Security Automation and Continuous Monitoring
Internet-Draft
Intended status: Informational
Expires: March 11, 2017

M. Cokus
D. Haynes
D. Rothenberg
The MITRE Corporation
J. Gonzalez
Department of Homeland Security

September 7, 2016

OVAL(R) Processing Model
draft-haynes-sacm-oval-processing-model-01

Abstract

This document defines Version 5.11.1 of the OVAL processing model which describes, in detail, how the major components of the OVAL Language Data Model are used to produce OVAL Definitions, OVAL System Characteristics, and OVAL Results.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 11, 2017.

Table of Contents

1. Introduction

The Open Vulnerability and Assessment Language (OVAL) [OVAL-WEBSITE] is an international, information security community effort to standardize how to assess and report upon the machine state of systems. For over ten years, OVAL has been developed in collaboration with any and all interested parties to promote open and publicly available security content and to standardize the representation of this information across the entire spectrum of security tools and services.

OVAL provides an established framework for making assertions about an system's state by standardizing the three main steps of the assessment process: representing the current machine state; analyzing the system for the presence of the specified machine state; and representing the results of the assessment which facilitates collaboration and information sharing among the information security community and interoperability among tools.

This draft is part of the OVAL contribution to the IETF SACM WG that standardizes how to produce OVAL Definitions, OVAL System Characteristics, and OVAL Results. It is intended to serve as a starting point for how SACM can carry out the assessment of a system in a standardized way.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Producing OVAL Definitions

Producing OVAL Definitions is the process by which information from some source external to OVAL is consumed by a person, tool, or service and then transformed into an OVAL Definition. Often this information comes from a security advisory, configuration checklist, or other data feed. Other times this information must be created through detailed system investigation and research of known issues. In either case, low level system state information is encoded in the form of an assertion about a system state.

2.1. Reuse of Definition, Test, Object, State, and Variable

The OVAL Language enables content reuse through the use of globally unique IDs. When producing OVAL Definitions, OVAL Tests, OVAL Objects, OVAL States, and OVAL Variables, existing content SHOULD be reused when possible.

2.1.1. Tracking Change

The version property provides the ability to track changes to OVAL Definitions, OVAL Tests, OVAL Objects, OVAL States, and OVAL Variables. Proper usage of the version property is critical for content sharing and reuse. When updating an OVAL Definition, OVAL Test, OVAL Object, OVAL State, or OVAL Variable the version property MUST be incremented for each revision.

2.1.2. Metadata

Each OVAL Definition, as defined by the oval-def:DefinitionType, includes a metadata property. The contents of the metadata property MUST NOT impact OVAL Definition evaluation. All information that is encoded in the metadata property SHOULD also be encoded in the OVAL Definition's criteria.

2.1.2.1. Authoritative References

The reference property of an OVAL Definition's metadata property SHOULD provide an authoritative citation for the specific system state being described by the OVAL Definition. OVAL Definitions with a class property value of 'vulnerability' SHOULD include a reference to the CVE Name for the vulnerability when one exists. OVAL Definitions with a class property value of 'compliance' SHOULD include a reference to the CCE Name for the configuration item when one exists. OVAL Definitions with a class property value of 'inventory' SHOULD include a reference to the CPE for the relevant operating system or application when a CPE Name exists.

2.1.2.2. Platforms and Products

The platform and product properties of an OVAL Definition's metadata property SHOULD provide a listing of platforms and products to which the OVAL Definition is known to apply.

2.1.3. Content Integrity and Authenticity

Content expressed in the OVAL Definitions Model MAY be digitally signed in order to preserve content integrity and authenticity. The

OVAL Definitions Model defines six locations for including a digital signature. Any of these locations MAY be used.

3. Producing OVAL System Characteristics

Producing OVAL System Characteristics is the process by which detailed system state information is collected and represented in a standard format. This information may be collected through direct interaction with an end system by using system APIs to query the state of the system, or by gathering information from some other source of system state information, like a configuration management database.

3.1. System Information

The oval-sc:system_info property of the OVAL System Characteristics model MUST accurately represent the system from which the data was collected. When the system data was collected from a source other than directly from the system being described, the oval-sc:system_info type MUST represent the original system from which the data was collected.

3.2. Collected Objects

When a set of OVAL Objects is used to guide the collection of system data, the OVAL Objects that were used MUST be recorded as objects in the oval-sc:collected_objects property of the OVAL System Characteristics model. This section describes the process of creating an oval-sc:object in the collection of oval-sc:collected_objects.

3.2.1. flag Usage

Each object listed in the oval-sc:collected_objects MUST specify the outcome of the data collection effort by setting the flag property to the appropriate value. The valid flag values are defined in the oval-sc:FlagEnumeration. The correct usage of the flag enumeration values in the context of the flag property is specified in the following table.

Value	Result
error	This value MUST be used when an error that prevents the collection of the OVAL Items for the OVAL Object. The object property SHOULD include one or more messages describing the error condition.
complete	This value MUST be used when the collection process for the OVAL Object was successful and accurately captured the complete set of matching OVAL Items.
incomplete	This value MUST be used when the collection process for the OVAL Object was successful but the complete set of matching OVAL Items is not represented by the set of references. The object property SHOULD include one or more messages explaining the incomplete flag value.
does not exist	This value MUST be used when no matching OVAL Items were found.
not collected	This value MUST be used when no attempt was made to collect the OVAL Object. The object property MAY include one or more messages explaining the not collected flag value.
not applicable	This value MUST be used the specified OVAL Object is not applicable to the system under test. The object property MAY include one or more messages explaining the not applicable flag value.

Table 1: flag Usage for Collected Objects

3.2.2. variable_instance property

When an OVAL Object makes use of an OVAL Variable, either directly or indirectly, each collection of values assigned to the OVAL Variable MUST be differentiated by incrementing the variable_instance property once for each assigned collection of values for the OVAL Variable. When more than one collection of values is assigned to an OVAL Variable, a given OVAL Object will appear as a oval-sc:collected_object once for each assigned value.

3.2.3. Item References

Each OVAL Item that is collected as a result of collecting a given OVAL Object MUST be referenced by the reference property of the object. A given OVAL Item MAY be referenced by one or more objects. This situation will occur when two distinct OVAL Objects identify overlapping sets of OVAL Items.

When the flag property has a value of 'not collected' or 'not applicable' the object MUST NOT include any OVAL Item references.

3.2.4. Variable Values

Each OVAL Variable and its value used when collecting OVAL Items for an OVAL Object MUST be recorded in the variable_value property of the object.

3.3. Conveying System Data without Objects

OVAL Objects are commonly used to guide the collection of OVAL Items. However, system state information may be collected without the use of OVAL Objects. OVAL Items MAY be collected by searching system data stores, API calls, algorithms, or other proprietary processes. When this is done, the OVAL System Characteristics will not contain a collected_objects section, however, it will contain a system_data section with all of the OVAL Items collected.

