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

Table of Contents

Introduction 2

Attendee List 4

What Has Changed Since Last Year? 7

  Highlights 7

  Last Year's Topics 7

Discussion Topics 7

  Taming OVAL Results 7

    Subtopic 1: More Granular Evaluation Results 8

    Subtopic 2: Lightweight OVAL Results 11

    Subtopic 3: What is the cause? 14

  Optimizing Item Searches 17

    Background 17

    Current Optimization Capabilities 18

    Optimization Capabilities for Version 5.8 20

    Optimization Capabilities for Version 6.0 27

    Wrap-up Discussion 28

    Conclusion 30

  Least Version Principle for OVAL Content 30

    Background 30

    Proposals 31

    Discussion 31

    Conclusion 31

  Application Specific Detection 32

    Background 32

    Discussion 33

    Conclusion 34

  Adding Datatypes 34

Attendee List

Apple	-	Shawn Geddis
Arellia Corp	-	Kevin Stephens
Belarc	-	Richard Defuria
		Gary Newman
BigFix	-	Eric Walker
Booz Allen Hamilton	-	Ying Qi
Cimcor	-	Timothy Rodriguez
		Jeff Wozniak
Cisco Systems	-	Joseph Dallatore
		John Stuppi
		Seth Hanford
Dept of State	-	Randell Adkins
		Vu Nguyen
DISA	-	Andy Clifford
		Jim Govekar
		Jason Mackanick
		Joe Mazzarella
		Scott Moore
		Alan Peltzman
		Pete Schmidt
		Jordan Shuhart
		Jacquie Sipes
		Jamaal Spearman
		Ricki Vanetesse
DTCC	-	Aharon Chernin
EMC	-	Dan Reddy
Federal Reserve	-	Blake Thomas
	-	Doug Taylor
G2, Inc	-	Matthew Kerr
		George Saylor
		Shane Shaffer
IBM	-	Scott Moore
McAfee	-	Richard Whitehurst

MITRE	-	Jonathan Baker Sean Barnum Steve Boczenowski Andrew Buttner Maria Casipe Brant Cheikes Michael Chisholm Matthew Hansbury Daniel Haynes Jasen Jacobsen Mike Lah Daniel Mitchell Linda Morrison Lisa Nordman Mary Parmelee Emily Reid Charles Schmidt Matthew Wojcik Bryan Worrell John Wunder Margie Zuk
Modulo Security	-	Marlon Gaspar
NASA	-	Gary Gapinski
nCircle	-	Ian Turner
NIST	-	John Banghart Harold Booth David Waltermire Paul Cichonski
NSA	-	Mike Kinney Jim Ronayne Joseph Wolfkiel
RSA	-	Matthew Coles Dennis Moreau
SecPod Technologies	-	Chandrashekhar Basavanna
SRA	-	Wei Chen Deirdre Regan

Symantec	-	Jim Boyles Richard Freeman Jason Meinhart Craig Morea Dragos Prisaca Josh Turpin
TASC	-	Glenn Fournier Tom Mowbray
Telos	-	Sudhir Gandhe
Tenable	-	Mehul Revankar
ThreatGuard	-	Rob Hollis
Tripwire	-	John Wenning
Triumfant, Inc	-	Bill Goodrich
US Army	-	Gerald Crismon Vladimir Giszpenc
USAF	-	Kathleen Lynch
USDOT	-	Brendan Harris

What Has Changed Since Last Year?

The OVAL session started with a brief recap of what has changed in OVAL since last year's event and an overview of the outcomes of last year's discussion items.

Highlights

The following highlights were cited:

- Version 5.6 – Released September 11, 2009
- OVAL Adoption Program launched
- Version 5.7 – Released May 11, 2010
- Version 5.8 Planned – August 18, 2010
- OVAL Repository
  - 1525 new definitions
  - 4510 modified definitions
  - 7287 total definitions

Last Year's Topics

Last year's topics included the following items:

- Deprecation Policy Review
- Schematron Usage in OVAL
- Element Name Reconciliation
- xsd:choice Structure on Objects
- Supporting N-Tuples in OVAL
- Pattern Match on Enumerations
- Tests Reference Multiple States
- Introduce PCRE Based Pattern Matches
- Emerging Use Case: "OVAL for System Inventory?"

A brief summary of the status on each of these items was presented. Detailed minutes from the 2009 OVAL Developer Days discussion can be found on the OVAL web site at:

https://oval.mitre.org/oval/about/developer_days.html

Discussion Topics

Taming OVAL Results

This topic consisted of three closely related subtopics that together were aimed at allowing for smaller more useful results sets. This topic was discussed in response to community feedback and feature requests related to making OVAL Results more useful. The general feature requests have been:

- More granular evaluation results are needed to show why a definition evaluated as it did.
- Full OVAL Results do not scale well to enterprise.
- Include only the actionable information in OVAL Results.
- Highlight the hidden data in OVAL Results.

All of these requests come with the expectation that OVAL Results must maintain interoperability. Products must continue to be able to exchange OVAL Results.

Subtopic 1: More Granular Evaluation Results

This subtopic is based on a feature request received over the oval-developer-list. The request was summarized as “add capability to specify test result other than true or false”. The requestor would specifically like OVAL to support the following examples:

1. Verifying installed version of an application:
   - `true` – when application version 7 is installed in the system
   - `false` – when the version of application is not 7
   - `not applicable` – when the application is not installed
2. Verifying file permissions:
   - `true` – when the file exists and its access rights are properly configured
   - `false` – when the file exists and its access rights are not properly configured
   - `not applicable` – when the file does not exist

In short, this can be thought of as needing to report that the system is neither compliant nor noncompliant if the application or file does not exist.

Background

As background for this discussion the current OVAL Result values were presented and discussed. As of version 5.7 the following result values are defined in the OVAL Results schema:

- `true` – the characteristics being evaluated match the information represented in the system characteristic file.
- `false` – the characteristics being evaluated do not match the information represented in the system characteristic file.
- `unknown` – the characteristics being evaluated cannot be found in the system characteristic file.
- `error` – the characteristics being evaluated exist in the system characteristic file but there was an error either collecting information or in performing analysis.
- `not evaluated` – a choice was made not to evaluate the given definition or test.
- `not applicable` – the definition or test being evaluated is not valid on the given platform.

The specific meaning and utility of the ‘`not applicable`’ results was reviewed in some detail since it is important to note that the ‘not applicable’ result value already has a special meaning and it would not be appropriate to overload that meaning for another purpose. As noted in the OVAL Results schema documentation, "… a result value of 'not applicable' means that the definition or test being evaluated is **not valid on the given platform**. For example, trying to collect Linux RPM information on a Windows system. Another example would be in trying to collect RPM information on a linux system that does not have the RPM packaging system installed." As it is defined, the 'not applicable' result value allows content authors to combine criteria logically for multiple platforms. For example, a vulnerability definition can be written to include a <registry_test/> and the <rpminfo_test/> in order to check both Windows and Red Hat systems. In order to support this capability the 'not applicable' result value is not considered when determining aggregate results. For example, a 'false' result AND a 'not applicable' result will aggregate to a 'false' result. Similarly, a 'true' result AND a 'not applicable' result will aggregate to a 'true' result.

Another issue that prevents an OVAL Results consumer from processing a results document as it is and locating the desired information is the fact that some of the results information may be obscured once the result of a test is determined. Each test has a check attribute and a check_existence attribute. The check_existence attribute allows a content author to make an assertion about the number of Items that must exist on a system. The check attribute allow a content author to make an assertion about the number of Items on the system that must satisfy the referenced State conditions. When determining the result of a Test if the check_existence is satisfied (true) then the check is considered. Otherwise the Test result is the result of the check_existence. This evaluation process means that there is no way to differentiate a 'false' result due to failing the check_existence from a 'false' result due to failing the check.

