An Introduction to the OVAL™ Language

Version 5.0

Introduction **3**

OVAL Language Benefits 3

OVAL Language Use Cases **3**

Security Advisory Distribution 4
Vulnerability Assessment 4
Patch Management 4
Configuration Management 4
Auditing and Centralized Audit Validation 5
Security Information Management Systems (SIMS) 5
System Inventory 5

How the OVAL Language Works **5**

Collecting Information from Systems 6
Standardized Tests 6
Results of the Tests 7

Structure of the OVAL Language **7**

The OVAL Definition Schema 7
The OVAL System Characteristics Schema 7
The OVAL Results Schema 7
The Component Schemas 8
Placement of a New Test *9*
Relation to the Affected Family *9*

The OVAL Language Review Process **10**

Planning 10
Draft/Internal Review 10
Release Candidate 10
Official 10

OVAL Language Versioning **10**

Previous Solutions 11
Single Language Version *11*
Point System *11*
Dated Version *12*
Accepted OVAL Ideology 12
Single-Point Version *12*
What Constitutes a Language Version Change *13*
Differentiating Language Versions via Namespace *13*

Community Participation **13**

OVAL-Compatible Products and Services 14

Conclusion **14**

Introduction

Open Vulnerability and Assessment Language (OVAL™) is an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services. OVAL includes a language used to encode system details, and an assortment of content repositories held throughout the community. The language standardizes the three main steps of the assessment process: representing configuration information of systems for testing; analyzing the system for the presence of the specified machine state (vulnerability, configuration, patch state, etc.); and reporting the results of this assessment. The repositories are collections of publicly available and open content that utilize the language.

The OVAL community has developed three schemas written in Extensible Markup Language (XML) to serve as the framework and vocabulary of the OVAL Language. These schemas correspond to the three steps of the assessment process: an OVAL System Characteristics schema for representing system information, an OVAL Definition schema for expressing a specific machine state, and an OVAL Results schema for reporting the results of an assessment.

Content written in the OVAL Language is located in one of the many repositories found within the community. One such repository, known as the OVAL Repository, is hosted by The MITRE Corporation. It is the central meeting place for the OVAL Community to discuss, analyze, store, and disseminate OVAL definitions. Each definition in the OVAL Repository determines whether a specified vulnerability, configuration issue, or patch is present on a system.

The information security community contributes to the development of OVAL by participating in the creation of the OVAL Language on the OVAL Developers Forum and by writing definitions for the OVAL Repository through the OVAL Community Forum. An OVAL Board consisting of representatives from a broad spectrum of industry, academia, and government organizations from around the world oversees and approves the OVAL Language and monitors the posting of the definitions hosted on the OVAL Web site. This means that the OVAL, which is funded by US-CERT at the U.S. Department of Homeland Security for the benefit of the community, reflects the insights and combined expertise of the broadest possible collection of security and system administration professionals worldwide.

OVAL Language Benefits

- A simple and straightforward approach for determining if a vulnerability, configuration issue, or patch exists on a given system.
- Standard schemas that outline the necessary security-relevant configuration information.
- A single XML document that encodes the precise details of specific issue.
- An open alternative to closed, proprietary, and replicated efforts.
- Supported by a community of security experts, system administrators and software developers
- Industry-endorsed via the OVAL Board and OVAL Developers Forum.

OVAL Language Use Cases

The OVAL Language enables interoperability between security products by providing a standard XML language with which to exchange information. It allows tools in different vertical markets to leverage other tools that accomplish different tasks. For example, a vulnerability assessment tool can leverage a vulnerability research service. Or a compliance checking engine can leverage Government security guidance.

Dividing the OVAL Language into three distinct schemas that represent the three phases of the checking process enables organizations to implement only those parts of the OVAL Language that truly make sense for their tool(s).

The advantage of this is that the OVAL Language is available to a wider range of security tools, thus enabling greater interoperability within the industry. The cases below exemplify some of the needs for a standard like OVAL Language, and its potential uses within the security community.

Security Advisory Distribution

One acknowledged need within the security industry is for application and operating system vendors to publish vulnerability information in a standard, machine readable format. The benefit of this is two-fold; first, it provides scanning tools with immediate access to content that can be used to assess the security posture of a system, and second, it moves the authoring of the technical details of a vulnerability from the reverse engineering efforts of the scanner tool developer, to a more authoritative source, the developer of the vulnerable software.

Vulnerability Assessment

Currently, organizations that develop vulnerability assessment tools also need to employ a team of content developers. The role of this team is to investigate vulnerabilities as they become known, gather all of the available information for a given vulnerability, run various tests against live systems to examine the parameters that indicate the presence of a vulnerability, develop the tests in the language of the tool in question, and do this all under a very strict time requirement. The final requirement obviously runs counter to those before it, and results in the potential for analysis to not be completely fulfilled before a test absolutely has to be disseminated to the vendor's customers.