3.4. Recording System Data and OVAL Items

The system_data property holds a collection of OVAL Items. This section describes the process of building an OVAL Item and the constraints that apply to OVAL Items.

3.4.1. Item IDs

Each OVAL Item contains a unique identifier which distinguishes it from other OVAL Items that are represented in the collection of system_data. Item IDs MUST be unique within an OVAL System Characteristics Model.

3.4.2. Unique Items

OVAL Items are differentiated by examining each OVAL Item's name and each of the OVAL Item's entity names and values. Each OVAL Item MUST represent a unique system data artifact. No two OVAL Items within an OVAL System Characteristics Model can be the same.

3.4.3. Partial Matches

A partial match is when an OVAL Item, containing some information, is reported in the OVAL System Characteristics rather than simply not reporting the OVAL Item. Partial matches are useful for debugging purposes when an OVAL Item does not exist on the system or is not collected due to limitations in the OVAL Capable Product. Please note that the use of partial matches is optional.

3.4.4. Item Status

The valid status values, for an OVAL Item, are defined in the oval-sc:StatusEnumeration. The correct usage of the status enumeration values in the context of the status property is specified in the following table.

Value	Result
error	This value MUST be used when there is an error that prevents the collection of an OVAL Item or any of its entities. The OVAL Item SHOULD include one or more messages describing the error condition.
exists	This value MUST be used when an OVAL Item is successfully collected.
does not exist	This value MUST be used when the OVAL Item is not found on the system being examined. The use of this value is optional and is only used to report a partial match. If a partial match is not being reported, the OVAL Item MUST NOT be reported in the OVAL System Characteristics. The OVAL Item MAY include one or more messages describing this status value.
not collected	This value MUST be used when no attempt is made collect the OVAL Item. The use of this value is optional and is only used to report a partial match. If a partial match is not being reported, the OVAL Item MUST NOT be reported in the OVAL System Characteristics. The OVAL Item SHOULD include one or more messages describing this status value.

Table 2: Item Status

3.4.5. Item Entities

OVAL Item Entities must be added to the OVAL Item such that it aligns with the constraints specified in the appropriate OVAL Component Model and the requirements in this section.

3.4.5.1. Determining which Entities to Include

OVAL Component Models define concrete OVAL Items and their entities. All entities within an OVAL Item are optional. When creating an OVAL Item any number of item entities MAY be included. However, sufficient OVAL Item entities MUST be included to ensure that the OVAL Item describes only a single system configuration item.

Many OVAL Items include entities that have dependencies upon other entities within the same OVAL Item. When dependencies exist between OVAL Item entities, if an entity is included then all entities that it depends upon MUST also be included in the OVAL Item. When using OVAL Objects to guide the collection of system data, the entities included in the OVAL Object SHOULD be included in the OVAL Items that it identifies.

When collecting system data an OVAL State MAY be used to determine which entities to include within and OVAL Item. This sort of processing can be an optimization made when collecting data. For example, if the OVAL State makes an assertion about a single entity it may not be necessary to include all other OVAL Item entities.

3.4.5.2. Status

The OVAL Item Entity status conveys the outcome of attempting to collect one property of a piece of system state information. The valid OVAL Item Entity status values are defined in the oval-sc:StatusEnumeration. The status of an OVAL Item Entity can be independent of other OVAL Item Entities and SHOULD NOT be propagated up to the containing OVAL Item. The following table indicates when to use each status value.

Value	Result
error	This value MUST be used when there is an error that prevents the collection of the OVAL Item Entity.
exists	This value MUST be used when the OVAL Item Entity exists on the system and is collected.
does not exist	This value MUST be used when the OVAL Item Entity does not exist on the system.
not collected	This value MUST be used when no attempt is made to collect the OVAL Item Entity.

Table 3: Item Entity Status

3.4.5.3. Datatype

The datatype of the OVAL Item Entity describes how the value of the OVAL Item Entity should be interpreted. The valid datatype values for an OVAL Item Entity are listed in the oval:DatatypeEnumeration and restricted as needed in OVAL Component Models. When assigning a datatype to an OVAL Item Entity, there are two cases to consider:

1. The datatype is fixed to a specific datatype value. In this case, the OVAL Item Entity MUST always use the specified datatype value.

2. The datatype can be one of several datatype values. In this case, the datatype value that most appropriately describes the value of the OVAL Item Entity SHOULD be used. If an OVAL Item Entity value is not present, the datatype value must be set to the default datatype value specified in corresponding OVAL Component Model.

3.4.5.4. Value

The final aspect of an OVAL Item Entity is its value. An OVAL Item Entity may contain simple character data or complex structured data as specified in the corresponding OVAL Component Model. All OVAL Item Entity values must conform to the constraints defined in the oval-sc:DatatypeEnumeration.

3.4.6. Content Integrity and Authenticity

Content expressed in the OVAL System Characteristics Model MAY be digitally signed in order to preserve content integrity and authenticity.

4. Producing OVAL Results

Producing OVAL Results is the process by which detailed system state information is evaluated against the expected state of a system and represented in a standardized format. This standardized format conveys the results of the evaluation which can indicate the presence of a vulnerability, compliance to a policy, installation of software, or even the presence of malware artifacts. Additionally, the results can be consumed by other tools where they can be interpreted and used to inform remediation of discovered issues.

4.1. Definition Evaluation

OVAL Definition Evaluation is the process examining the characteristics of a system and applying one or more logical statements about those characteristics to determine an overall result for the system state that the OVAL Definition describes. Each OVAL Definition has zero or one logical criteria components, which are combined using logical operators, such as 'AND' and 'OR'. The overall result of evaluating an OVAL Definition is determined by evaluating its criteria component. This process is described in detail in the following section.

4.1.1. Evaluating a Deprecated OVAL Definition

When evaluating a deprecated OVAL Definition, that does not have a criteria construct, the OVAL Definition MUST evaluate to 'not evaluated'. If a deprecated OVAL Definition contains a criteria construct, the OVAL Definition SHOULD evaluate as if it were not deprecated. However, the OVAL Definition MAY evaluate to 'not evaluted'.

4.1.2. Criteria Evaluation

A criteria component of an OVAL Definition combines one or more logical statements in order to determine a result value. A criteria can be made up of other criteria, criterion, or extend_definitions, along with an operator property that specifies how to logically combine the specified logical statements. For more information on how to combine the individual results of the logical statements specified within a criteria, see Section 4.5.3.2.4.2. The result value of the criteria is determined by first evaluating the operator

property to combine the logical statements and then evaluating the negate property. See Section 4.1.5 for additional information on how to negate the result of the criteria.

4.1.2.1. applicablity_check

If a value for the applicability_check property is specified on the criteria construct, in an OVAL Definition, the applicability_check property and value MUST be replicated on the criteria construct in the OVAL Results.