Proposal 1: Add a Result Value

In this proposal the suggestion was made to add a new result value like: false_existence. This value could be used to record the fact that the check_existence evaluation failed. Under this proposal the not applicable result value could continue to be used as it is.

The initial examples could be reworked as follows with the proposed false_existence result value:

1. Verifying installed version of an application:
   - true – when application version 7 is installed in the system
   - false – when the version of application is not 7
   - false_existence – when the application is not installed
2. Verifying file permissions:
   - true – when the file exists and its access rights are properly configured
   - false – when the file exists and its access rights are not properly configured
   - false_existence – when the file does not exist

These specific simple examples would work with this new result value.

In considering this proposal it is important to note that as defined in OVAL version 5.7 a test essentially evaluates to a Boolean with a few other possible error like results. Adding in this new result value would be a move away from this Boolean result. Accordingly all the evaluation tables that specify how to combine each of the possible result values would need to be recreated. These evaluation tables must specify how to combine a '`false`' and a '`false_existence`' result and this decision will almost certainly not meet everyone's needs all the time.

Finally, it was noted that moving away from the Boolean result and adding a new result value might eventually lead us to an unwieldy proliferation of result values.

Proposal 2: Add an Attribute

In this proposal the suggestion was made to add a new attribute to the `oval-res:TestType` to record the result of evaluating the `check_existence` attribute. This new attribute would not be considered during Definition evaluation and would simply provide high level metadata to record one step in the evaluation process for a Test.

The initial examples could be reworked as follows with the proposed `existence_result` attribute:

1. Verifying installed version of an application:
    - `true` – when application version 7 is installed in the system
    - `false` – when the version of application is not 7
    - `existence_result='false'` – when the application is not installed
2. Verifying file permissions:
    - `true` – when the file exists and its access rights are properly configured
    - `false` – when the file exists and its access rights are not properly configured
    - `existence_result='false'` – when the file does not exist

In considering this proposal it is important to note that the evaluation process is not modified in any way by this new attribute. The attribute is simply metadata about that evaluation process. As such there is no need to alter any of the existing evaluation tables. This proposal also preserves the Boolean result values for a Test and Definition. However, this proposal requires extra work on the part of the OVAL Results consumer to process the content and determine if a given result was false due to an existence failure or a state failure.

Discussion

One participant pointed out that it is desirable for the tool or the higher level context to make the decision to evaluate a given OVAL Definition on a given system or not. This is commonly done in XCCDF with a platform check in a Rule or possibly even at the Group or Benchmark level.

In considering proposal 1 the following comments were made:

- How would this proposal handle sets of items and reporting partial existence of the needed items? The suggestion was made that another result value would be needed for this situation to record partial failure of the existence check.
- It is a slippery slope with a ternary state. This could lead to a large set of possible conditions.

- Perhaps we go the other way and not allow tools to ask this sort of two part question and instead simply create two different definitions: one definition to test for applicability and another to test for the configuration setting.
- Keeping the checks (OVAL Definitions) as granular as possible allows for flexibility and ensures that this problem can be supported in OVAL as long as there is a higher level context in which a determination about which checks to run can be made.

The discussion comments led to the group questioning whether this particular issue is really something that OVAL needs to solve. One could argue that OVAL has all the needed capability today and that there is not a strong need to push this capability down into OVAL.

An additional suggestion was made to consider adding a new Test or set of Tests like the `<unknown_test/>` that would evaluate to a specified result value whenever considered.

In considering proposal 2 the following comments were made:

- When a test fails the `existence_check`, the Items that are collected as a result of processing the Test's referenced Object are not evaluated. These items are all referenced in the `oval-res:TestType` with the `<tested_item/>` element. This element holds the id of an Item in the collected System Characteristics and the result of evaluating that Item. When the `existence_check` fails, the result record for each `<tested_item/>` is '`not evaluated`'. This information should be enough to convey that the `existence_check` failed. Essentially, OVAL already has information that would allow a tool to determine that a false result was due to a false `existence_check`.
- The proposed `existence_result` attribute could be thought of as adding in yet another type of result response and therefore is changing a long standing premise of OVAL that each Definition evaluates to a single result value.
- Adding some sort of third result will simply add confusion to result consumers.

Conclusion

Throughout the discussion there was a recurring theme of whether or not OVAL should address this feature request at all. In the end there was little to no support for making any change to OVAL to support this feature request. In response to this discussion the topic will be revisited on the oval-developer-list with the suggestion of closing out this feature request.

Subtopic 2: Lightweight OVAL Results

This subtopic is based on an open feature request to "add result directive to allow for lighter weight results with ability to track causal information". In this case the requested feature is actually a bit overloaded. There are really two distinct items to be discussed. First, adding additional directives to allow for more control of the content of an OVAL Result document. Second, highlight the causal information that is needed by result consumers to make easy use of an OVAL Results document. In this discussion the group focused on the first item. The second item was discussed in the "What is the cause?" subtopic.

Background

As background for this discussion the current OVAL Result directives were presented and discussed. OVAL Result directives can be thought of as a set of flags that describe the information included in an OVAL Results document.

As of version 5.7 the OVAL Result directives can be used to specify the level of detail included in an OVAL Results document as follows:

- Results may be included by result value. For example, an OVAL Results producer can exclude all OVAL Definitions with a result of 'not evaluated' or only include results for OVAL Definitions that had a result of 'true'.
- Either full system data and evaluation results or simply an OVAL Definition id and a result value many be included for each OVAL Result value.

OVAL Result directives are set by the result producer and included in the OVAL Result document. There is an assumption that the result producer is somehow configurable and capable of producing OVAL Result documents at varying levels of verbosity.

As OVAL Results are currently defined, there is a moderate level of flexibility in the allowed result outputs that can easily reduce the size of a result document by 20% or more in realistic usage scenarios.

Discussion

The discussion of this subtopic focused on exploring and better understanding the shortcomings of the version 5.7 OVAL Results directives. This discussion started by considering the following questions:

- Does anyone really support the directives?
    - SCAP 1.0 explicitly requires full OVAL Results.
    - Directives are not currently implemented in OVALDI**.**
- Do we need more options than thin and full?
    - Does the definition document need to be included?
        - If not, then how does a result consumer really know what was evaluated?
- Do we need to consider definition @class?
    - For example, include full results for true vulnerability definitions and thin results for all other true definitions.
- How do more options affect interoperability?
    - Are full results needed for generic content sharing?

During the discussion the following comments were made:

- Within a product I know what OVAL Definition document was used during evaluation. It would be nice not to require the source OVAL Definition document in the OVAL Results so that my clients could return OVAL Results without the source OVAL Definition document. From my server, I could then produce full OVAL Results including the source OVAL Definitions if needed.