For vendors, having vulnerability information structured in a standard format allows them to quickly consume data from multiple sources, and lets them refocus some of their resources away from content generation, and onto tasks to enhance the functionality of their tool.

From the tool customer's perspective, the primary requirement for having a standard content format is that it demystifies the vulnerability assessment process, and provides them with the ability to do apples-to-apples comparisons of tools. Having a well documented, standard format, provides users with the information they need to be able to understand the details of an issue, and to determine how a specific tool is conducting its business. The open standard also provides users with the opportunity to generate their own statements in the language and, if a tool allows, interpret them. When conducting tool comparisons, given a specific set of definitions, each tool tested should return the same result. If this is not the case, it is no longer a result of tools taking different approaches to detecting a vulnerability, and removes the burden from the customer to determine which they think returns the most accurate results. The end result is that the customer can focus more on selecting a tool with the features that best meet their needs, and less on the more difficult problem of which does the correct job of detecting vulnerabilities.

Patch Management

The needs of the patch management vendors are similar to those of the vulnerability assessment vendors. There is some amount of reverse engineering that must be done in order to generate content for their tool to be able to apply a patch. Having this information generated by the software vendor, and formatted in a standard format removes the reverse engineering requirement, and allows content to be provided to the end customers in a more timely fashion. A second requirement for this class of tools is to be able to easily consume vulnerability assessment results from a variety of scanning tools. If these results were offered in a standard format, interoperability between tools would no longer be an issue.

Configuration Management

Configuration management tools concern themselves with examining a machine's configuration state, comparing it against a known good or mandated configuration state, and reporting the results. There are a number of publicly available best practice configuration guides (e.g. The National Security Agency (NSA) Configuration Guides), and many more developed specifically for individual organizations. In many cases, these guides exist in paper form only, and it is up to the IT Staff to translate the document into something that can be applied, and enforced on a consistent basis. There are also automated solutions available, most notably the Center for Internet Security's Benchmark tools, which can scan a system for compliance against a given configuration, and offer tailoring

capabilities to suit the specific needs of an organization. Unfortunately, these tools often rely upon proprietary data formats, making it difficult to introduce new policies to the tool, or move data from one tool to another.

Having a standard language for expressing system configuration issues offers many benefits in this area. Firstly, a single configuration specification need only be written once. At this point it can be consumed by any configuration management tool. Secondly, organizations can more easily develop and maintain their own configuration standards, as it only requires learning a single language, and not a language specific to a particular tool. Finally, as with some of the cases above, divesting the language from the tool provides the tool vendor with a wider repository of content, and allows them to focus more on functionality and features.

Auditing and Centralized Audit Validation

Audit validation is responsible for providing reports about the state of a machine at any given time in the past. There are two basic needs in this area; first and foremost is capturing machine configuration information at a level of granularity that allows an organization to monitor, track, and reconstruct the transition of a system's configuration from one state to another. The second need is that the data be stored in a standardized, data-centric format, thus ensuring that it is not bound to a specific tool, which may or may not be available at the time it is necessary to review the data.

Security Information Management Systems (SIMS)

Security Information Management Systems (SIMS) rely upon the output of a variety of security, auditing, and configuration tools, as well as their own agents, to build a comprehensive view of the security posture of an organization's network. Clearly, the fewer data formats the SIM needs to understand, the more flexible and powerful this tool can be. As with the Patch Management class of tools, standardizing the data exchange formats between tools greatly simplifies the interoperability requirements, and provides the end-users with a wider array of applications from which to choose.