4.1.3. Criterion Evaluation

The result of a criterion construct is the result of the OVAL Test that it references, after the negate property has been applied. See Section 4.1.5 for additional information on how to negate the result of an OVAL Test.

The variable_instance property of the criterion is carried over from the variable_instance value of the referenced OVAL Test.

4.1.3.1. applicability_check

If a value for the applicability_check property is specified on the criterion construct, in an OVAL Definition, the applicability_check property and value MUST be replicated on the criterion construct in the OVAL Results.

4.1.4. Extend Definition Evaluation

The result of an extend_definition construct is the result of the OVAL Definition, that it references, after the negate property has been applied. See Section 4.1.5 for additional information on how to negate the result of an OVAL Definition.

The variable_instance property of the extend_definition is carried over from the variable_instance value of the referenced OVAL Definition.

4.1.4.1. applicability_check

If a value for the applicability_check property is specified on the extend_definition construct, in an OVAL Definition, the applicability_check property and value MUST be replicated on the extend_definition construct in the OVAL Results.

4.1.5. Negate Evaluation

When the negate property is 'true', the final result of a construct MUST be the logical complement of its result value. That is, for any construct that evaluates to 'true', the final result would become 'false', and vice versa. The negate property does not apply to non-Boolean result values. If a non-Boolean result value is encountered, the final result MUST be the non-Boolean result value. If the negate property is set to 'false', the final result of a construct will be its original result value.

4.1.6. Variable Instance

The value of the variable_instance property is derived from the variable_instance values of the OVAL Definitions and OVAL Tests that are referenced within the OVAL Definition's criteria. When an OVAL Definition references another OVAL Definition or an OVAL Test that makes use of an OVAL Variable, each collection of values assigned to the OVAL Variable MUST be differentiated by incrementing the variable_instance property. The variable_instance value is incremented once for each assigned collection of values for the OVAL Variable. When more than one collection of values is assigned to an OVAL Variable, an OVAL Definition will appear in the definitions section once for each assigned collection of values.

4.2. Test Evaluation

An OVAL Test is the standardized representation of an assertion about the state of a system. An OVAL Test contains references to an OVAL Object that specifies which system data to collect and zero or more OVAL States that specify the expected state of the collected system data. OVAL Test Evaluation is the process of comparing the collected set of system data, as OVAL Items, to zero or more OVAL States.

The result of the OVAL Test Evaluation is then determined by combining the results of the following three test evaluation parameters:

1. Existence Check Evaluation - The process of determining whether or not the number of OVAL Items, that match the specified OVAL Object, satisfy the requirements specified by the check_existence property.

2. Check Evaluation - The process of determining whether or not the number of collected OVAL Items, specified by the check property, match the specified OVAL States.

3. State Operator Evaluation - The process of combining the individual results, from the comparison of an OVAL Item to the specified OVAL States, according to the state_operator property.

4.2.1. Existence Check Evaluation

Existence Check Evaluation is the process of determining whether or not the number of OVAL Items, that match the specified OVAL Object, satisfy the requirements specified by the check_existence property. The check_existence property specifies how many OVAL Items that match the specified OVAL Object must exist on the system in order for the OVAL Test to evaluate to 'true'. To determine if the check_existence property is satisfied, the status of each OVAL Item collected by the OVAL Object must be examined.

The following tables describe how each ExistenceEnumeration value affects the result of the Existence Check Evaluation. The far left column identifies the ExistenceEnumeration value in question, and the middle column specifies the different combinations of individual OVAL Item status values that may be found (EX = exist; DE = does not exist; ER = error; NC = not collected). The last column specifies the final result of the Existence Check Evaluation according to the combination of individual OVAL Item status values.

item status value count				
EX	DE	ER	NC	existence piece is
1+	0	0	0	True
0	0	0	0	False
0+	1+	0+	0+	False
0+	0	1+	0+	Error
0+	0	0	1+	Unknown
--	--	--	--	Not Evaluated
--	--	--	--	Not Applicable

Figure 1: Existence Check Evaluation for 'all exist'

item status value count				
				existence piece is
EX	DE	ER	NC	
0+	0+	0	0+	True
1+	0+	1+	0+	True
--	--	--	--	False
0	0+	1+	0+	Error
--	--	--	--	Unknown
--	--	--	--	Not Evaluated
--	--	--	--	Not Applicable

Figure 2: Existence Check Evaluation for 'any exist'

item status value count				
				existence piece is
EX	DE	ER	NC	
1+	0+	0+	0+	True
0	1+	0	0	False
0	0+	1+	0+	Error
0	0+	0	1+	Unknown
--	--	--	--	Not Evaluated
--	--	--	--	Not Applicable

Figure 3: Existence Check Evaluation for 'at least one exists'

item status value count				
existence piece is				
EX	DE	ER	NC	
0	0+	0	0	True
1+	0+	0+	0+	False
0	0+	1+	0+	Error
0	0+	0	1+	Unknown
--	--	--	--	Not Evaluated
--	--	--	--	Not Applicable

Figure 4: Existence Check Evaluation for 'none exist'

item status value count				existence piece is
EX	DE	ER	NC	
1	0+	0	0	True
2+	0+	0+	0+	False
0	0+	0	0	False
0,1	0+	1+	0+	Error
0,1	0+	0	1+	Unknown
--	--	--	--	Not Evaluated
--	--	--	--	Not Applicable

Figure 5: Existence Check Evaluation for 'only one exists'

4.2.2. Check Evaluation

Check Evaluation is the process of determining whether or not the number of collected OVAL Items, specified by the check property, match the specified OVAL States. The check property specifies how many of the collected OVAL Items must match the specified OVAL States in order for the OVAL Test to evaluate to 'true'. For additional information on how to determine if the check property is satisfied, see Section 4.5.3.2.4.1.

4.2.3. State Operator Evaluation

State Operator Evaluation is the process of combining the individual results, from the comparison of an OVAL Item to the specified OVAL States, according to the state_operator property, to produce a result for the OVAL Test. For additional information on how to determine the final result using the state_operator property, see Section 4.5.3.2.4.2.

4.2.4. Determining the Final OVAL Test Evaluation Result

While the final result of the OVAL Test Evaluation is the combination of the results from the three evaluations (Existence Check Evaluation, Check Evaluation, and State Operator Evaluation), how the result is calculated will vary depending upon if the optional collected object section is present in the OVAL System Characteristics. However, in either case, if the result of the Existence Check Evaluation is 'false', the Check and State Operator Evaluations can be ignored and the final result of the OVAL Test will be 'false'.