- When the OVAL Results schema was created there was no concept of a results data stream. There did not used to be a benchmark with an id that served as a wrapper around a set of OVAL Results. At this point we may not need the full source OVAL Definition document.
- If I know an OVAL Definition id and I simply want the results back, I do not want the full definition I just requested back.
- We need to look at all the baggage that is included within the SCAP component standards. If there is simply too much baggage, then we will need to move on to other solutions.
- In XCCDF the original document is optional. A benchmark can be evaluated and then just the results can be provided without the full source XCCDF document.
- Adding more result options to OVAL simply adds complexity. We are currently guessing about future use cases of OVAL Results that we don't understand and trying to define formats that will meet those needs. Much of what is being discussed can be achieved through a simple transform of an OVAL Results document. Are we adding complexity to OVAL that is simply not needed?
- Should OVAL focus on the simpler end system assessment use case and allow others to develop solutions on top of OVAL that address exchanging result information on an enterprise scale?
- In SCAP 1.1 several different OVAL Result combinations will be included to allow for more compact OVAL Results. In SCAP 1.2 there is additional opportunity for improvement. NIST is looking into a more formal request/response format for SCAP 1.2 and additional OVAL Result granularity could be used.
- One vendor indicated that they simply make use of a transformation of a full OVAL Result document to create what they call a findings result.
- Requiring each vendor to write their own transformation will defeat tool interoperability so we really should look into how we can define a useful format in OVAL for conveying OVAL assessment results.
- When a definition is created, the author does not always know what is important to report.
- Could a simple hash or checksum of the source OVAL Definitions document be included rather than denormalizing all the result data by including a copy of the source document?
- Even when a compliance definition results in true people sometimes want to know the full details and what the relevant observed system value is.
- For result content should we extend beyond full and thin? Let's consider allowing for variations in between the two. For example, allow thin plus full definition criteria. On top of that allow full test results. Then on top of that allow full item data. In general, more result granularity will allow for tailoring by the evaluation requestor. In order to increase granularity we need to expose a schema for declaring what the results should look like. Consider adding an OVAL Results Directives schema for this purpose. Note that other standards could reuse that schema to indicate that a given set of OVAL Definitions should be evaluated and that a given OVAL Results format must be provided.

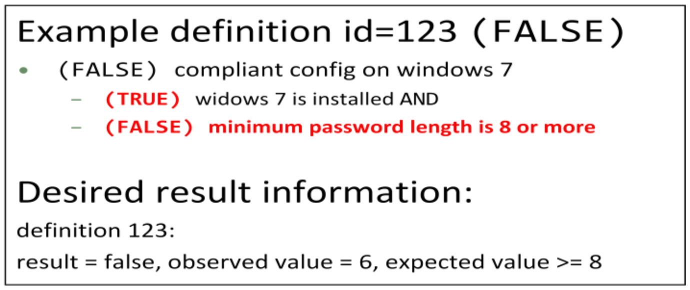
Conclusion
During the course of the discussion the following items were noted:

- OVAL Results Directives should allow for differentiation of OVAL Results content by class.
- OVAL Results should consider allowing the source OVAL Definition document to be optional. There might be a in between option here that would include simply the ids of the Definitions that were evaluated.
- OVAL should consider creating a lightweight schema for specifying the directives that must be used in reporting OVAL Results.

Throughout the discussion the point was made that much of what was being discussed in terms of controlling the content of a given OVAL Results document could be achieved by applying a XSL transform. The downside here is simply that the output of the transform needs to be standardized. There needs to be a standard result format of varying levels of verbosity that can represent an OVAL Definition evaluation. These topics will be further discussed on the oval-developer-list.

Subtopic 3: What is the cause?
This subtopic is based on an open feature request to "add result directive to allow for lighter weight results with ability to track causal information". In this case the requested feature is actually a bit overloaded. There are really two distinct items to be discussed. First, adding additional directives to allow for more control of the content of an OVAL Result document. Second, highlight the causal information that is needed by result consumers to more easily make use of an OVAL Results document. In this discussion the group focused on the second item.

Background
Full OVAL Results likely contain the data needed to determine the observed settings on a system that led to a compliance failure or vulnerability report. The following simple example was discussed:


Example definition id=123 (FALSE)

- (FALSE) compliant config on windows 7
  - (TRUE) widows 7 is installed AND
  - (FALSE) **minimum password length is 8 or more**

Desired result information:

definition 123:
result = false, observed value = 6, expected value >= 8

In the example above, the interesting data is that the Test referenced by the minimum password length criterion evaluated to false. This seems fairly straight forward here, but as the criteria of a Definition get more complex it becomes increasingly difficult to identify the important pieces of data in an OVAL Results document.

As further background it was pointed out that generally a Definition author knows what information is of the most interest when creating a new Definition. Most OVAL Definitions include a fair amount of preconditions and other boiler plate checks that are generally not that interesting to report to an end user. With this in mind, it was noted that the most interesting data is generally examined with a State entity. Within a single OVAL Definition there can be many States and each State can have many entities. A State is used to make an assertion about an Item. Items have entities that are evaluated by State entities. Item entities may have a multiplicity of 0 – N.

Proposal: Add a `report_value` Attribute

If the assumption can be made that a definition author knows which items are most interesting to report then why not allow the author to highlight or flag those items?

This proposal suggests adding an optional `report_value` attribute to all State entities. This Boolean attribute would default to false and when true would indicate that the system value(s) should be highlighted or reported in the OVAL Results document. The current behavior would remain unchanged when the `report_value` attribute is false.

The second component to this proposals is the addition of an <oval-res:observed_value/> element in the existing <oval-res:tested_item/> element. A <oval-res:tested_item/> could have an unbounded set of child <oval-res:observed_value/> elements. Each <oval-res:observed_value/> would hold the name of the reported State entity, the data type observed on the system, and the observed value.

In considering this proposal it is important to note that this is not a complete solution. This solution would work quite well in simple cases, but more complex definition criteria would remain difficult to process and accurately report the underlying cause of a given Definition result. If accepted this proposal would also result in even larger full OVAL Result documents as it would essentially duplicate some of the data that is somewhat buried in an OVAL Results document.

Discussion

The discussion on this subtopic first focused on understanding how vendors are solving this problem today. There are products that display the desired information already. How are vendors doing this and can we develop a common solution? In response to these questions the following comments were made:

- One individual pointed out that their organization decided to create its own results format to solve this problem. This decision was motivated by the size of the typical OVAL Results file, the fact that they did not want to have to include the source OVAL Definitions that were evaluated, and the fact that they wanted an easier way to look up and report observed values. It was noted that the current structure of OVAL System Characteristics makes it fairly challenging to look up system objects such as file or registry keys, especially when variables are used.

In discussing the “Add a `report_value` Attribute” proposal the following comments were made:

- We might want to consider essentially reporting all values and then allow a content author to deemphasize a given value. This would essentially be the opposite of the proposal. Rather than highlight data, allow an author to specifically indicate that a given value should not be reported.
- There are cases in which for a reporting purpose we might only care about a given value if some other Test is true. These dependencies will remain challenging with this proposal.
- Should there be a criteria or criterion attribute to allow for highlighting and reporting at that level? Would this help in complex cases?
- When performing an existence check a State is not used. How would we handle this if we are not using a State? Is this creating an inconsistency in where result data is held in an OVAL Result document?
  Response: When performing an existence check, collected Items are recorded with `<oval-res:tested_item/>` element in the `<oval-res:test/>`, and each `<oval-res:tested_item/>` element has its result set to '`not evaluated`'.
- Adding this new structure seems counter to the other OVAL Results discussion topics where we were considering reducing the size of an OVAL Results document.
- Hypothetically, a directive could be added to control whether or not this proposed result information is included in an OVAL Results documents or not.
- When preparing for an assessment it will become increasingly import to understand the possible directives and their meaning.
- End users are not generating content. Typically they get the content from some authority. How with this allow an end user to more easily see the reported data?
  Response: The content authorities will need to annotate their content to allow end users to benefit from this capability.
- It is often the case that the full Item data is interesting, not simply the value. For example, in the case of a Windows registry key I want to know the full hive, key, name, and value combination.
- Additionally, highlighting the observed value on a system and reporting that to an end user may not be very useful because the value might not be intended for human consumption. For example, a value might be a bit mask where each bit might have a significant meaning and the end user cannot be expected to interpret those bits. The end user needs the more human readable value that is often displayed in tool UIs.
- When States are reused adding a `report_value` attribute will mean that a value is reported all the time, even when it may not be appropriate.
- Another option would be to add an id to the items that should be reported. At the Definition level the id could be referenced in a way that would indicate that the item should be reported. This might allow a bit more granularity in determining what to reporting.
- Would it be simpler to just provide a lightweight framework and then let content authors include their own style sheet of other set of rules for what to report?

