

Security Automation Developer Days



June 14, 2010



OVAL Session Overview

- What's Changed Since Last Year?
 - Highlights & Last Year's Topics
- Taming OVAL Results
- Optimizing Item Searches

- Break???

- Least Version Principle for OVAL Content
- Application Specific Detection
- Adding Datatypes
- Wrap Up

What's Changed Since Last Year?





Highlights

- 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 (1)

- 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?”



Last Year's Topics (2)

- Deprecation Policy Review
 - As of version 5.6 all deprecated items now have additional deprecation metadata.
 - Listing of deprecated constructs is available on the OVAL web site.
 - Version 5.7 release removed unused long standing deprecated items.



Last Year's Topics (3)

- Schematron Usage in OVAL
 - OVAL Adoption Requirements require that all content be compliant with XML Schema and Schematron rules.
 - Significant refactoring and testing of the Schematron rules.
 - Introduction of phases to allow more flexibility
 - Tuning of xpath statements to improve performance
 - Version 5.8 drafts include the removal of ~3000 Schematron rules.



Last Year's Topics (4)

- Element Name Reconciliation
 - Agreement that correcting and aligning element names is not worth the implementation effort.
 - Need an explicit mapping between tests, objects, states, and items.
 - This mapping should not be left interpretation based on a pattern in element names.
 - This issue remains open and is currently up for consideration in version 5.8.



Last Year's Topics (5)

- xsd:choice structure on objects
 - Introduced choice structure for filepaths in version 5.6.
 - Established precedence for use in other areas.
 - Next candidate could be Windows Registry related checks.



Last Year's Topics (6)

- Supporting N-Tuples in OVAL

- Introduced the 'record' datatype in version 5.7.

WQL - SELECT Name, ScreenSaverTimeOut FROM Win32_Desktop;

```
<wmi57_item id="oval:sample:ste:2" operator="AND" version="1" xmlns="...">
```

```
  <result datatype="record">
```

```
    <oval-sc:field name="name" datatype="string">user1</oval-sc:field>
```

```
    <oval-sc:field name="screensavertimeout" datatype="int">900</oval-sc:field>
```

```
  </result>
```

```
</wmi_state>
```

- Created new versions of applicable tests to utilize the new datatype.
- Still need to consider records of records and records as variable values.



Last Year's Topics (7)

- Pattern match on enumerations
 - As of version 5.6 all schema documentation has been updated to allow the use of the pattern match operation on entities that are normally restricted to a set of enumerated values.
 - Due to the restriction on element values the regular expression must be supplied via a variable reference.



Last Year's Topics (8)

- Tests reference multiple states
 - As of Version 5.6 a test may reference 0 – N states.
 - In use in the OVAL Repository.
 - Simplifies complex tests.



Last Year's Topics (9)

- Introduce PCRE based pattern matches
 - Version 5.6 changed the supported regular expression syntax to PCRE with only minor exceptions.
 - Regular Expression Support document posted on the OVAL web site and referenced in schema documentation.



Last Year's Topics (10)

- Emerging Use Case: “*OVAL for System Inventory?*”
 - Ongoing work on “OVAL Reporting” based on last years feedback.
 - Request for comments – Jan 5, 2010
 - Draft schema release – April 14, 2010
 - Under consideration for inclusion in version 5.8
 - Need community support and feedback.
 - More on this later...

Taming OVAL Results





Taming OVAL Results

Smaller More Useful Result Sets

- Feedback and requests for:
 - more granular evaluation results
 - results that scale to the enterprise
 - results that include only the actionable information
 - highlight the hidden data in OVAL Results
 - ... and maintain interoperability



More Granular Evaluation Results

Feature Request: *add capability to specify test result other than true, or false*

Users would like OVAL Results to support the following examples:

- 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

- 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

Neither compliant nor noncompliant if the application or file does not exist.



What are the result values?

- 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.



What does 'not applicable' really mean?

*... 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.*

- Allows a content author to logically combine criteria for multiple platforms.
 - a vulnerability definition that applies to windows and linux can use the `<registry_test/>` and the `<rpminfo_test/>`.
- not applicable results are **not considered** when determining an aggregate result.
 - false AND not applicable ==> false
 - true AND not applicable ==> true



What is the underlying issue?