4.2.4.1. Final OVAL Test Evaluation Result without a Collected Objects Section

When the Collected Objects section is not present in the OVAL System Characteristics, all OVAL Items present in the OVAL System Characteristics must be examined. Each OVAL Item MUST be examined to determine which match the OVAL Object according to Section 4.3.1 and Section 4.3.2. Once the set of matching OVAL Items is determined, they can undergo the three different evaluations that make up OVAL Test Evaluation.

4.2.4.2. Final OVAL Test Evaluation Result with a Collected Objects Section

When the Collected Objects section is present in the OVAL System Characteristics the flag value of an OVAL Object, in the Collected Objects section, must be examined before the Existence Check Evaluation is performed.

If the OVAL Object, referenced by an OVAL Test, cannot be found in the Collected Objects section, the final result of the OVAL Test MUST be 'unknown'.

Otherwise, if the OVAL Object, referenced by an OVAL Test, is found, the following guidelines must be followed when determining the final result of an OVAL Test.

- If the flag value is 'error', the final result of the OVAL Test MUST be 'error'.
- If the flag value is 'not collected', the final result of the OVAL Test MUST be 'unknown'.
- If the flag value is 'not applicable', the final result of the OVAL Test MUST be 'not applicable'.
- If the flag value is 'does not exist', the final result is determined solely by performing the Check Existence Evaluation.
- If the flag value is 'complete', the final result is determined by first performing the Check Existence Evaluation followed by the Check Evaluation and State Operator Evaluation.
- If the flag value is 'incomplete', the final result is determined as follows:
  - If the check_existence property has a value of 'none_exist' and one or more OVAL Items, referenced by the OVAL Object, have a

status of 'exists', the final result of the OVAL Test MUST be 'false'.

* If the check_existence property has a value of 'only one exists' and more than one OVAL Item, referenced by the OVAL Object, has a status of 'exists', the final result of the OVAL Test MUST be 'false'.

* If the result of the Existence Check Evaluation is true, the following special cases during the Check Evaluation MUST be considered:

  + If the Check Evaluation evaluates to 'false', the final result of the OVAL Test MUST be 'false'.

  + If the check property has a value of 'at least one satisfies' and the check evaluation evaluates to 'true', the final result of the OVAL Test MUST be 'true'.

* Otherwise, the final result of the OVAL Test MUST be 'unknown'.

Value	Result
error	error
complete	Depends on check_existence and check attributes
incomplete	Depends on check_existence and check attributes
does not exist	depends on check_existence and check attributes
not collected	unknown
not applicable	not applicable

Table 4: Mapping between oval-sc:FlagEnumeration Value and Test Result

4.2.5. Variable Instance

When an OVAL Test makes use of an OVAL Variable, either directly or indirectly, OVAL Test is evaluated once for each collection of values assigned to the OVAL Variable. Each evaluation result for the OVAL Tests MUST be differentiated by incrementing the variable_instance property once for each assigned collection of values for the OVAL Variable. When more than one collection of values is assigned to an

OVAL Variable, an OVAL Test will appear in the tests section once for each assigned collection of values.

4.3. OVAL Object Evaluation

At the highest level, OVAL Object Evaluation is the process of collecting OVAL Items based on the constraints specified by the OVAL Object Entities and OVAL Behaviors, if present, in an OVAL Object. An OVAL Object contains the minimal number of OVAL Object Entities needed to uniquely identify the system state information that makes up the corresponding OVAL Item. The methodology used to collect the system state information for the OVAL Items is strictly an implementation detail. Regardless of the chosen methodology, the same OVAL Items MUST be collected on a system for a given OVAL Object except when the flag for the collected OVAL Object has a value of 'incomplete'.

4.3.1. Matching an OVAL Object to an OVAL Item

An OVAL Item matches an OVAL Object only if every OVAL Object Entity, as guided by any OVAL Behaviors, matches the corresponding OVAL Item Entity in the OVAL Item under consideration.

4.3.2. Matching an OVAL Object Entity to an OVAL Item Entity

An OVAL Object Entity matches an OVAL Item Entity only if the value of the OVAL Item Entity matches the value of the OVAL Object Entity in the context of the specified datatype and operation. See Section Section 4.5.3.2.4.3 for additional information regarding the allowable datatypes, operations, and how they should be interpreted.

4.3.3. OVAL Object Entity Evaluation

OVAL Object Entity Evaluation is the process of searching for system state information that matches the values of an OVAL Object Entity in the context of the specified datatype and operation. This process is further defined below.

4.3.3.1. Datatype and Operation Evaluation

The datatype and operation property associated with an OVAL Object Entity specifies what system state information should be collected from the system in the form of an OVAL Item. When comparing a value specified in the OVAL Object Entity against system state information, the operation must be performed in the context of the specified datatype; the same operation for two different datatypes could yield different results. See Section 4.5.3.2.4.3 for additional

information on how to apply an operation in the context of a particular datatype.

4.3.3.2. nil Object Entities

For many OVAL Object Entities, there are situations in which the OVAL Object Entity does not need to be considered in the evaluation of the OVAL Object. When the nil property is set to 'true', it indicates that the OVAL Object Entity must not be considered during OVAL Object Evaluation and must not be collected. For more information about a particular OVAL Object Entity and how the nil property affects it, see the appropriate OVAL Component Model.

4.3.3.3. Referencing an OVAL Variable

An OVAL Variable may be referenced from an Object Entity in order to specify multiple values or to use a value that was collected from some other source. When the var_ref property is specified, the var_check property SHOULD also be specified. See Section 4.5.3.2.4.3.1.1 for more information on how to evaluate an OVAL Object Entity that references a variable.

In addition to the OVAL Item Entity value matching the values specified in the OVAL Variable according to the var_check property, the flag associated with the OVAL Variable must also be considered. The OVAL Variable flag indicates the outcome of the collection of values for the OVAL Variable. It is important to consider this outcome because it may affect the ability of an OVAL Object Entity to successfully match the corresponding OVAL Item Entity. Additionally, this flag will also impact the collected object flag.

The following table describes what flags are valid given the flag value of the OVAL Variable referenced by an OVAL Object Entity.

OVAL Variable Flag	Valid OVAL Object Flags
error	error
complete	error, complete, incomplete, does not exist, not collected, not applicable
incomplete	error, incomplete, does not exist, not collected, not applicable
does not exist	does not exist
not collected	does not exist
not applicable	does not exist

Table 5: Valid Flag Values Given the Referenced OVAL Variable Flag

For additional information on when each flag value MUST be used, see Section 3.2.1.

4.3.3.4. Collected Object Flag Evaluation

However, when there are multiple OVAL Object Entities in an OVAL Object the flag values for each OVAL Object Entity must be considered when determining which flag values are appropriate. The following table describes how multiple flag values influence the collected object flag of the OVAL Object referencing the variable (ER = error; CO = complete; IN = incomplete; DE = does not exist; NC = not collected; NA = not applicable;).