System Inventory

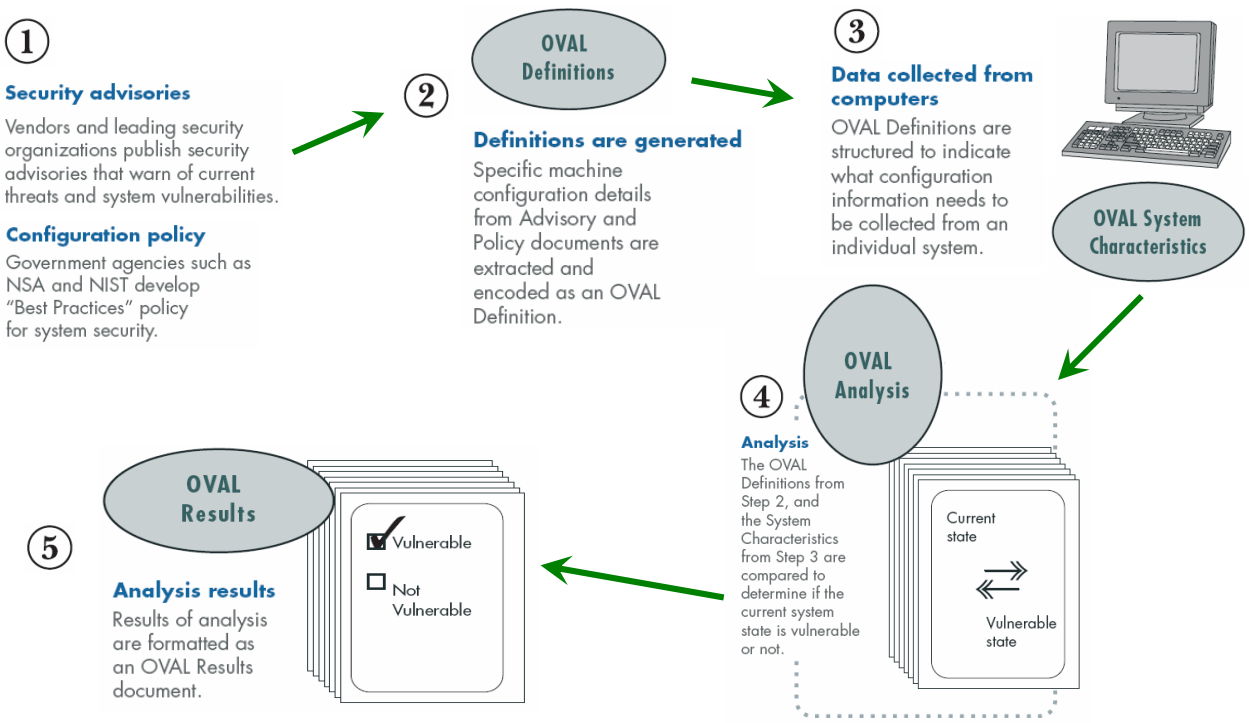
A common issue that exists, not only for security tools, but any tool that is trying to conduct some sort of evaluation of a computer system, is determining the attributes of that system (e.g. operating system, patch level, installed applications…), and being able to convey those attributes clearly and consistently. Currently, there is no universally accepted method for gathering the attributes from a system, nor is there a common way to express those attributes such that they can be easily consumed by another application. The need for this capability is widely acknowledged, and its use would be widespread.

How the OVAL Language Works

The OVAL Language involves three main categories:

- OVAL System Characteristics schema for collecting configuration data from systems for testing.
- OVAL Definition schema for testing the presence of a specific machine state. (vulnerable, compliance, etc)
- OVAL Results schema for reporting the results from the evaluated systems.

The diagram below shows how these three categories work together during a standard vulnerability assessment process.



Collecting Information from Systems

The OVAL System Characteristics schema defines a standard XML format for representing system configuration information, which includes operating system parameters, installed software application settings, and other security relevant configuration values. The schema provides a "database" of system characteristics against which OVAL definitions can be compared in order to analyze a system for the presence of a particular machine state. The schema can also be used as an exchange format that can be incorporated into a variety of tools. By utilizing the provided OVAL System Characteristics file, other applications would not need to perform data collection, but rather can use the provided information to perform analysis. MITRE's reference OVAL Interpreter is an example of an application that generates data in the OVAL System Characteristics schema format and makes it available to these other applications. Other information security products and services that incorporate the OVAL System Characteristics schema are listed on the OVAL-Compatible Products and Services webpage at http://oval.mitre.org.

Standardized Tests

The OVAL Definition schema is the language framework for writing OVAL Definitions in XML. OVAL Definitions encode the details of a specific machine state (when is a system vulnerable, in compliance, etc.) enabling testing of a system to be automated. The OVAL Language's standardized schemas also allow a wide range of computer security professionals to discuss the technical details of determining whether a vulnerability is present on a system, whether the configuration settings of a system meets a security policy, and/or whether a patch is present on a system.

There are two parts to the schema for writing OVAL Definitions, a core schema that describes the basics of the format, and individual component schemas for tests that are specific for individual OS platforms or applications. For example, there is a UNIX schema containing test written for UNIX platforms, and a Windows schema for tests written for Windows platforms.