- Tests have a `@check` and a `@check_existence` attribute.
 - `@check_existence` – an assertion about the number of items in the collected set
 - `@check` – an assertion about the state of the items the collected set
- Cannot differentiate a `false` result due to failing the `@check_existence` from a `false` result due to failing the `@check`.
 - When determining the result of a test if the `@check_existence` is satisfied (`true`) then the `@check` is considered. Otherwise the result is the result of the `@check_existence`.



Option 1: Add a Result Value

- Should we add a new result value?
 - Continue to use not applicable when a test is not valid on a given platform
 - When @check_existence fails report some other false like value like **false_existence**
- File permissions example reworked:
 - 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

Considerations:

- A test essentially returns a boolean with the addition of a few possible error codes.
 - Should we move away from a boolean response?
- Evaluation tables would need to be updated
 - How do you combine the possible result values?
false AND false_existence ==> ???
- Does this lead to a proliferation of result values?
- Is this a problem that OVAL should solve?
 - Leave the solution up to higher level context. Could the same result be achieved by separating the existence check from the state check with two definitions.

Option 2: Add an Attribute

- Should we add an attribute to hold just the `@check_existence` result?
 - Add `@existence_result` to `oval-res:TestType`
 - `@result` attribute remains unchanged
 - new `@existence_result` attribute holds the result of the `@check_existence` evaluation
- File permissions example reworked:
 - `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

Considerations:

- Does not change the evaluation result
 - Only metadata about the evaluation of `@check_existence`
 - Maintains the boolean response of tests and definitions
 - No need to alter evaluation tables
- Need to extract the extra information to differentiate a false due to an existence failure from a false due to a state failure.
- Is this a problem that OVAL should solve?
 - Leave the solution up to higher level context. Could the same result be achieved by separating the existence check from the state check with two definitions.



Additional Considerations

- Are there other options to consider?
 - Neither option fully addresses the requested capability
 - No consistent evaluation data to use for driving some other type of result value
- Does OVAL need to support this capability?



Taming OVAL Results

Smaller More Useful Result Sets

- Feedback and requests for:
 - more granular evaluation results
 - results that scale to the enterprise
 - results that include only the actionable information
 - highlight the hidden data in OVAL Results
 - ... and maintain interoperability



Lightweight OVAL Results

- Feature Request: *add result directive to allow for lighter weight results with ability to track causal information*
 - Need for smaller more useful result sets.
 - Only include the data that is useful downstream.
 - results that scale to the enterprise
 - results that include only the actionable information
 - highlight the hidden data in OVAL Results
 - ...and maintain interoperability



OVAL Results Directives

- A set of flags that describe the information included in a results file.
 - Include & exclude results by result type
 - directives state which of the possible results are included
 - If a given result type is set to `false` the all definitions with that result value are excluded.
 - Control the level of detail for included results
 - `thin` – only the minimal amount of information will be provided
 - The criteria child element of a definition **MUST** not be present
 - System characteristic information for the objects used by the definition **MUST** not be presented.
 - `full` – very detailed information will be provided
 - The results of all extended definitions and tests included in the criteria
 - The actual information collected off the system must be presented.
 - Directives are set by the result producer
 - Assumption that the result producer is somehow configurable.



OVAL Results Directives Example

```
<directives>
  <definition_true content="full" reported="true"/>
  <definition_false content="thin" reported="true"/>
  <definition_unknown content="full" reported="true"/>
  <definition_error content="full" reported="true"/>
  <definition_not_evaluated content="thin" reported="false"/>
  <definition_not_applicable content="thin" reported="false"/>
</directives>
```

- The document includes:
 - Full details for true results
 - Full details for unknown results
 - Full details for error results
 - Minimal information for false results
- The document excludes:
 - All not evaluated results
 - All not applicable results

Example results:

- definition document: 105KB
- full results: 143KB
- tailored results: 117KB



How do directives fall short?

- Does anyone really support the directives?
 - only full result explicitly called out in SCAP 1.0
 - not currently implemented in OVALDI
- Do we need more options than `thin` and `full`?
 - Does the definition document need to be included?
 - How does a result consumer really know what was evaluated?
- Do we need to consider `definition @class`?
 - Include full results for true vulnerability definitions and thin results for all other true definitions
- How do more options impact interoperability?
 - For generic content sharing aren't full results needed?



Taming OVAL Results

Smaller More Useful Result Sets

- Feedback and requests for:
 - more granular evaluation results
 - results that scale to the enterprise
 - results that include only the actionable information
 - highlight the hidden data in OVAL Results
 - ... and maintain interoperability



What is the cause?