Finally, the proposed OVAL Reporting schema was reviewed to highlight how the proposals in this session differ from this new schema. More information about the OVAL Report schema can be found in the oval-developer-list archives here:

> http://making-security-measurable.1364806.n2.nabble.com/OVAL-Reports-Schema-tp4904766p4904766.html

In summary, the OVAL Reporting schema reuses existing OVAL Objects, States, and Variables to specify a set of information to collect from an end system. The formatting of this information is left entirely up to the author. The report author supplies a XSL template that allows for output as desired. It is important to note the OVAL Reporting is not about making an assertion about a machine state, it only defines a framework for collecting information and then formatting it.

Conclusion

This subtopic needs additional discussion over the oval-developer-list. There are a number of issues that need to be considered.

Optimizing Item Searches

In the OVAL Language, an object is used to specify which items on the system to collect. The collected set of items is determined by specifying operations and values for the entities that describe the item in the object. Depending on the object, and the state of the system, it is possible that a very large set of items will be collected. As a result, tools have to process this data and write it to OVAL System Characteristics and OVAL Results documents which takes both time and system resources. It is often the case that many of the collected items are irrelevant to the assertion being made and the ability to specify more specific sets of items will greatly improve the performance of tools.

This discussion focused on determining whether this is a problem that needs to be addressed, and if so, how and when should it be addressed. The discussion began with an overview of the problem and its influence on tools and the artifacts that they produce. This was followed by a review of the current capabilities in the OVAL Language for item optimization, proposals for adding optimization techniques in Version 5.8, as well as considerations regarding item optimization in Version 6.0. Lastly, the discussion ended with an opportunity to discuss any outstanding topics and summarize the thoughts of the community.

Background

The original discussion that initiated this topic is on the oval-developer-list and can be found at the following link.

http://making-security-measurable.1364806.n2.nabble.com/oval-object-criteria-tp4931198.html

The primary example that exemplifies this problem is the need to find all world-writable files on a system. In the OVAL Language, this is be accomplished by first creating a `<file_test/>` that will collect every file on the system.

```
<file_object id="oval:sample:obj:1">
    <path operation="pattern match">.*</path>
    <filename operation="pattern match">.*</filename>
</file_object>
```

Next, a `<file_test/>` that represents a world-writable file will need to be specified.

```
<file_state id="oval:sample:ste:1">
    <owrite datatype="boolean">1</owrite>
</file_state>
```

Lastly, a `<file_test/>` will need to be specified to compare each collected `<file_item/>` against the `<file_state/>`. Then based on the `check` and `check_existence` attributes, an assertion will be made about the state of the system.

```
<file_test id="oval:sample:tst:1">
    <object object_ref="oval:sample:obj:1"/>
    <state state_ref="oval:sample:ste:1"/>
</file_test>
```

The problem that arises out of this example is that the `<file_object/>` will collect a `<file_item/>` for **every** file on the system. Unfortunately, many of the items collected are irrelevant to the assertion of whether or not world-writable files exist on the system. For example, on the sample system, only 1,254 of the 148,745 files are world writable. The collection of irrelevant items also promotes unnecessarily large OVAL System Characteristics and OVAL Results files that are not only difficult to read, but require extra resources to process. For example, the sample system produced a 160MB OVAL System Characteristics file for every `<file_item/>` as opposed to a 1.24MB OVAL System Characteristics file for every world-writable `<file_item/>`. As a result, solutions to this problem will target the reduction of items collected.

Current Optimization Capabilities

In the OVAL Language, for each collected object in an OVAL System Characteristics file a `flag` attribute provides information regarding the outcome of a collected object. For example, if every item that matches the object is collected and written to the OVAL System characteristics file, the collected object will have a `flag` attribute of `true`. Or, if an attempt to collect an `<rpminfo_object/>` on a Windows system, the collected object will have a `flag` attribute of `not applicable` because a Windows system does not use RPMs. The OVAL Language currently supports item optimization capabilities with the `flag` attribute value of `incomplete`. The OVAL Language defines the `flag` attribute value of `incomplete` as follows.

A flag of 'incomplete' indicates that a matching item exists on the system, but only some of the matching items have been identified and are represented in the system characteristics file. It is unknown if additional matching items also exist…

The `flag` attribute value of `incomplete` allows for optimization because tools can write a subset of the collected items, which match the object, to the OVAL System Characteristics file. To highlight how the `flag` attribute value of `incomplete` can be used to optimize item searches an example is provided below.

```
<file_test id="oval:sample:tst:1" check_existence="none_exist">
    <object object_ref="oval:sample:obj:1"/>
</file_test>

<file_object id="oval:sample:obj:1">
    <oval-def:set>
        <oval-def:object_reference>oval:sample:obj:2</oval-
def:object_reference>
        <oval-def:filter action="include">oval:sample:ste:1</oval-
def:filter>
    </oval-def:set>
</file_object>

<file_object id="oval:sample:obj:2">
    <path operation="pattern match">.*</path>
    <filename operation="pattern match">.*</filename>
</file_object>

<file_state id="oval:sample:ste:1">
    <owrite datatype="boolean">1</owrite>
</file_state>
```

This example will evaluate to true if there are no files on the system that are world writable. The `<file_object id="oval:sample:obj:1"/>` has a `<set/>` that references `<file_object id="oval:sample:obj:2"/>` which will collect every file on the system. The set also contains a `<filter/>` that will include any `<file_item/>` that matches `<file_state id="oval:sample:ste:1"/>` which characterizes world-writable files. Lastly, the `<file_test id="oval:sample:tst:1" check_existence="none_exist"/>` references `<file_object id="oval:sample:obj:1"/>`, and with the specification of the `check_existence` attribute value `none_exist`, the test will evaluate to `true` if a matching `<file_item/>` is not found on the system.

This example can be optimized by only collecting a `<file_item/>` if it satisfies the `<filter/>` specified in the `<set/>`. Essentially, this means that only the world-writable files will be collected rather than every file that exists on the system. The second way that this example could be optimized is based off the `check_existence` attribute. In this example, the `<file_test/>` will evaluate to `true` if no world-writable files exist on the system. If at least one `<file_item/>` exists on the system, the test will evaluate to `false`. As a result, this can be optimized by short-circuiting the collection process when the first world-writable `<file_item/>` is discovered. This is possible because the result of the assertion can be determined with a single `<file_item/>`. Both of these optimizations are valid as long as the collected objects are given a `flag` attribute value of `incomplete`.

However, it is important to note that with a `flag` attribute value of `incomplete`, on a collected object, a result of `true` or `false` cannot always be determined. According to the OVAL Language, with a `flag` attribute value of `incomplete`, it is unknown if additional matching items exist on the system and, as a result, the test must evaluate to `unknown`. This means that certain assertions such as `check="all"` are not possible because it cannot be said that all items match a specific state if it is unknown if additional items also exist.