Results of the Tests

The OVAL Results schema defines a standard XML format for reporting the results of an evaluation of a system. The results data contains the current state of a system's configuration as compared against a set of OVAL Definitions. The OVAL Results schema allows applications to consume this data, interpret it, and take the necessary actions to mitigate the vulnerabilities and configuration conflicts. For example, installing patches, altering system configuration settings, and/or taking external precautions to limit access to the affected systems. This schema also defines a standard exchange format that can be incorporated into a variety of tools. MITRE's reference OVAL Definition Interpreter is an example of an application that generates data in the OVAL Results schema format, and makes it available to other applications. Other information security products and services that incorporate the OVAL Results schema are listed on the OVAL-Compatible Products and Services webpage at http://oval.mitre.org.

Structure of the OVAL Language

The official OVAL Language consists of three different categories: system characteristics, definitions, and results. Each category contains a core schema and a number of component schemas.

The OVAL Definition Schema

The OVAL Definition schema is used to define the XML framework for writing: (1) OVAL Vulnerability Definitions by defining the conditions that must exist on a computer for a specific vulnerability to be present, (2) OVAL Patch Definitions by defining the conditions on a computer that determine whether a particular patch is appropriate for a system, and (3) OVAL Compliance Definitions by defining the conditions on a computer that determine compliance with a specific policy or configuration statement.

The OVAL Definition schema, along with the OVAL System Characteristics schema and OVAL Results schema, are not individual schemas unto themselves but are each composed of a “core” schema and a collection of “component” schemas. The core schema provides the general structure of an OVAL Definition, as well as a place for expressing metadata that is independent of the tests (e.g., CVE identifier, affected platforms, and descriptions), while the component schemas define the specific tests the OVAL Language uses to identify vulnerability, configuration, and security issues within an operating system (OS) or application.

The OVAL System Characteristics Schema

The OVAL System Characteristics schema defines a standard XML format for representing system configuration information. This configuration information includes OS parameters, installed software application settings, and other security relevant configuration values. The purpose of this schema is to provide a “database” of system characteristics against which the OVAL Definitions can be compared in order to analyze a system for vulnerabilities, configuration issues, and patches. In essence, the schema defines a standard system characteristics exchange format that can be incorporated into a variety of tools, such as those listed in the OVAL-Compatible Products and Services section on the OVAL Web site.

As with the OVAL Definition schema, the OVAL System Characteristics schema is composed of a core schema and a collection of component schemas. The function of the core schema is similar to that of the core schema in the OVAL Definition schema. The component schemas define the format and content of the configuration parameters that are collected, and roughly correspond to the tests specified within the OVAL Definition schema.

The OVAL Results Schema

The OVAL Results schema defines a standard XML format for storing the results of an evaluation of a system. The results data contains the current state of a system’s configuration as compared against a set of OVAL Definitions. The OVAL Results schema allows applications to consume this data, interpret it, and take the necessary actions to mitigate the vulnerabilities and configuration issues (for example, install patches, alter system configuration settings, and/or take external precautions to limit access to the affected systems). This schema also defines a standard

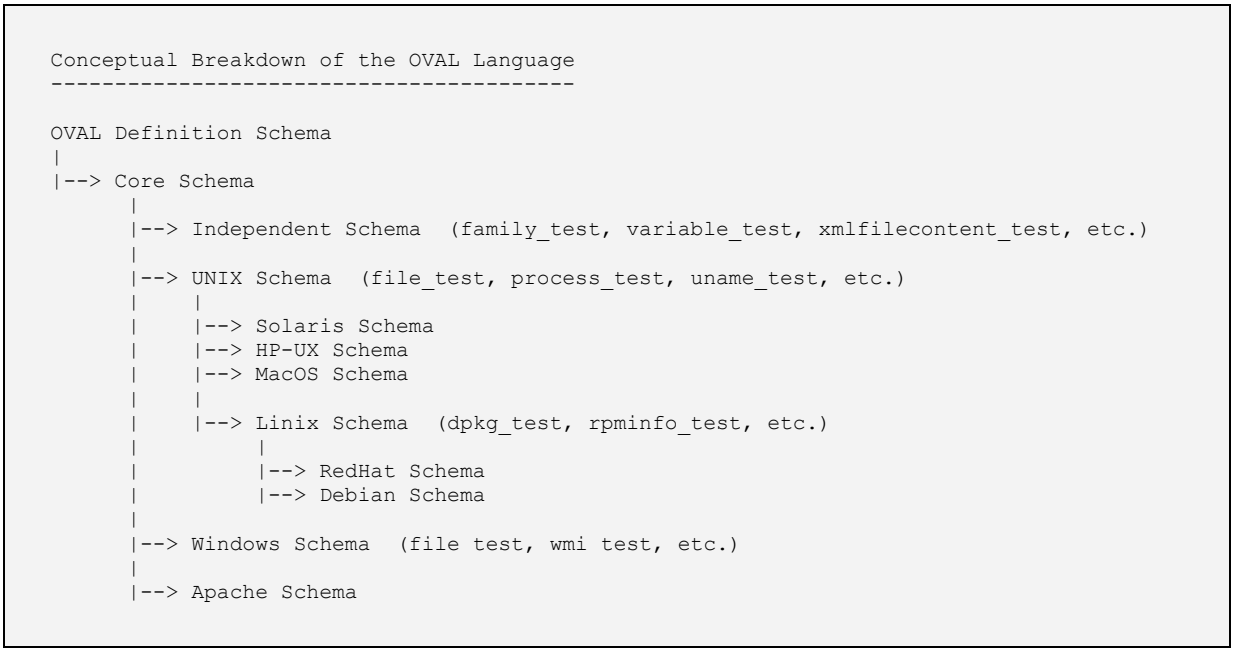
vulnerability and misconfiguration exchange format that can be incorporated into a variety of tools, such as those listed in the OVAL-Compatible Products and Services section on the OVAL Web site.