Resulting Flag	OVAL Component Flag Count					
	ER	CO	IN	DE	NC	NA
error	1+	0+	0+	0+	0+	0+
complete	0	1+	0	0	0	0
incomplete	0	0+	1+	0	0	0
does not exist	0	0+	0+	1+	0	0
not collected	0	0+	0+	0+	1+	0
not applicable	0	0+	0+	0+	0+	1+

Figure 6: Collected Object Flag Evaluation

4.3.4. Set Evaluation

The set construct provides the ability to combine the collected OVAL Items of one or two OVAL Objects using the set operators defined in the SetOperatorEnumeration. See Section 4.3.4.1 for more information about the allowed set operators.

The processing of a set MUST be done in the following manner:

1. Identify the OVAL Objects that are part of the set by examining the object_references associated with the set. Each object_reference will refer to an OVAL Object that describes a unique set of collected OVAL Items.

2. For every defined filter Section 4.3.4.2, apply the associated filter to each OVAL Item.

3. Apply the set operator to all OVAL Items remaining in the set.

4. The resulting OVAL Items will be the unique set of OVAL Items referenced by the OVAL Object that contains the set.

4.3.4.1. Set Operator

Set operations are used to combine multiple sets of different OVAL Items, as identified by the object_reference and limited by any filter, into a single unique set of OVAL Items. The different operators that guide process are in the SetOperatorEnumeration. For each operator, if only a single object_reference has been supplied then the resulting set is simply the complete set of OVAL Items identified by the referenced OVAL Object after any included filters have been applied.

The tables below explain how different flags are combined for each set_operator to return a new flag. These tables are needed when computing the flag for collected objects that represent object sets in an OVAL Definition. The top row identifies the flag associated with the first set or object reference. The left column identifies the flag associated with the second set or object reference. The matrix inside the table represents the resulting flag when the given set_operator is applied. (E=error, C=complete, I=incomplete, DNE=does not exist, NC=not collected, NA=not applicable)

set_operator is COMPLEMENT		Object 1 Flag					
	E	C	I	DNE	NC	NA	
Object 2 Flag	E	E	E	E	DNE	E	E
	C	E	C	I	DNE	NC	E
	I	E	E	E	DNE	NC	E
	DNE	E	C	I	DNE	NC	E
	NC	E	NC	NC	DNE	NC	E
	NA	E	E	E	E	E	E

Figure 7: set_operator = COMPLEMENT

set_operator is INTERSECTION		Object 1 Flag					
	E	C	I	DNE	NC	NA	
Object 2 Flag	E	E	E	E	DNE	E	E
	C	E	C	I	DNE	NC	C
	I	E	I	I	DNE	NC	I
	DNE	DNE	DNE	DNE	DNE	DNE	DNE
	NC	E	NC	NC	DNE	NC	NC
	NA	E	C	I	DNE	NC	NA

Figure 8: set_operator = INTERSECTION

set_operator is UNION	Object 1 Flag					
	E	C	I	DNE	NC	NA
Object 2 Flag: E	E	E	E	E	E	E
Object 2 Flag: C	E	C	I	C	I	C
Object 2 Flag: I	E	I	I	I	I	I
Object 2 Flag: DNE	E	C	I	DNE	I	DNE
Object 2 Flag: NC	E	I	I	I	NC	NC
Object 2 Flag: NA	E	C	I	DNE	NC	NA

Figure 9: set_operator = UNION

4.3.4.2. Filter

The filter construct provides a way to control the OVAL Items that are included a set. See Section 4.3.5 for additional information.

4.3.4.3. object_reference

When evaluating an object_reference, an error MUST be reported it the OVAL Object identifier is invalid, the referenced OVAL Object does not exist, or the referenced OVAL Object does not align with the OVAL Object that is referring to it.

4.3.5. OVAL Filter Evaluation

An OVAL Filter is a mechanism that provides the capability to either include or exclude OVAL Items based on their system state information. This is done through the referencing of an OVAL State that specifies the requirements for a matching OVAL Item and the action property that states whether or not the matching OVAL Items will be included or excluded.

When evaluating an OVAL Filter, an error MUST be reported if the OVAL State identifier is not legal, the referenced OVAL State does not exist, or the referenced OVAL State does not align with the OVAL Object where it is used.

The action property specifies whether or not the matching OVAL Items will be included or excluded. The action property enumeration values are defined in Section the ArithmeticEnumeration in [I-D.draft-haynes-sacm-oval-definitions-model].

4.3.5.1. Applying Multiple Filters

When multiple OVAL Filters are specified, they MUST be evaluated sequentially from first to last to the collection of OVAL Items under consideration.

4.3.6. OVAL Object Filter

When applying a filter to OVAL Objects, every collected OVAL Item is compared to the OVAL State referenced by the OVAL Filter. If the collected OVAL Items match the OVAL State they are included or excluded based on the action property. The final set of collected OVAL Items is the set of collected OVAL Items after each OVAL Filter is evaluated. See Section 4.3.5.

4.4. OVAL State Evaluation

The OVAL State is the standardized representation for expressing an expected machine state. In the OVAL State each OVAL State Entity expresses the expected value(s) for a single piece of configuration information. OVAL State Evaluation is the process of comparing a specified OVAL State against a collected OVAL Item on the system. OVAL State Evaluation can be broken up into two distinct parts:

1. State Entity Evaluation - The process of determining whether or not an OVAL Item Entity, in a collected OVAL Item, matches the corresponding OVAL State Entity specified in an OVAL State.

2. State Operator Evaluation - The process of combining the individual results, from the comparison of an OVAL Item Entity against the specified OVAL State Entity, according to the operator property.

4.4.1. OVAL State Entity Evaluation

OVAL State Entity Evaluation is the process of comparing a specified OVAL State Entity against the corresponding collected OVAL Item Entities. This comparison must be done in the context of the datatype and operation, whether or not an OVAL Variable is referenced, and whether or not there are multiple occurrences of the corresponding OVAL Item Entity in the collected OVAL Item.

4.4.1.1. Datatype and Operation Evaluation

The datatype and operation property associated with an OVAL State Entity specifies how the collected OVAL Item Entity compares to the value(s) specified in the OVAL State Entity. When comparing a value specified in the OVAL State Entity against a collected OVAL Item

Entity, the operation must be performed in the context of the specified datatype. See Section 4.5.3.2.4.3.1 for additional information on how an operation is applied in the context of a particular datatype.

4.4.1.2. var_check Evaluation

An OVAL Variable can be referenced from an OVAL State Entity to specify multiple values that the corresponding OVAL Item Entities will be compared against or to utilize a value that was collected from some other source. For information on how to evaluate an OVAL State Entity that references an OVAL Variable, see Section 4.5.3.2.4.3.1.1.