During the discussion of the optimization capabilities using the `flag` attribute value of `incomplete`, the following questions and comments were made:

- Of what entity is the `flag` an attribute?
  Response: The `flag` is an attribute of collected objects in the OVAL System Characteristics document meaning it could be any object.
- What does one do when an object is used in more than one test with different determination characteristics?
  Response: You need to be aware of where this optimization is applied. In certain situations, this optimization could evaluate to a result of `unknown`.
  Response: In the case where the existence check is no matching files exist on the system, and your object is collecting every file on the system, you could optimize by stopping the data collection process once an item has been found and then setting the object's `flag` attribute, in the system characteristics file, to `incomplete`. If you try and reuse the object for an existence check, other than one where no matching items exist on the system, it would evaluate to a result of `unknown`.
- Has anyone attempted to apply this optimization in their tool?
  Response: One vendor indicated that they applied the optimization for one test and that the documentation provided the necessary information required to do so.
  Response: It makes more sense to apply this optimization technique in situations where the search space is large.
- A concern was raised that this optimization technique may not be as helpful in situations where the item collection is not performed at runtime such as when items are retrieved from a periodically updated database.

Optimization Capabilities for Version 5.8

With the release of Version 5.8 of the OVAL Language approaching, it is important to consider what optimization capabilities can be included if desired.

Proposal 1: Add a filter element to objects

The first proposal for optimizing item searches in Version 5.8 is to allow for 0-n `<filter/>` elements to all objects in the OVAL Language. Each `<filter/>` element will allow for the specification of an `action` attribute which specifies whether to include or exclude items that match the state specified in the `<filter/>` element. It is also important to note that with this change a `complete` object will be the set of items after each `<filter/>` has been applied to the items collected by the object's required entities. An example of this proposal can be seen below.

```
<file_object id="oval:sample:obj:1">
    <path operation="pattern match">.*</path>
    <filename operation="pattern match">.*</filename>
    <filter action="include">oval:sample:ste:1</filter>
</file_object>

<file_state id="oval:sample:ste:1">
    <owrite datatype="boolean">1</owrite>
</file_state>
```

The primary advantage to this solution is that the filtering capability is a well-known concept that was introduced to the `<set/>` construct in Version 5.0 of the OVAL Language. The implementation of this solution may also be very similar to applying `<filter/>` elements in a `<set/>` and could potentially allow for code reuse and abstraction. Since the solution would be applied across all objects, it would solve the problem of not being able to fine-tune item collection everywhere. In addition, the ability to specify an unbounded number of filters will provide a content author with more flexibility in determining which items should be collected as they can include or exclude as many states as needed. Lastly, the use of states provides a mechanism to specify multiple values using variables as well as specify ranges using different operations.

The most notable drawback to this solution is that the filtering of items does not make sense for every object in the OVAL Language. For example, objects that collect a single item (e.g. `<passwordpolicy_object/>`) and objects that collect small sets of items (e.g. `<interface_object/>`). In both of these examples, the ability to optimize item searches will not result in a significant reduction of the output file size or the resources required by tools to process the data. However, this drawback is mitigated because it is not required for a content author to include `<filter/>` elements in the object. Another disadvantage to this solution is that, while the `<filter/>` element allows additional flexibility, it also increases the risk of a content author specifying `<filter/>` elements that could cancel each other or even result in an object that does not collect any items (i.e. a `flag` attribute value of `does not exist`). Lastly, the introduction of an additional element to objects increases the complexity for content authors in that they need to determine when it is, and is not, appropriate to use this new capability as well as the risk, described above, that is associated with it.

During the discussion of the Version 5.8 proposal to add a filter element to all objects in the OVAL Language, the following question was raised:

- Can't you already do that with sets today?
  Response: The difference here is that when you use an object in a `<set/>` and apply a `<filter/>` to it, the object is still going to go and collect all of those items and they will be

written to the OVAL System Characteristics file. This would allow you to remove them before they are written to the OVAL System Characteristics file.

Proposal 2: Add a filter and action behavior

Another option for optimizing item searches is to add `filter` and `action` attribute behaviors to the `<behaviors/>` element. Like the `<filter/>` element, the `<behaviors/>` element was introduced in Version 5.0 of the OVAL Language and is a well-known concept. This also aligns with the notion that the `<behaviors/>` element is a mechanism to provide a more granular definition of an object. The `filter` attribute will allow also for the specification of a state that will represent the items to include or exclude, as specified by the `action` attribute, from the set of items specified by the object. Again, this change means a `complete` object will be the set of items after the `<behaviors/>` element has been applied to the items collected by the object's required entities. An example of this proposal can be seen below.

```
<file_object id="oval:sample:obj:1">
    <behaviors filter="oval:sample:ste:1" action="include"/>
    <path operation="pattern match">.*</path>
    <filename operation="pattern match">.*</filename>
</file_object>

<file_state id="oval:sample:ste:1">
    <owrite datatype="boolean">1</owrite>
</file_state>
```

The key advantage to this solution is that specific objects, those that collect large sets of items, can be targeted as opposed to applying a solution to every object even when it may not make sense. Also, since this is conceptually similar to applying the `<filter/>` element in the `<set/>` construct, it may present the possibility for code re-use and abstraction.

While this solution still uses states to specify the items to filter, it only allows for the use of one state which means there is less flexibility to fine-tune the collected set of items. At the same time, this may be beneficial as it will reduce that likelihood that content authors will create content that results in a collected object having a `flag` attribute value of `does not exist`.

During the discussion of the Version 5.8 proposal to add a filter and action attribute behavior to the behaviors element, the following question and comment were made:

- It appears to me that Option 2 is not as useful as Option 1 because it is restricted to only those objects that one thought were worthy of having a filter. Behaviors have always been an odd construct within the language and a straight forward filter that could be applied to some or even all objects would be more useful than behaviors that were only found in certain objects.
Response: An object may have a limited data set in most cases, but in some cases it may have a large data set. Thus, it makes sense to apply the filtering capability to every object such that it is available if needed.
- What Boolean logic is available for combining filters (AND, OR, NOT, etc.)?

Response: If you have multiple filters, each filter will be applied sequentially to the set of collected items.
Response: Is there set theory available to apply the union, intersection, etc. operations?
Response: When you use sets in the OVAL Language, it applies all of the filters prior to the set operations (union, intersection, etc.).

Proposal 3: Add entity behaviors

A third proposal for optimizing item searches is to add entity behaviors to the `<behaviors/>` element that represent the state entities for the respective object. An example of this proposal can be seen below.

```
<file_object id="oval:sample:obj:1">
    <behaviors oread="1" owrite="1" oexec="1"…/>
    <path operation="pattern match">.*</path>
    <filename operation="pattern match">.*</filename>
</file_object>
```

The key advantage of this proposal is that the `<behaviors/>` element is well-known and conceptually the addition search controls, as behaviors, aligns with the notion of specifying a more granular definition of an object. In addition, this proposal will only target the objects where there is a need to actually optimize item searches (i.e. those objects that have the potential to collect large sets of items).

Out of the proposals, this is by far the most restrictive proposal in that it does not use states to specify the resulting set of collected items. As a result, the functionality of specifying ranges and multiple values will not be possible. It will only be possible to say that an item matches if its values align with those specified in the `<behaviors/>` element. In addition, since each set of behaviors will be different, the implementation will not lend itself to a clean and general solution. Lastly, this proposal will scale poorly. When specifying behaviors for an object, a decision will need to be made as to which state entities should be added. If every entity is added, the list of behaviors can become extremely large. For example, the `<file_state/>` has twenty-three entities!