Again, the OVAL Results schema is composed of a core schema and a collection of component schemas. The primary difference between the component schemas in the OVAL Results schema versus the OVAL Definition schema, is that in the OVAL Results schema each test contains the information used to determine the presence or absence of a vulnerability or misconfiguration in addition to the specific system data analyzed to arrive at the final result.

The Component Schemas

The component schemas consist of a collection of OVAL Tests that are related at some level by the software they describe. Tests that are identical across multiple different component schemas, and thus represent a more general class of test, are grouped together in a higher-level component schema. For example, there are different component schemas for Red Hat, Solaris, and Debian, but all of these have the same structure for a file test. Because of this similarity, the file test is located in a conceptually higher-level UNIX schema. The conditions that determine the level at which a test should be placed are outlined later in this document.

This hierarchical structure of schemas is purely conceptual, however, as there is no actual linking between them. For example, a package test in the Solaris schema does not inherit aspects from a package test in the UNIX schema. Note that all tests do inherit a standard set of attributes and a note element from the core schema. The conceptual hierarchy is a by-product of the way the OVAL Language has developed. While the need for a true hierarchy within the OVAL Language has not yet arisen, it has been recognized that this need may evolve at some point in the future.



OVAL Definitions can pull tests from multiple different schemas. For example, an individual definition might use the family_test from the independent schema (note that the independent schema defines a collection of tests that are not related to a specific piece of software), the process_test from the UNIX schema, and the rpminfo_test from the Linux schema. The validation of these definitions is guided by the namespace associated with each test.

An advantage of multiple component schemas is the separation of tests that describe different types of software (e.g., the tests that deal with Red Hat and Solaris), while at the same time grouping related tests together (e.g., the tests dealing with common aspects of UNIX). A software vendor who finds the need to create an addition to the OVAL Language in order to support their software can group the set of new tests into a new component schema. The ease with which a new component schema can be added to the OVAL Language allows the software vendor to focus their efforts on refining their own schema, instead of trying to modify existing schema(s) that contain tests that do not apply to their product.

Another advantage of this structure is that, as new software is supported by OVAL, previously written content using higher-level component schemas does not have to be modified. For example, in previous versions of OVAL, schemas were broken down by operating system and there was no concept of a higher-level UNIX schema, therefore, although the file_test was identical for the Red Hat, Solaris, and Debian platforms, an instance of the test had to be included in each schema. In some cases, this breakdown caused multiple, identical OVAL Definitions to be written to account for each of the platforms. To further compound the problem, addition of a new UNIX-based platform, such as HP-UX, would cause a new duplicate test to be created. Under the current approach, a single definition can be written to encompass all UNIX-based platforms, even those that are added after the definition has been created.

Finally, breaking the OVAL Language into component schemas allows tools to target a select set of applications or OSs and in doing so, reduce their process overhead and execution time. In versions of OVAL prior to 4.1, tools were forced to manage a single, large schema for their platform of choice. This sometimes meant that the schema included tests irrelevant to the tool, and increased the processing overhead.

The potential pitfall to this approach is a schema structure that is unwieldy and difficult for a user to navigate. Therefore, great care must be taken to ensure that considerable thought is put into the rules governing the structure of the schema, the rules are followed consistently, and the rules are well documented. By providing the user with a logical, fully documented format, they should be able to quickly locate any test within the schema.

Placement of a New Test

When a new test is created, there are a couple of questions that need to be answered in order to determine its placement in the language hierarchy. Does this test meet the needs of examining a specific software component on its own, or is there a collection of related new tests whose function is to describe the component in question? Is this test targeted at a specific application, or does it address a more general configuration issue?