Feature Request: *add result directive to allow for lighter weight results with ability to track causal information*

- Need for smaller more useful result sets
 - highlight the hidden data in OVAL Results
 - result format with just the actionable information
 - ...and maintain interoperability
- How do we provide only the causal information?
 - Full OVAL Results most likely contain required data
 - data is difficult to uncover

What is the cause?

Example definition id=123

- compliant config on windows 7
 - windows 7 is installed AND
 - minimum password length is 8 or more

Example definition id=123 (FALSE)

- (FALSE) compliant config on windows 7
 - **(TRUE)** windows 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



Is There and Existing Solution?

- Have vendors already solved this problem?
 - Some products display the desired information already.
 - How are vendors solving this problem?

- Can we develop a common solution?

Sample Compliance Definition

```
<definitions>
  <definition id="oval:example:def:6" version="1" class="compliance">
    <metadata>...</metadata>
    <criteria operator="AND">
      <extend_definition comment="Windows Vista is installed" definition_ref="oval:example:def:228"/>
      <criteria comment="min passwd length is correct" test_ref="oval:example:tst:61"/>
    </criteria>
  </definition>
  <definition id="oval:example:def:228" version="3" class="inventory">...</definition>
</definitions>
<tests>
  <family_test id="oval:example:tst:99" comment="the installed operating system is Windows" ...>
    <object object_ref="oval:example:obj:99"/>
    <state state_ref="oval:example:ste:99"/>
  </family_test>
  <registry_test id="oval:example:tst:7914" comment="Windows Vista is installed" ...>
    <object object_ref="oval:example:obj:5590"/>
    <state state_ref="oval:example:ste:3828"/>
  </registry_test>
  <passwordpolicy_test id="oval:example:tst:61" comment="min passwd length is correct" ...>
    <object object_ref="oval:example:obj:61"/>
    <state state_ref="oval:example:ste:61"/>
  </passwordpolicy_test>
</tests>
<objects>
  <family_object id="oval:example:obj:99" comment="This is the default family object." .../>
  <registry_object id="oval:example:obj:5590" comment="This registry key ProductName" ...>
    <hive>HKEY_LOCAL_MACHINE</hive>
    <key>SOFTWARE\Microsoft\Windows NT\CurrentVersion</key>
    <name>ProductName</name>
  </registry_object>
  <passwordpolicy_object id="oval:example:obj:61" .../>
</objects>
<states>
  <family_state id="oval:example:ste:99" comment="Microsoft Windows family" ...>
    <family>windows</family>
  </family_state>
  <registry_state id="oval:example:ste:3828" comment="The registry key matches with Vista" ...>
    <value operation="pattern match">.*[Vv]ista.*</value>
  </registry_state>
  <passwordpolicy_state id="oval:example:ste:61" ...>
    <min_passwd_len operation="greater than or equal" datatype="int">12</min_passwd_len>
  </passwordpolicy_state>
</states>
```



A Closer Look at States

```
<passwordpolicy_state id="oval:example:ste:61" ...>  
  <min_passwd_len operation="greater than or equal" datatype="int">12</min_passwd_len>  
</passwordpolicy_state>
```


- The interesting data is generally examined with a State entity
 - There can be many states, each with many entities
- States relate to Items
 - An entity in a state has a multiplicity of 0 – 1
 - Items have entities that correspond to state entities
 - Item entities may have a multiplicity of 0 – N



Allow authors to flag the interesting State entities

- ASSUPTION: A definition author knows which items are most interesting.
 - Allow the author to flag or highlight a piece of data

```
<passwordpolicy_state id="oval:example:ste:61" ...>  
  <min_passwd_len operation="greater than or equal"  
    datatype="int"  
    report_value="true">12</min_passwd_len>  
</passwordpolicy_state>
```

A red arrow points from the left towards the `report_value="true"` attribute in the XML snippet, highlighting the new attribute.

- Add a new `@report_value` attribute
 - optional attribute on all state entities
 - true indicates that the system value(s) should be highlighted/reported
 - false no behavior change (default is false)



Reported Values in OVAL Results

```
<test test_id="oval:example:tst:1" result="true" check="at least one"  
  check_existence="at_least_one_exists" state_operator="AND" ...>  
  <tested_item item_id="2" result="true">  
    <observed_value name="state_entity_name" datatype="int">6</observed_value>  
  </tested_item>  
</test>
```

Expand `<oval-res:tested_item/>`
to hold each value that is reported.