4.4.1.3. entity_check Evaluation

An OVAL Item may contain multiple occurrences of an OVAL Item Entity to represent that the OVAL Item has multiple values for that particular OVAL Item Entity. The entity_check property specifies how many occurrences of an OVAL Item Entity MUST match the OVAL State Entity, as defined in Section 4.4.1, in order to evaluate to 'true'. The valid values for the entity_check property are defined by the CheckEnumeration. See Section 4.5.3.2.4.1 for more information about how to apply the property.

4.4.1.4. Determining the Final Result of an OVAL State Entity Evaluation

The final result of an OVAL State Entity Evaluation is determined by first comparing the value specified in the OVAL State Entity with each occurrence of a corresponding OVAL Item Entity, in an OVAL Item, in the context of the specified datatype and operation as defined in Section 4.4.1.1. The results of the comparisons are evaluated against the specified entity_check property according to Section 4.5.3.2.4.1. This will be the final result of the OVAL State Entity Evaluation unless an OVAL Variable was also referenced.

If an OVAL Variable was referenced, the above procedure must be performed for each value in the OVAL Variable. The final result must then be computed by examining the var_check property and the individual results for each OVAL Variable value comparison. See Section 4.5.3.2.4.3.1.1.

4.4.2. Operator Evaluation

Once the OVAL State Entity Evaluation is complete for every OVAL State Entity, the individual results from each evaluation MUST be combined according to the operator property specified on the OVAL

State. The combined result will be the final result of the OVAL State Evaluation. See Section 4.5.3.2.4.2 for more information on applying the operator to the individual results of the evaluations.

4.5. OVAL Variable Evaluation

OVAL Variable Evaluation is the process of retrieving a collection of values from sources both local and external to OVAL Definitions as well as manipulating those values through the evaluation of OVAL Functions. OVAL Variables can be used in OVAL Definitions to specify multiple values, manipulate values, retrieve values at execution time, and create generic and reusable content.

4.5.1. Constant Variable

A constant_variable is a locally defined collection of one or more values that are specified prior to evaluation time.

4.5.1.1. Determining the Flag Value

A constant_variable is only capable of having a flag value of 'error', 'complete', or 'not collected'. The flag values of 'does not exist' and 'incomplete' are not used for the evaluation of a constant_variable because a constant variable is required to contain at least one value. The flag value of 'not applicable' is not used because the constant_variable construct is platform independent. The following table outlines when a constant variable will evaluate to each of the flag values.

Value	Description
error	This flag value must be used when one or more values do not conform to the specified datatype as defined in the oval:DatatypeEnumeration.
complete	This flag value must be used when all values conform to the specified datatype and the collection of constant variables is supported in the OVAL-capable product.
incomplete	-
does not exist	-
not collected	This flag value must be used when the tool does not support the collection of constant_variables.
not applicable	-

Table 6: When a constant_variable Evaluates to a Specific oval-sc:FlagEnumeration Value

4.5.2. External Variable

An external_variable is a locally declared, externally defined, collection of one or more values. The values referenced by an external_variable are collected from the external source at run-time.

4.5.2.1. Validating External Variable Values

The OVAL Language provides the PossibleValueType and PossibleRestriction constructs as a mechanism to validate input coming from sources external to the OVAL Definitions.

4.5.2.1.1. Possible Restriction

The possible_restriction construct specifies one or more restrictions on the values of an external variable. When more than one restriction is used the individual results of each comparison between the restriction and the external variable value must be combined using the selected operator attribute. The default operation performed is 'AND'. See Section 4.5.3.2.4.2 for more information on how to combine the individual results. The final result, after

combining the individual results, will be the result of the possible_restriction construct.

4.5.2.1.1.1. Restriction

Each restriction allows for the specification of an operation and a value that will be compared to a supplied value for the external_variable. The result of this comparison will be used in the computation of the final result of the possible_restriction construct. See Section 4.5.2.1.3 for additional information on how to determine the result of the comparison between the specified value and the external variable value using the specified operation in the context of the datatype specified on the external_variable.

4.5.2.1.2. Possible Value

The possible_value construct specifies a permitted external variable value. The specified value and the external variable value must be compared as string values using the equals operation. See Section 4.5.2.1.3 for additional information on how to determine the result of the comparison. The result of this comparison will be used in determining the final result of validating an external variable value.

4.5.2.1.3. Determining the Final Result of Validating an External Variable Value

The final result of validating an external_variable value is determined by combining every possible_restriction and possible_value constructs using the logical 'OR' operator. That is, each value in the external_variable will be evaluated against the combination of possible_restriction and possible_value constructs and the results of this evaluation will be combined using the 'OR' operator. See Section 4.5.3.2.4.2 for more information on how to combine the individual results using the 'OR' operator.

4.5.2.2. Determining the Flag Value

An external variable is only capable of returning a flag value of 'error', 'complete', or 'not collected'. The following table outlines when an external variable will evaluate to each of the flag values. The flag values 'does not exist' and 'incomplete' are not used because an external_variable is required to contain at least one value. The flag value of 'not applicable' is not used because the external_variable construct is platform independent.

Value	Description
error	This flag value must be used when one or more values do not conform to the specified datatype as defined in the oval:DatatypeEnumeration. This flag value must be used when there was an error collecting the values from the external source. This flag value must be used when there is a value, collected from the external source, that does not conform to the restrictions specified by the possible_value and possible_restriction constructs or if there is an error processing the possible_value and possible_restriction constructs. This flag value must be used when the final result of validating the external variable values is not 'true'. This flag must be used when the external source for the variable cannot be found.
complete	This flag value must be used when the final result of validating every external variable value is 'true' and conforms to the specified datatype.
incomplete	-
does not exist	-
not collected	This flag value must be used when the tool does not support the collection of constant_variables.
not applicable	-

Table 7: When a external_variable Evaluates to a Specific oval-sc:FlagEnumeration Value

4.5.3. Local Variable

A local_variable is a locally defined collection of one or more values that may be composed of values from other sources collected at evaluation time.

4.5.3.1. OVAL Function Evaluation

An OVAL Function is a construct, in the OVAL Language, that takes one or more collections of values and manipulates them in some defined way. The result of evaluating an OVAL Function will be zero or more values.

4.5.3.1.1. Nested Functions

Due to the recursive nature of the ComponentGroup construct, OVAL Functions can be nested within one another. In this case, a depth-first approach is taken to processing OVAL Functions. As a result, the inner most OVAL Functions are evaluated first, and then the resulting values are used as input to the outer OVAL Function and so on.

4.5.3.1.2. Evaluating OVAL Functions with Sub-components with Multiple Values

When one or more of the specified sub-components resolve to multiple values, the function will be applied to the Cartesian product of the values, in the sub-components, and will result in a collection of values.

4.5.3.1.3. Casting the Input of OVAL Functions

OVAL Functions are designed to work on values with specific datatypes. If an input value is encountered that does not align with required datatypes an attempt must be made to cast the input value(s) to the required datatype before evaluating the OVAL Function. If the input value cannot be cast to the required datatype the flag value, of the OVAL Function, MUST be set to 'error'.