The answer to the first question will determine whether the test should be incorporated into an existing component schema, or grouped with other new tests to form a new component schema. For example, a new test that examines the configuration of a specific Windows service will be incorporated into the existing Windows schema. A set of five new tests that examine the configuration of an Oracle Database would form the basis for a standalone Oracle schema.

The answer to the second question determines which component schema a test will be placed in. Tests that can be used more generally across a set of schemas are placed in a higher-level component schema, while tests that are more focused on a specific aspect of a system's configuration will be placed further down the conceptual hierarchy.

As with any decision of this type, there is bound to be debate over the merits of creating a new schema or not, and whether a test is truly general or not. In such cases, the members of the OVAL Community are encouraged to present their arguments, with the OVAL Board acting as the final arbiter. The goal is that this discussion will be incorporated into the documentation for the test, as well as provide precedents to refine the process in the future.

Relation to the Affected Family

In versions prior to 4.1 of the OVAL Language, definitions were grouped by the value of the affected family. The list of values for the affected family, and hence the possible grouping, was tied to the different component schemas. Each valid affected family corresponded to an existing component schema. This relationship has been removed from version 4.1 and later. For example, it would not make sense to have an affected family of Apache. Note that

the affected family is meant to describe the family of OSs that the OVAL Definition applies to. The purpose of the affected family has remained the same but the values are now declared independently of the component schemas.

The OVAL Language Review Process

The purpose of the OVAL Language Review Process is to ensure that all of the members of the OVAL Community have an opportunity to provide input in the development and direction of the OVAL Language. It also provides OVAL tool developers with a set of milestones to assist them in planning their own development schedules to maintain compatibility with the OVAL Language.

The entire process is managed by the OVAL Moderator, who is an individual or an organization that maintains OVAL and provides impartial technical guidance to the OVAL Board on all matters related to the ongoing development of OVAL. A representative from The MITRE Corporation currently serves as the OVAL Moderator.

The timeline associated with the review process will vary depending upon whether the planned modifications will result in a major or minor language version change. A major version change will require more effort in each stage of the process than a minor version change, and therefore, the overall time it will take to move from the planning stage to the release stage will be longer. The difference between a major and minor version change is outlined later in this document.

Planning

The OVAL Moderator begins the process of gathering suggestions and comments from the OVAL Community for evolving the existing OVAL Language. The OVAL Board reviews the suggestions and determines which ones should be considered in the new version of the OVAL Language. The length of this period is based upon the number, extent, and urgency of the proposed changes.

Draft/Internal Review

A new version of the OVAL Language is officially proposed to the OVAL Community for consideration as the next version. The OVAL Community is expected to review the schema and to propose additions, deletions, and modifications, all of which are further reviewed by the community. During this period the OVAL Moderator coordinates the testing of the draft language and updates to any OVAL maintained tools, in order to ensure that the proposed changes to the language are valid and usable.

Release Candidate

The OVAL Board has determined that the proposed OVAL Language has reached a level of consensus within the OVAL Community, and the OVAL Moderator has verified that the language is valid. In the release candidate stage, the language remains frozen for a period of time determined by the OVAL Board. It is during this stage that vendors and tool developers can update their tools with the knowledge that the schema will remain stable. Subsequent release candidates may be released if a serious problem is discovered in the proposed language.

Official

The OVAL Web site is updated to comply with the new version of the OVAL Language, including all OVAL Definitions, and the OVAL Interpreter. The previous schema files and its associated elements are then archived on the OVAL Web site.

OVAL Language Versioning

Since each OVAL Definition is validated against the OVAL Language, some sort of version information is needed to ensure that the correct version of the language is used for validation. This is necessary as older content may no longer validate against newer versions of the language, and recent content may not validate against previous

versions of the language. The language version also helps tools identify the OVAL content that they can process correctly.

The three different categories of the OVAL Language (Definitions, System Characteristics, and Results) each contain multiple files, making versioning a challenge. Changes in one category will usually result in similar changes to the other categories, while changes within a component schema do not normally result in changes to the other component schemas. This dynamic brings up the question: If there is a change to the Windows schema, should the version of the Red Hat schema change?

Previous Solutions

This section is intended to provide a discussion about the different approaches to language versioning that have been tried and/or thought about in previous versions of OVAL. Each approach is presented with a set of advantages and disadvantages. The hope is that by documenting the previous solutions, similar mistakes can be avoided in the future.

Single Language Version

One idea for language versioning was to use a single version for the entire OVAL Language and when a change is made to any of the component schemas, regardless of the language category, the version of all the schema files are updated. For example, if the current language is at version 2 and a new Windows test is added to the OVAL Definition schema, the version for each and every individual schema within the OVAL Language (including the Red Hat Systems Characteristics schema and the Debian core schema) would change to version 3.