- Associates the exported value with a test
 - Each test can have multiple `<oval-res:tested_item/>`
 - One item might hold many `<oval-res:observed_value/>`
- Records the name of the State entity
 - Easy to find the specific State entity
- Records the observed datatype

Considerations:

- Does not fully present the desired result information
 - Complex criteria structures will remain difficult to determine an underlying cause
- However, the desired data is accessible for any simple case
- Makes the result document even bigger



What About OVAL Reporting?

What does OVAL Reporting do?

1. Collect information about a system
 - E.g., registry key values, file permissions, etc.
2. Format the information as specified by the author
 - Supports correlation of data from multiple sources (e.g. WMI & Registry or RPM info & processes)
 - Can output in any text-based format (text, HTML, XML, etc.)

OVAL Reporting is not about making an assertion

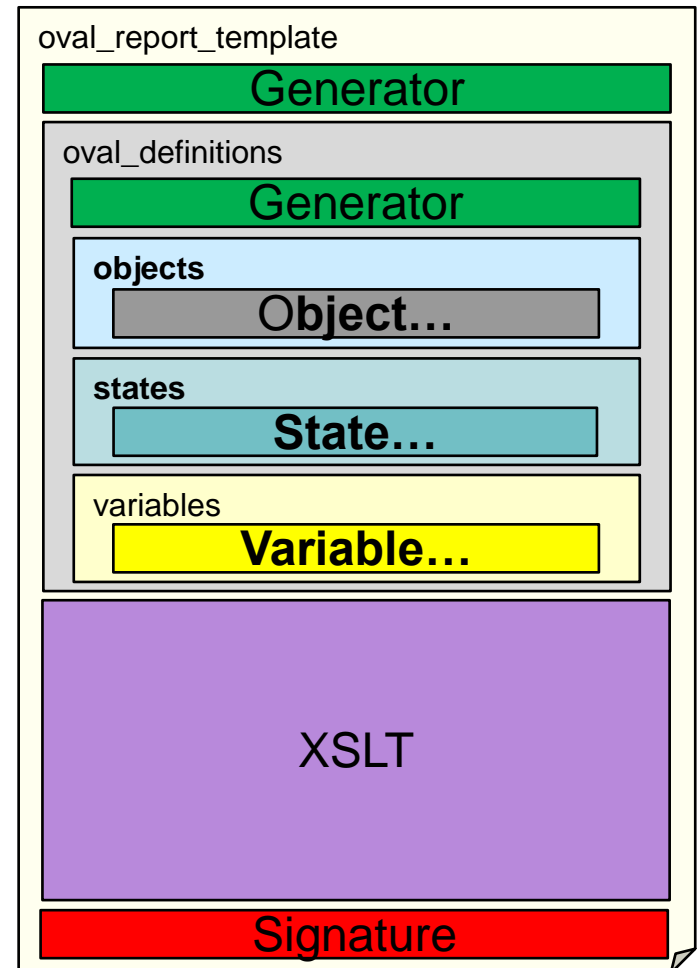
OVAL Report Template Structure

- **Generator**
 - Metadata about file construction

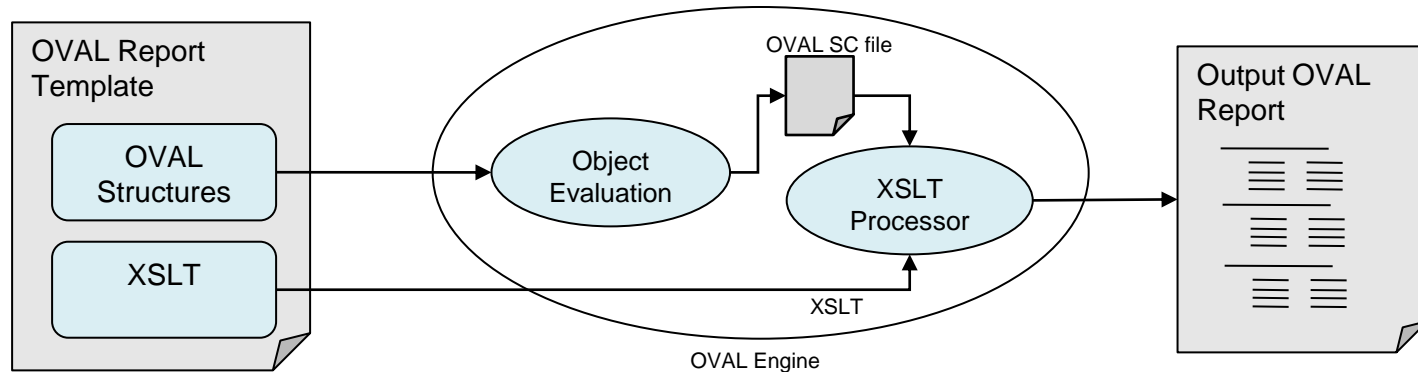
- **OVAL Block**
 - OVAL Objects, States, and Variables
 - Guides data collection