During the discussion of the Version 5.8 proposal to add entity behaviors to the behaviors element, the following comment was made:

- The benefit of an approach like this or Option 2 is that you can say the things that people care about are a particular attribute, flag, etc. It is a broadening of the behavior notion whereas Option 1 is really taking states and generically applying a filter notion across everything. This approach would allow you to leverage your investment in state analysis in your interpreter engine. If you don't have the investment or your design is different, this option, will allow you to say files are a problem so we are going to provide a targeted fix for files.

Why Can't We Just Add Additional Entities to Objects?

As outlined in the oval-developer-list discussion, there is a desire to add additional entities to objects that will allow for a more granular definition of the set of items to collect. An example of this is provided below.

```
<file_object id="oval:sample:obj:1" >
    <path operation="pattern match">.*</path>
    <filename operation="pattern match">.*</filename>
    <owrite datatype="boolean">1</owrite>
</file_object>
```

The addition of entities to objects can be accomplished in two ways. The first way is to make the additional entities "required" which means they **must** be specified in the object. Unfortunately, this is not possible as it would invalidate existing content. However, this could be avoided by deprecating existing tests, objects, and states and re-creating new ones that include the additional entities. Unfortunately, this will result in the component schemas doubling in size. Lastly, this option introduces the problem of how to deal with entities where the value does not matter. For example, to specify that the value of the `<owrite/>` does not matter, it would require that the `<owrite/>` entity reference a `<constant_variable/>` that contains the values "0" and "1". This is not desirable. All of these issues can be resolved by introducing the new entities as "optional" meaning that the additional entities are **not** required to be specified in the object.

While the addition of entities, in objects, is technically possible in Version 5.8, it would represent a significant change in the OVAL Language. First, it will change the traditional notion of what an object is. An object is the minimum set of required entities necessary to identify a set of items on the system. The first option will break the notion of an object being the minimal set of entities required to identify an item and the second option will break that same notion as well as the notion that the entities in objects are to be required. Both options will also require significant changes to the schema and that tools support the additional functionality. The community was then asked to consider if this is something that should be done in a minor release?

During the discussion of the issue of why we can't just add entities to objects, the following comments were made:

- Could you just make the object entities nillable? That is the standard procedure for fields that we don't care about?
  Response: It is really a different concept. Nillable object entities are reserved for entities that form a hierarchical structure and it is necessary to ignore the lower level entities in favor of only collecting those that are higher up. For example, with the `<registry_object/>`, the name entity could be set to nil resulting in the collection of only the hive and key. You could come up with a definition for do not collect `<owrite/>`.
- Why are you suggesting that we make the object entities required?
  Response: We are just considering the different options.
- If you are identifying a set, you are still identifying the minimum set of attributes required to define the set.
  Response: By specifying the additional entities you are specifying a set.
  Response: Think of the path and filename as a composite ID for a single file on a system. Across all objects, we have been consistent in specifying how you identify a file, registry key, or RPM.

Response: You can already use patterns in file to get a set. As soon you start dealing with sets, you need a filtering capability to identify the set accurately.

- How would we handle more complex logical filtering like world read and world write or user read and user write?
  Response: You would want a container to support that Boolean logic.
  Response: A state does that. If you look at the `<filter/>` option, a state has the child entities world read and world write or user read and user write. You would need one filter for world read and world write and another for user read and user write.
  Response: What if you wanted an OR relationship (one or the other, but not necessarily both)?
  Response: You would need an additional construct to specify the Boolean relationship between the filters.
  Response: You could still do that with sets.
- A question about Option 1, it does not seem like you need the path and filename in the object? A state contains that information. You could either declare a filepath, a path and filename, or a `<filter/>`. It is really a choice of those three.
  Response: It would be a big change in how some tools process content if an empty file object indicated that all files on the system should be collected.
  Response: It is a choice of one of those three. Either it is required to have a path and filename, a filepath, or a `<filter/>`. Any one of those three options would be sufficient and would not invalidate the current process.
  Response: That would be a very different way of using states.
  Response: I do not see why you need to restrict it that way. If you want to look for world readable files in a certain directory you need to specify the path.
  Response: You are not losing that capability, by not specifying the path, as it is available in the state.
- I think Option 1 is the most desirable of the three options. First, it is conceptually the closest to how people think about calling the potential target objects out by a `<filter/>` mechanism. Second, it is directly symmetrical with what one can say about an object. All of the entities found, within a state, can be used to direct the object collection phase. Lastly, in the absence of a way to exercise of Boolean logic, you are restricted to a logical AND or a logical OR. One could choose both, but I strongly suggest that we surround it with something that allows the `<filter/>` elements to be combined together in a deliberate fashion as opposed to a single fashion.
- A benefit of Option 1 is that it is going to promote re-use. Rather than building multiple file objects, you could build a single `<file_object/>` and potentially filter it in different ways.
  Response: It does not require a pragma that says the optimization, which is not explicitly specified in the OVAL Language, might take place regardless. It is a way to build the capability into the OVAL Language as part of the Item collection process.
  Response: You would be able to fine-tune exactly what you are looking for.
- Would it be a big leap to have tool vendors support additional object entities?
  Response: One vendor reported that supporting additional object entities would not be difficult.
- I raised a question on the oval-developer-list regarding the fear of versioning. If a major version is required, what exactly is the problem with versioning?

Response: I would say the biggest challenge right now is the maintenance tail we have. We have to maintain the Version 5 release which is driven by the fact that many of you in the room have products and tools that support Version 5 and are depending on it working as well as having incremental advances such that your tool can continue to process the latest Microsoft security advisories. I think it comes down to finding a way to get the Version 5 line relatively stable so that we are only doing minor tweaks. At which point, we can focus primarily on Version 6. It also means that we will stop working on Version 5.x which is hard to do.
Response: I do not really understand the logic behind that. It seems that we have disbanded the notion of breaking changes, but breaking changes were the thing that would have precipitated a numeric major release. Outside of that, I am not sure that would. Do any vendors have thoughts on this?
Response: Is your point that most of the changes that we have been talking about can be done in such a way that it wouldn't break?
Response: Yes.
Response: My counter argument is that it seems that most vendors care about the SCAP version and those are specified in 5.3, 5.4, 5.6, and 5.8. If instead of 5.8, it happened to be 6.0, it would not make that much of a difference in the sense that you would have to move from 5.6 to the next version. I do not see that as a huge difference. I agree with the backwards breaking reason for changing numbers, however, I don't understand why we are not doing it.
Response: I would love rev more frequently and have minor versions more frequently such that we can fix issues in the language that needs to be cleaned up or fixed. We talked about this in the past and the issue is that we still have to maintain the Version 5 line for some period. I do not see how we can start breaking all of the content that is out there. I would like to think that there is a split in the community between those who support SCAP, and are after SCAP validated, and those who are using OVAL because it solves the problem for their product and are not necessarily interested in SCAP or SCAP validation.
Response: Content is driving the concerns about doing major revs because major revs would exist in a namespace and would likely invalidate the OVAL content from a schema validation perspective. This puts us in a position where we will have to maintain two sets of content for some period; potentially two to three years. While there is not a lot of cost in doing these types of revs, there are many hidden costs when it comes to content that is broken from a backwards compatibility perspective. This is one of the reasons why there has been a historic concern regarding a major versioning change.
Response: You use the word broken, but is it broken simply because it no longer adheres to a new schema?
Response: Yes.
Response: Assuming that one were to carefully devise changes to the OVAL Language such that they did not invalidate the semantic content of the older OVAL content, providing an XSL transform would allow one to move the older content to a newer version of the language and would obviate some of the concerns that people would have with the adoption of newer easier to use schemata.