The argument for a single language version was that when dealing with XML validation, the content either validates or does not validate, so there is no difference between major and minor changes. Unfortunately, this was not a good argument as there are changes, specifically additions to the language, which do not invalidate existing content.

An advantage of this approach is its simplicity. There is only one version number, which is very easy to track and manage from a project standpoint. Therefore, a tool is not required to account for all the versions of the different schema files that make up a definition. For example, a Windows definition would not have to maintain the core schema version and the Windows schema version required to validate it. A single version also makes it easy for tools to determine if they can talk to each other: "Do you speak Version 2? Yes. Great!"

If the advantage to the single language version approach is its simplicity, then it is also its main disadvantage. The biggest issue becomes how to update the existing language with new tests and how to add new component schemas. With a single language version, any addition would require movement to a new language version. Unfortunately, due to the nature of the language, the rapid changes that are made within computer software, and the community-based development effort, this would cause new versions to be made too frequently. Most of the time, the changes would not affect existing content, as the addition of a new test would not cause existing OVAL Definitions to be invalid. Many members of the community argued that because of this, a small change should not have such a global effect on the language version.

In addition, modifications to only one of the language categories, e.g., the System Characteristics schema, but not to the others would still result in a new version of the other categories. OVAL Community members or tools that are only concerned with OVAL Definitions will be confused when a set of definitions becomes invalid, even though no change has been made to the OVAL Definition schema. Why does my tool have to be upgraded to a new OVAL Language version when nothing has changed?

In short, the single schema version plan would require a number of version changes. It is a reality that rapid, global version changes will not be understood by the community and will give the impression of an unstable language.

Point System

Another approach considered for versioning the OVAL Language has been called "The Point System." Here, the core schemas from the different categories are given the same integer version number – for example, version 2. As

part of the major version of the schema, each component schema will be given a minor version number. Note that the same minor version is shared by the corresponding component schemas in the different categories. For example, the initial release of version 2 of the OVAL Language will contain version 2.0 of the Windows component schema. If there is a modification to the Windows component schema, then the new version in all language categories will be 2.1. The other component schemas (Red Hat, Apache, etc.) will remain at version 2.0. If the core schema is updated to version 3, then all the component schema versions are updated to 3.0.

The major version of the OVAL Language advances when a change is made to any part of the schema files that would invalidate any existing content. Changes to the schema files that do not invalidate previous content (for example, the addition of new tests) will cause a minor version change to the specific component schema in which the change is made. Since the core schema has only a single version number and not a pointed number, any change (including additions) to the core schema files will require a major version change.

The advantage to this approach is that language changes that do not invalidate the current content (e.g., inclusion of a new test), would not require an update of the major version and would instead represent a minor version change. The major version number only changes when there is a change to the core schema, or when a component schema changes in such a way that it invalidates the existing OVAL content.

This allows OSs and applications to be less bound by the release schedule of the OVAL Language. New OS and application versions that require new OVAL Tests can add them and force only minor version changes within their specific component schema. Also, if an existing test needs to be modified, which would normally require a major version change, a new copy of this test can be created instead and given a new name (e.g., new_registry_test) while keeping the original test in the schema. This is similar to how new APIs are added to programming languages.

The potential problem with this approach is that it may cause some confusion among the OVAL Community about why the Windows definitions use version 2.1 and the Solaris definitions use version 2.2, when the OVAL Web site states that the current language version is 2. With proper knowledge of the OVAL Language and help pages on the OVAL documentation, this concern can be mitigated.

Another more significant drawback is depending upon the rate at which minor changes are released, this could require tools to track numerous versions of language against definitions written for those versions. For example, a tool would need to keep track of the versions of the core schema files, the Windows component schema files, the Oracle component schema files, and so on.

Dated Version

This option has a single major version attached to the core schema files and uses a date string to version the component schemas. It is similar to the point system described in the previous section, and suffers from the same set of drawbacks. The only added benefit is that the date at which the schema was modified becomes evident by looking at the version.

Accepted OVAL Ideology

Single-Point Version

The accepted OVAL ideology for language versioning is a combination of the single language version and the point system. Every schema file is bound by the same version number as in the "Single Language Version" approach but the version contains both a major and minor component. These major and minor components are what allow changes to the schema to be classified as either major or minor. For example, the addition of a test in the Solaris component schema will cause a version change to every schema file but because the addition of a test does not invalidate existing content, then only the minor portion of the version will increment. More detail about what constitutes a major version change and a minor version change is described later in this document.