- **XSLT Block**
 - XSLT 1.0 content
 - Guides data organization and formatting

- **Signature**
 - Optional digital signature



OVAL Report Template Processing



- Start with an OVAL Report Template
 - Contains OVAL Structures and XSLT
- OVAL structures are sent to the OVAL Engine
 - Objects are collected
 - An OVAL System Characteristics (SC) file is produced
- SC file and XSLT portion of the Report Template are passed to a basic XSLT processor
- The XSLT processor produces the output report as defined by the XSLT



Taming OVAL Results Summary

Smaller More Useful Result Sets

- Feedback and requests for:
 - more granular evaluation results
 - results that scale to the enterprise
 - results that include only the actionable information
 - highlight the hidden data in OVAL Results
 - ... and maintain interoperability
- What didn't we address?
 - Was a failure due to a precondition?
 - Complex content will remain complex

Optimizing Item Searches





Optimizing Item Searches

- Topic started on the oval-developer-list
 - <http://making-security-measurable.1364806.n2.nabble.com/oval-object-criteria-tp4931198.html>
- We will cover:
 - Background
 - How can we optimize now?
 - How could we optimize in 5.8?
 - Looking beyond 5.x
 - Discussion



Optimizing Item Searches

- Background
- How can we optimize now?
- How could we optimize in 5.8?
- Looking beyond 5.x
- Discussion



Background

- Need to find all files on a system that are world writable
- How would we accomplish this?



Background

- We need to 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>
```



Background

- We need to express the state of a world writable file

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



Background

- We need to check every collected item against the specified state

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



So What's the Problem?

- OVAL System Characteristics and Results files are created with a `<file_item>` for **EVERY** file found on the system
 - Most items are un-needed
 - 1254 out of 148745 files were world writable (less than 1%)
 - Creates very large files
 - 160 MB for all files (1.24 MB for world writable files)
- Affects the performance of tools
 - CPU & memory
 - Time
- Does anyone see this as relevant problem today? What about in the future?



Optimizing Item Searches

- Background
- How can we optimize now?
- How could we optimize in 5.8?
- Looking beyond 5.x
- Discussion



Using @flag= 'incomplete'

- The @flag attribute provides information about the outcome of a collected object in an OVAL System Characteristics file.
- *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...*
- How can we optimize with an 'incomplete' object?
- Let's look at an example...



Using @flag='incomplete'

Evaluate to true if there are no world writable files

```
<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>
```



Using @flag='incomplete'

```
<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>
```

Let's optimize!



Using @flag='incomplete'

- We can optimize in two ways:
 - Based on items specified by the filter
 - Based on the @check_existence attribute
- We cannot always determine a result with an 'incomplete' object (@check='all')



Optimizing Item Searches

- Background
- How can we optimize now?
- How could we optimize in 5.8?
- Looking beyond 5.x
- Discussion



Option 1: Add a <filter> Element

```
<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">
  <overwrite datatype="boolean">1</overwrite>
</file_state>
```

Considerations:

- **Already familiar with filters**
- **Major impact**
 - All objects
- **High flexibility**
 - Unbounded (0 to n)
 - Could result in @flag='does not exist'
- **Implementation**
 - May be similar to applying filters in sets
- **Filtering doesn't always make sense**
 - Single item objects (<win-def:passwordpolicy_object>)
 - Objects with small data sets (<unix-def:interface_object>)
- **Another authoring choice**



Option 2: Add a Filter Behavior

```
<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>
```

Considerations:

- **Already familiar with behaviors**
- **Limited impact**
 - Only target the objects that make sense
- **Less flexibility**
 - Bounded (0 to 1)
 - Could result in @flag='does not exist'
- **Implementation**
 - May be similar to applying filters in sets



Option 3: Add Entity Behaviors