Response: The issue is if we wanted to come up with a major version, an XSL transform to auto transform the content forward would help. However, I have heard from vendors that there is a cost associated with re-queuing the content and, if something does not work, the customers will complain.
Response: That is good if you embed the XSLT in the engine.
Response: Why even transform it, the content identifies schema.
Response: To make your product support both versions.
Response: It is an ongoing maintenance problem. There may be a desire to upgrade to newest version of OVAL and having a path forward through a transform or upgrade process is desirable.
Response: We will talk about content maintenance in more detail during the next session.
Response: If we are talking about revving more frequently, there are many hidden costs that we should be thinking about such as updating documentation, writing reference implementations, and the time that it takes the community to review and implement the schemas to ensure there are no problems. All of these things have to be repeated every time we do a rev. The more frequently that we rev, the more overhead we experience.
Response: Vendors have the same issue from a QA perspective. The features we have are going to be cumulative, but if we have to do three releases to get there, it is going to cost much more. In addition, I don't know how frequently our customer base can even upgrade their products.
Response: Determining what the appropriate iteration level should be for these types of efforts is a complex problem. There is a great deal of impact one way or another.

- What is the immediacy of the need that prompted these suggested changes to the OVAL Language? If it is an immediate need, due to very large result sets, this could be added as an optional item. If people did not implement it, they would simply carry on by ignoring that particular thing. That could be a caveat of some vendors' implementations. From a technical perspective, I regard the application of a filter, at the object collection time, as almost directly symmetric with the analysis of collected objects from the standpoint of applying a test or criterion. I don't think this would be particularly disruptive to most implementations of OVAL and it would be beneficial because it would allow people to keep their result sets down as long as they were aware that such a feature existed.

Optimization Capabilities for Version 6.0

Unlike adding optimization capabilities in Version 5.8, Version 6.0 is not restricted by the requirement to maintain backwards compatibility as defined in the OVAL Language Versioning Methodology which can be found at the following link:

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

As a result, there is a chance to learn from our experiences with the language over the years and improve and simplify the language at the same time.

To promote discussion, the community was asked to consider the impact of making all entities searchable in the context of Version 6.0.

Impact of making all entities searchable

- Do we really need states?
    - What about filters and ranges?
    - What would a test look like?
- Could results be based on the existence of items?
- Do we need object entities to be required?

During the discussion of the issue of looking into Version 6.0, the following questions and comments were made:

- Does anyone have any thoughts on getting rid of states?
  Response: We talked about dual role of checks against states. It does not only check that you match a state but also the existence or non-existence of a particular item. If you are not searching for a specific thing and then comparing against state, you lose that ability to differentiate between whether or not something exists and whether or not it is the particular configuration that you care about.
  Response: You would only be collecting the particular things that you cared about. If you found an item, you would know whether or not it existed and if it matched.
  Response: But, I would not get the password length is 12. If I filter from the very beginning, for the length is 12, I will not get the item that says the length was 8 instead of 12.
- I am a big proponent of not making backwards breaking changes if we do not have to. I don't see any of these as required.
  Response: These are not required, just things to think about.
  Response: I tend to agree. It doesn't seem that any of these backwards breaking changes would advance the language.

Wrap-up Discussion

To conclude this discussion the group was asked the following questions:

- Are there other ways to optimize item searches?
- Is this a problem we would like to address?
    - When would we like to make a change?
    - Do we need item optimization for every object?
    - How flexible does a solution need to be?
    - Will solutions be feasible to implement?
- Other questions, concerns, comments

In response, the following comments were made:

- Does anything stick out as other ways to optimize item searches that we did not talk about today? Any different ideas?
  Response: It would be useful, if we are to do optimizations, to have an explicit or implicit ordering of things that are evaluated. The best you could do, if you come up with one or more optimization techniques, is to do a partial ordering of all of the tests or the object filtering. It

would be possible to put an optimization into an OVAL interpreter if you said that you would always restrict yourself to objects that met or did not meet a particular test implicitly without saying it explicitly. For example, if the fundamental test in question was that all files on a system exhibit a certain state, there is no reason that you would have to slavishly collect all of those objects and then subject them to the test if you simply folded the test into the object collection phase.

Response: That is essentially what the filter is doing.

Response: But, it would not, it is not necessarily that explicit if you were to just go ahead and do it implicitly. You would not get a lot of opposition from people that didn't want to see the 15 or 30 thousand files present on the system for each and every object.

Response: But, in doing that, you'll lose the ability to separate the object collection from the object evaluation.

Response: Exactly.

Response: That is an option that one tool could make.

Response: It seems you also lose the location of the specification of where the test evaluation actually occurs. You would have to put an object in the criterion.

Response: The OVAL Language is a functional specification and not a procedural specification. There is nothing that says that the objects must be collected. One could easily traverse an OVAL Definition document; perform each criterion one by one, evaluate the underlying tests, and come up with the same result. It's not necessary to postulate the existence of a large set of collected objects and the like.

Response: Right, but even in that case, you would still need to have a test that references that object or you would never get to the point where you can evaluate it.

Response: I understand that. It is not necessary to postulate the existence of an object in the collection phase followed by the evaluation of state criteria which means that there's no reason not to combine them. There is no language restriction that says one cannot combine the object collection phase with the object evaluation phase.

Response: A concern was raised that, downstream, if someone takes the results from a tool that has done that optimization and has not collected all of the items because they realized that they only needed 10% of the items. If you tried to walk that results document, you would see the test passed the check which said `check_existence = all`. This says that all files exist and even though you have not actually recorded all in the document. As a result, there would be inconsistencies which we would need to figure out how to handle.

Response: That's exactly what some people are asking for as described earlier in the afternoon.

- Short circuiting is another optimization. It does cause problems when you get to the remediation phase except in the case of the precondition. In that case, because OVAL does not have a precondition, you are evaluating the criteria and then you have to evaluate the rest and it does not have value there. In other cases, for remediation purposes, it does make sense to evaluate it. A way to short-circuit only sometimes would be beneficial, but I do not have an answer right now.

- Do we need to do something in Version 5.8 as far as adding a filter element? Is that something that's necessary to do?
  Response: The community stated that there was a need to add a filter element in Version 5.8.
  Response: A concern was raised that if you do not do something, as general as the filter mechanism, you increase the idiosyncratic nature of the language. It becomes much harder for people to understand what they are supposed to be doing because on one test they may be able to use a behavior and others they may not whereas the general solution of filtering at object collection is far easier to understand and more closely approximates the real word thinking of someone crafting a particular test.
  Response: It does have some potential impact on being able to report on the exceptions. If reporting on the exceptions is important, you need to make sure that the objects that are the exception aren't included in the results.
  Response: One can still filter and retrieve all world writable files without necessarily producing the list of all non-world-writable files.

Conclusion
At the end of the discussion, it was concluded that the need to optimize item searches is necessary and should be addressed in Version 5.8 of the OVAL Language. Out of the proposals to optimize item searches in Version 5.8, the first proposal of adding an unbounded `<filter/>` element, to all objects, received strong support from the community. However, it was also made clear that there is a need to discuss and provide a mechanism for logically combining `<filter/>` elements in an explicit manner.

Least Version Principle for OVAL Content

Background
The OVAL Repository contains content contributed by members of the community, and provides that content for tool and community consumption. The discussion on the versioning policy started with an overview of the current policy.