There is some concern that the accepted ideology forces users to update all of their schema files when a new version is released, even if it is only a minor release, affecting a component schema that they currently do not use. This is a valid concern and although no mitigation strategy exists it is worth noting that it is standard practice with the release

of new versions of programming languages, such as C or Java, that developers retrieve the entire language regardless of the extent or location of changes made.

Balancing this concern is the simplicity gained by having a single language version for the collection of files in the OVAL Language, making it easier for the OVAL Community to track the changes made to the OVAL Language. Note that another advantage of this ideology is that it mitigates the disadvantage of the point system that required storing all of the versions associated with the different schema files in the definition.

What Constitutes a Language Version Change

Whenever a modification is made to the OVAL Language, the version of all the schema files must change. Modifications that do not invalidate OVAL Content that were written for previous versions of the current major release, result in a minor version change. Meanwhile, modifications that do invalidate older content result in a major version change.

For example, the addition of a new component schema, a new test in an existing schema, or even a new child element in an existing test, do not invalidate the content according to the previous major version release of the language, and therefore result in a minor version change. By contrast, the deletion of an existing test or the renaming of an existing attribute, will invalidate previously written content, and therefore, will result in a major version change.

There are cases where a major version release can be deferred. This could be done by changing the inner-workings of a test in a minor release while still supporting the previous version of the test, or by creating a new version of the test with a different name. This technique is used frequently in programming APIs where a new version of the API will be written with '_ex' or some other moniker appended to the original name. The original API is still available, but developers are encouraged to use the new extended version. This same technique can be used with the OVAL Language although use of the new test will not only be encouraged, it will be forced at the next major release of the language. Each initial major release will remove all old versions of tests and rename the newer version back to the original name. This policy will be in place to keep the meaning of the names given to elements significant and not full of outdated names that have been passed down from previous versions.

Differentiating Language Versions via Namespace

To differentiate content that has been written in previous versions of the language, the language major version will be represented in the namespace associated with the schema. For example the core definition schema might have a namespace of "http://oval.mitre.org/XMLSchema/oval-4" and the Windows definition schema might be "http://oval.mitre.org/XMLSchema/oval-4#windows".

By including the major version in the namespace, tools will be able to identify the correct major version of the language to use in validation, and therefore be able to support multiple major versions of the language at the same time.

Only the major version is included in the namespace, since including the minor version would cause all existing content to become invalid with the release of a new minor version. In addition, any namespace aware application would break with the release of a new minor version forcing tool vendors to update their code.

Minor versions of the language are differentiated using the schema_version element found in an instance document. This does mean that any application trying to determine the correct minor version will have to first parse the XML document, causing a potential performance penalty.

Community Participation

The OVAL Language is industry-endorsed via the OVAL Board and OVAL Developers Forum, ensuring that the language reflects the combined expertise of the broadest possible group of security and system administration professionals worldwide.

The OVAL Board includes members from major operating system vendors, commercial information security tool vendors, academia, government agencies, and research institutions. Other information security experts will be invited to participate on the Board on an as-needed basis based upon recommendations from Board members. Archives of Board meetings and discussions are available for review and comment on the OVAL Web site.

The MITRE Corporation currently maintains the OVAL Language and provides impartial technical guidance to the OVAL Board on all matters related to the ongoing development of the language. In partnership with government, MITRE is an independent, not-for-profit corporation working in the public interest. It addresses issues of critical national importance, combining systems engineering and information technology to develop innovative solutions that make a difference.

OVAL-Compatible Products and Services

The OVAL Compatibility Program is an attempt to develop consistency within the security community regarding the use and implementation of OVAL (both the language and the repositories). The compatibility program's main goal is to create a set of guidelines that will help enforce a standard implementation. An off shoot of this is that users are able to distinguish between, and have confidence in, compatible products knowing that the implementation of OVAL coincides with the standard set forth.

In terms of the language, OVAL Compatibility means that a tool, service, Web site, database, or advisory/alert incorporates the OVAL Language in a pre-defined and standard way. An OVAL-Compatible product uses the OVAL Language as appropriate for communicating details of vulnerabilities, patches, security configuration settings, and other machine states.

For more information about OVAL Compatibility, including the process for becoming OVAL-Compatible, please visit the OVAL Web site at http://oval.mitre.org.

Conclusion

With a stated revision process, a clearly laid out versioning scheme, and a well defined relationship between the individual schema files, the OVAL Community will be better able to incorporate the OVAL Language into their tools and business procedures. As the language continues to change and evolve, and the schema development process is further refined, this document will be updated accordingly to ensure that the OVAL Community has access to this information. For answers to any further questions, please visit the OVAL Web site at http://oval.mitre.org or contact the OVAL Team at MITRE via the e-mail address oval@mitre.org.