```
<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>
```

Considerations:

- **Already familiar with behaviors**
- **Limited impact**
 - Only target the objects that make sense
- **Very limited flexibility**
 - Bounded (0 to 1)
 - Could result in @flag='does not exist'
 - Cannot utilize operations or variables
- **Implementation**
 - Very object specific
- **Poor scalability**
 - There may be many entities (<unix-def:file_state> has 23 entities!)



Can We Just Add Object Entities?

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

- Yes, we could...
 - If we made additional entities “required”
 - Would invalidate existing content
 - Would need to deprecate existing tests, objects, and states
 - What if I don’t care about particular entities?
 - If we made additional entities “optional”
 - Would not invalidate existing content
 - Could specify only the entities needed
- But, this actually changes a lot...
 - Changes how we think of objects!
 - Extensive changes to the component schemas
 - Tools would need to support this functionality
- Would we really want to do this in a minor release?



Optimizing Item Searches

- Background
- How can we optimize now?
- How could we optimize in 5.8?
- Looking beyond 5.x
- Discussion



Make All Entities Searchable in 6.0

- Major overhaul of the language
 - Opportunity to simplify the language
- 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?



Optimizing Item Searches

- Background
- How can we optimize now?
- How could we optimize in 5.8?
- Looking beyond 5.x
- Discussion



Discussion

- 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, or comments

Least Version Principle for OVAL Content





Introduction

- The OVAL Repository contains content that has been submitted by members of the community
 - Includes vulnerability, compliance, inventory, and patch definitions
- Provides valid OVAL Documents for tool and community consumption
- The OVAL Repository serves as a model for other content repositories
- How do we know which version of the schema a document is compatible with?
 - Are changes necessary?



Current Versioning Policy

- What do we do now?
 - The repository only serves one version of the content (the latest version)
 - All content is moved forward with each release
 - No changes to the content
 - No removal of deprecated language constructs
- What does this mean?
 - + Easy to maintain
 - + Easy to create content
 - + Encourages tool compatibility with the latest version of the language
 - No file history
 - Users and tools must figure out how to handle later content



Things to Consider

- Is the current policy sufficient?
- Factors to consider with any solution:
 - Maintainability
 - Investment in infrastructure implementation
 - Difficulty in content creation
 - Backwards compatibility
 - Tools performance



Least Version Principle

- How can we better communicate the schema compatibility for a given document?
- Provide a language construct that dictates the oldest compatible version
 - Document level: use existing `<schema_version>` element in `<generator>`
 - Definition level: create new `<schema_version>` element in definition
- Provide a policy by which that version is determined
 - Set by content creators
 - Automatically determined by repository tools

Application Specific Detection





Introduction

- Application specific schemas exist in OVAL
 - Apache
 - SharePoint
- Why?
 - SharePoint: can not use conventional OVAL methods to gather certain information
 - Apache: difficult to express inventory definitions
 - Is Apache x.y.z Installed?



Recent Issues

- Developer list discussions regarding the Apache httpd_test
 - The httpd_object refers to ALL httpd binaries on a system
 - A scan of a system is required by OVAL interpreters
- “Scanning the system” has different meanings
 - Scan the output of ps
 - Scan the local filesystem
 - Scan all attached devices
 - Consistent results at risk
- Opinions regarding this problem varied
 - Definition authors should provide an expected binary hash
 - Definitions should leverage package management systems
 - Objects should refer to an expected location
 - OVAL should not dictate methods of discovering binaries
- The problem exists outside of Apache
 - PHP
 - Java
 - MySQL



Discussion

- Should OVAL have application-specific schemas?
- Can we leverage existing OVAL constructs?

Adding Datatypes





Many Requests for New Datatypes

- Feature Request: *add support for xml values*
 - Examine xml that may be extracted from other repositories (SQL database, Active Directory, etc.)
- Feature Request: *add support for ipv4 to ipv6 address comparisons*
 - Compare ipv4 to ipv6 and ensure the two addresses are properly formatted.
- Feature Request: *add path datatype to allow for proper path comparisons*
 - Compare two paths. Are /var/tmp/ and /var/tmp equal?
- ... many more possibilities



Discussion

- Datatypes are defined in the oval-common-schema
 - Reused across all the core schemas
 - Allow for customization of operations
- Can there be too many datatypes?
 - All definition evaluators must support all datatypes
- Should we add in all these new datatypes or stick with more primitive types?



Thank You