Currently, the OVAL Repository only serves one version of the content, in the latest version of the language. With every new release of the OVAL Language, the content in the repository is moved forward as well. No actual changes are made to the content, and deprecated language constructs are not removed. This policy is easy to maintain, and with one version of content, it encourages tool compatibility with the latest version of the language. The downside is that there is no history of an OVAL Definition captured in the repository, and the burden is placed on the user or tool to figure out how to handle content with a later version number.

There are several key points to consider when evaluating any changes to the versioning policy:

- How difficult will it be to keep the repository up to date with accurate versions?
- Will there be a significant investment required in tools and process to handle the new policy?
- Will there be any burden placed on content creators?

One of the primary goals of the OVAL Repository is to make content creation as easy as possible, to encourage a large content submitting community.

- How will the new policy affect tools and backwards compatibility?
- Will a change affect vendor tools, the amount of time it takes to process content submissions, or the performance of the OVAL Repository site?

The OVAL Repository also serves as a model for other content repositories, so any proposed changes should be considered from the perspective of what a general OVAL repository should support.

Proposals

The proposal presented to the group was to determine the least version for each item (definition, test, object, state) when importing to the repository. There were two options presented for how to communicate this information: either at the document or definition level. Having the least version at the document level would not change the language, and would require no changes to tools. However, in a large document, it may potentially cause a high least version because of just one definition, even when the rest of the definitions have a lower compatible version. Marking the least version on each definition would require a small change to the language, and would require tools to change the way they process OVAL Documents. This would be a more granular and accurate way to communicate version information in the document.

Discussion

A question was asked regarding how content was stored in the repository after processing. An answer was given that we store both the individual parts of the content, as well as the entire XML fragment, to allow a fast search capability, as well as fast retrieval of XML.

The suggestion was made that instead of validating against all schema versions, perhaps we should just compare the differences between versions. This would require having an algorithm for tracking changes from version to version (that is 5.6 -> 5.7, etc.)

The question was asked if this issue only affected major releases, and the answer was given that it indeed affects minor versions as well.

A comment was made that NIST has seen and dealt with this issue by schema validating content to determine least version, in a similar manner to how MITRE is proposing.

A suggestion was made to mark content only at the definition level, understanding that it would have an impact on reusable items like test, objects, and states.

A comment was made that the use of namespaces could alleviate some of the issues here by documenting the different versions at the XML fragment level.

Conclusion

The consensus was that there was a large benefit to implementing the least version principle, as it would help SCAP-validated tools use the content in the repository. There was not a clear consensus on specifically what method to use, and there were several additional features desired by the community. Also, there was a suggestion to make multiple versions of the repository available; specifically, to have an SCAP stream that is compatible with the current version of SCAP. It was noted that the benefit of these additional features would be weighed against other priorities, and that discussion would definitely continue in the OVAL Community Forum.

Application Specific Detection

Background

The OVAL Language currently contains two application-specific schemas: SharePoint and Apache. The SharePoint schema exists to provide a method for accessing SharePoint data as previous mechanisms found within OVAL were found to be inadequate. The Apache schema exists to answer one question: is a particular version of Apache installed?

The Apache schema documentation for the httpd_object states:

> "The httpd_object element is used by an httpd test to define the different httpd binary installed on a system. There is actually only one object relating to this and it is the collection of all httpd binaries. Therefore, there are no child entities defined. Any OVAL Test written to check version will reference the same httpd_object which is basically an empty object element. A tool that implements the httpd_test and collects the httpd_object must know how to find all the httpd binaries on the system and verify that they are in fact httpd binaries."

Because the httpd_object refers to all httpd binaries on the system, a system scan would need to be performed in order to discover all binaries of the name, "httpd". The method and scope of the scan is not defined and thus left up to the implementation to define. For example, one implementation may only scan the running processes, while another implementation may only scan the local file system while omitting mounted network attached devices. The consistency of results between implementations is at risk and cannot be guaranteed because of this.

This topic has been recently discussed on the oval-developer-list. The archives of the conversation can be found here:

> http://making-security-measurable.1364806.n2.nabble.com/apache-httpd-test-tp4985454p4985454.html

The primary suggestions for addressing this issue were:

- Provide an expected path to search under via attribute or child element on httpd_object elements
- Leverage platform-specific mechanisms for verifying installed software (the Windows Registry, RPM, DPKG, etc.)
- Remove application-specific schemas that define tests which could be performed using conventional OVAL mechanisms

The problem of handling version information for specific applications extends well beyond Apache. As such, the existence and purpose of application-specific schemas needs to be re-examined.

Discussion

During the course of discussion the following comments were made:

- In general, the idea of removing or disallowing the creation of application-specific schemas is bad. SharePoint demonstrates the value of application-specific schemas by providing methods for accessing data that was once inaccessible. However, as n the case of Apache, I do not see a problem with removing application-specific schemas which define tests that can be performed using OVAL constructs and best practices.
- OVAL should encapsulate a series of trusted commands that can be invoked on a particular platform for gathering data about a particular application or platform.
  Response: There are obvious security issues with allowing OVAL to invoke scripts or binaries.
  Response: More than likely you are using OVAL for checking for the existence of a CVE or vulnerability so by invoking a particular binary, you could be exposing the system to a compromised or vulnerable binary. I am in favor of having an application-specific schema and checks because sometimes you can't leverage package management systems for discovering versions of installed software. For example, RPM doesn't detect the version of Apache that is bundled with some Oracle software.
  Response: So given the case that you cannot always trust a binary to be executed to gather information, the intelligence for determining the vulnerable state of an application and then magically pulling out the version information must be built into every OVAL-compliant tool. Is that feasible?
  Response: We cannot leverage utilities like strings, or rpm, or dpkg to pull out this information. We can't require tools to decompile binaries and analyze the code. The underlying problem is with the release and installation procedure of applications on various platforms; it's a problem that OVAL is not intended to fix.
- As we have SCAP encompass more and more applications, we will require more application-specific schemas. Relative to this issue, at NIST we have the NSRL (National Software Reference Library) which is a database of hashes for applications. They are congressionally chartered to purchase and hash software. We are trying to work with them to link up CPE information and these hashes.
  Response: Apache is a locally-developed product in essence, because most users who install Apache compile it themselves so as to install particular modules and options. As such, an organization would need to have an internal database of hashes for their internal software configurations.
- OVAL cannot go on to write application schemas for each requested application. The language would grow far too large. We need to deal with this now while we only have two schemas.

- We need to address these schemas on a case-by-case basis and under a lot of scrutiny. SharePoint demonstrates the value of application specific schemas, and others may come up in the future.

Conclusion

In general, schemas like the Apache schema were seen as lacking clarity and thus, should be removed. It was felt that if application information could be gathered using conventional methods content authors should use them instead of proposing a new schema. However, as is the case with SharePoint, if a schema adds functionality to the language and allows once inaccessible information to be gathered, a new schema should be proposed and added to OVAL.

Moving forward, the current Apache test will be deprecated and removed in a later release, pending approval. Current content that uses the http_test should be updated to leverage existing OVAL mechanisms for binary discovery.

Removing the Apache test and rewriting the test using conventional methods guarantees the consistency of OVAL Results among different implementations. By removing ambiguity and requiring authors to provide scoped information, we can place more trust in the content being consumed.

By addressing the issue of application-specific schemas early (while there are only two) we establish a mindset for proposing and adding application-specific schemas to OVAL in the future.

Adding Datatypes

This topic was not discussed and has been raised on the oval-developer-list. This discussion was planned in case there was extra time on the agenda.