4.5.3.1.4. Determining the Flag Value

When determining the flag value of an OVAL Function, the combined flag value of the sub-components must be computed in order to determine if the evaluation of the OVAL Function should continue. The following tables outline how to combine the sub-component flag values.

Notation	Description
x	x individual OVAL Component flag values are...
x,y	x or y individual OVAL Component flag values are...

Table 8: Flag Value Table Notation

num of components with flag						resulting flag is
E	C	I	DNE	NC	NA	
1+	0+	0+	0+	0+	0+	Error
0	1+	0	0	0	0	Complete
0	0+	1+	0	0	0	Incomplete
0	0+	0+	1+	0	0	Does Not Exist
0	0+	0+	0+	1+	0	Not Collected
0	0+	0+	0+	0+	1+	Not Applicable

Figure 10: Determining the Flag Value for an OVAL Function

Once the flag values of the sub-components have been combined the evaluation of an OVAL Function must only continue if the flag value is 'complete'. All other flag values mean that the evaluation of the OVAL Function stops and the flag of the OVAL Function MUST be 'error'. The following table outlines how to determine the flag value of an OVAL Function.

Value	Description
error	This flag value must be used if the combined sub-component flag is a value other than 'complete'.This flag value must be used if an error occurred during the computation of an OVAL Function. This flag value must be used if an attempt to cast an input value to a required datatype failed.
complete	This flag value must be used if the combined sub-component flag is complete and the evaluation of the OVAL Function completes successfully.
incomplete	-
does not exist	-
not collected	-
not applicable	-

Table 9: When a OVAL Function Evaluates to a Specific oval-sc:FlagEnumeration Value

4.5.3.2. OVAL Components

A component is a reference to another part of the content that allows further evaluation or manipulation of the value or values specified by the referral.

4.5.3.2.1. Literal Component

A literal_component is a component that allows the specification of a literal value. The value can be of any supported datatype as specified in the oval:DatatypeEnumeration. The default datatype is 'string'.

4.5.3.2.1.1. Determining the Flag Value

A literal_component is only capable of evaluating to a flag value of 'error' or 'complete'. The flag values 'does not exist' and 'incomplete' are not used because an external_variable is required to contain at least one value. The flag value of 'not applicable' is

not used because the literal_component construct is platform independent. The following table outlines when a literal_component will evaluate to each of the flag values.

Value	Description
error	This flag value must be used when the value does not conform to the specified datatype as defined in the oval:DatatypeEnumeration.
complete	This flag value must be used when the value conforms to the specified datatype as defined in the oval:DatatypeEnumeration.
incomplete	-
does not exist	-
not collected	-
not applicable	-

Table 10: When a Literal Component Evaluates to a Specific oval-sc:FlagEnumeration Value

4.5.3.2.2. Object Component

An object component is a component that resolves to the value(s) of OVAL Item Entities or OVAL Fields, in OVAL Items, that were collected by an OVAL Object. The property, object_ref, must reference an existing OVAL Object.

The value that is used by the object component must be specified using the item_field property of the object component. This indicates which entity should be used as the value for the component. In the case that the OVAL Object collects multiple OVAL Items as part of its evaluation, this can resolve to a collection of values. In the case that an OVAL Item Entity has a datatype of 'record', the record_field property can be used to indicate which field to use for the component.

4.5.3.2.2.1. Determining the Flag Value

An object_component is only capable of evaluating to a flag value of 'error', 'complete', 'incomplete', or 'not collected'. The flag values 'does not exist' and 'incomplete' are not used because an object_component is required to contain at least one value. The following table outlines when an object_component will evaluate to each of the flag values.

Value	Description
error	This flag value must be used when the value does not conform to the specified datatype as defined in the oval:DatatypeEnumeration. This flag value must be used if the OVAL Object does not return any OVAL Items. This flag value must be used if an entity is not found with a name that matches the value of the item_field property. This flag value must be used if a field is not found with a name that matches the value of the record_field property.
complete	This flag value must be used when every value conforms to the specified datatype as defined in the oval:DatatypeEnumeration and when the flag of the referenced OVAL Object is 'complete'.
incomplete	This flag value must be used when every value conforms to the specified datatype as defined in the oval:DatatypeEnumeration and when the flag of the referenced OVAL Object is 'incomplete'.
does not exist	-
not collected	This flag value must be used when the OVAL-capable product does not support the collection of object_components.
not applicable	-

Table 11: When a Object Component Evaluates to a Specific oval-sc:FlagEnumeration Value

4.5.3.2.3. Variable Component

An variable component is a component that resolves to the value(s) of the referenced OVAL Variable. The property, var_ref, must reference an existing OVAL Variable.

4.5.3.2.3.1. Variable Component Flag Value

A variable_component is only capable of evaluating to a flag value of 'error', 'complete', 'incomplete', or 'not collected'. The following table outlines when a variable_component will evaluate to each of the flag values.

Value	Description
error	This flag value must be used when the flag value of the referenced OVAL Variable is 'error'. This flag value must be used when the referenced OVAL Variable cannot be found.
complete	This flag value must be used when the flag value of the referenced OVAL Variable is 'complete'.
incomplete	This flag value must be used when the flag value of the referenced OVAL Variable is 'incomplete'.
does not exist	This flag value must be used when the flag value of the referenced OVAL Variable is 'does not exist'.
not collected	This flag value must be used when the OVAL-capable product does not support the collection of variable_components.
not applicable	-

Table 12: Determining the Flag Value

4.5.3.2.3.1.1. Determining the Flag Value

A local_variable can contain an OVAL Function or an OVAL Component. As a result, the flag value must consider both the flag of the OVAL Function or OVAL Component along with the additional conditions from being an OVAL Variable. The following table describes when each flag value must be used.

Value	Description
error	This flag value must be used when one or more values do not conform to the specified datatype as defined in the oval:DatatypeEnumeration. This flag value must be used when there was an error collecting the values from the external source. This flag value must be used when the specified datatype is 'record'. This flag value must be used when the flag value of the specified OVAL Function or OVAL Component is 'error'.
complete	This flag value must be used when the flag value of the specified OVAL Function or OVAL Component is 'complete' and every value conforms to the specified datatype.
incomplete	-
does not exist	This flag value must be used when the flag value of the referenced OVAL Variable is 'does not exist'.
not collected	This flag value must be used when there are no values.
not applicable	-

Table 13: When a Local Variable Component Evaluates to a Specific oval-sc:FlagEnumeration Value

4.5.3.2.4. Common Evaluation Concepts

This section describes a set of evaluation concepts that apply to several aspects of producing OVAL Content.

4.5.3.2.4.1. Check Enumeration Evaluation

Check Enumeration Evaluation is the process of determining whether or not the number of individual results, produced from the comparison of some set of values, satisfies the specified CheckEnumeration value.

The following tables describe how each CheckEnumeration value affects the final result of an evaluation. The far left column identifies the CheckEnumeration value in question. The middle column specifies the different combinations of individual results that the

CheckEnumeration value may bind together. The last column specifies the final result according to each combination of individual results. It is important to note that if an individual result is negated, then a 'true' result is 'false' and a 'false' result is 'true', and all other results stay as is.

check attr	num of individual results						final result is
	T	F	E	U	NE	NA	
ALL	1+	0	0	0	0	0+	True
	0+	1+	0+	0+	0+	0+	False
	0+	0	1+	0+	0+	0+	Error
	0+	0	0	1+	0+	0+	Unknown
	0+	0	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 11: Check Enumeration Evaluation for 'all'

check attr	num of individual results						final result is
	T	F	E	U	NE	NA	
	1+	0+	0+	0+	0+	0+	True
	0	1+	0	0	0	0+	False
AT	0	0+	1+	0+	0+	0+	Error
LEAST	0	0+	0	1+	0+	0+	Unknown
ONE	0	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 12: Check Enumeration Evaluation for 'at least one'

check attr	num of individual results						final result is
	T	F	E	U	NE	NA	
	1	0+	0	0	0	0+	True
	2+	0+	0+	0+	0+	0+	** False **
	0	1+	0	0	0	0+	** False **
ONLY	0,1	0+	1+	0+	0+	0+	Error
ONE	0,1	0+	0	1+	0+	0+	Unknown
	0,1	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 13: Check Enumeration Evaluation for 'only one'

check attr	num of individual results: T	F	E	U	NE	NA	final result is
NONE SATISFY	0	1+	0	0	0	0+	True
	1+	0+	0+	0+	0+	0+	False
	0	0+	1+	0+	0+	0+	Error
	0	0+	0	1+	0+	0+	Unknown
	0	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 14: Check Enumeration Evaluation for 'none satisfy'

4.5.3.2.4.2. Operator Enumeration Evaluation

Operator Enumeration Evaluation is the process of combining the individual results of evaluations using logical operations. The following table shows the notation used when describing the number of individual results that evaluate to a particular result.

Notation	Description
x	x individual results are...
x,y	x or y individual results are...
x+	x or more individual results are...
Odd	an odd number of individual results are...
Even	an even number of individual results are...

Table 14: Operator Value Table Notation

The following tables describe how each OperatorEnumeration value affects the final result of an evaluation. The far left column identifies the OperatorEnumeration value in question. The middle column specifies the different combinations of individual results that the OperatorEnumeration value may bind together. The last column specifies the final result according to each combination of individual results. It is important to note that if an individual result is negated, then a 'true' result is 'false' and a 'false' result is 'true', and all other results stay as is.

operator is	num of individual results						final result is
	T	F	E	U	NE	NA	
AND	1+	0	0	0	0	0+	True
	0+	1+	0+	0+	0+	0+	False
	0+	0	1+	0+	0+	0+	Error
	0+	0	0	1+	0+	0+	Unknown
	0+	0	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 15: Operator Enumeration Evaluation for 'AND'

operator is	num of individual results						final result is
	T	F	E	U	NE	NA	
ONE	1	0+	0	0	0	0+	True
	2+	0+	0+	0+	0+	0+	** False **
	0	1+	0	0	0	0+	** False **
	0,1	0+	1+	0+	0+	0+	Error
	0,1	0+	0	1+	0+	0+	Unknown
	0,1	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 16: Operator Enumeration Evaluation for 'ONE'

operator is	num of individual results						final result is
	T	F	E	U	NE	NA	
OR	1+	0+	0+	0+	0+	0+	True
	0	1+	0	0	0	0+	False
	0	0+	1+	0+	0+	0+	Error
	0	0+	0	1+	0+	0+	Unknown
	0	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 17: Operator Enumeration Evaluation for 'OR'

operator is	num of individual results						final result is
	T	F	E	U	NE	NA	
XOR	odd	0+	0	0	0	0+	True
	even	0+	0	0	0	0+	False
	0+	0+	1+	0+	0+	0+	Error
	0+	0+	0	1+	0+	0+	Unknown
	0+	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

Figure 18: Operator Enumeration Evaluation for 'XOR'

4.5.3.2.4.3. OVAL Entity Evaluation

OVAL Entity Evaluation is the process of comparing the specified value(s), from an OVAL Object or State Entity, against the corresponding system state information in the context of the selected datatype and operation.

4.5.3.2.4.3.1. Datatype and Operation Evaluation

The result of applying an operation in the context of a specified datatype MUST evaluate to 'true' only if the values being compared satisfy the conditions of the operation for the specified datatype. If the values being compared do not satisfy the conditions of the operation, the final result MUST be 'false'.

To ensure consistency in the comparison of the value(s) specified in the OVAL Object and State Entities with the system state information, the operations for each datatype must be defined. The following table describes how each operation must be performed in the context of a specific datatype.

Value	Description
binary	Data of this type conforms to the W3C Recommendation for hex-encoded binary data [W3C-HEX-BIN]. equals: The collected binary value is equal to the specified binary value only if the collected binary value and the specified binary value are the same length and the collected binary value and the specified binary value contain the same

	characters in the same positions.	
	not equal: The collected binary value is not equal to the specified binary value only if the collected binary value is not the same length as the specified binary value or the collected binary value and specified binary value do not contain the same characters in the same positions.	
boolean	Data of this type conforms to the W3C Recommendation for boolean data [W3C-BOOLEAN] (f = false; t = true;).	
	equals:	

Specified Value	Collected Value: f / 0	Collected Value: t / 1
f / 0	t	f
t / 1	f	t

	not equal:	

Specified Value	Collected Value: f / 0	Collected Value: t / 1
f / 0	f	t
t / 1	t	f

debian_evr_string	Data of this type conforms to the format EPOCH:UPSTREAM_VERSION-DEBIAN_REVISION and comparisons involving this type MUST follow the algorithm described in Chapter 5 (Section 5.6.12) "Debian Policy Manual" [DEBIAN-POLICY-MANUAL]. One implementation of this is the cmpversions function which is located in dpkg's enquiry.c.	
	equals: The collected debian_evr_string value c is equal to the specified debian_evr_string	

		value s only if the result of the algorithm described in the cmpversions function is 0.
		not equal: The collected debian_evr_string value c is not equal to the specified debian_evr_string value s only if the result of the algorithm described in the cmpversions function is -1 or 1.
		greater than: The collected debian_evr_string value c is greater than the specified debian_evr_string value s only if the result of the algorithm described in the cmpversions function is 1.
		greater than or equal: The collected debian_evr_string value c is greater than or equal to the specified debian_evr_string value s only if the result of the algorithm described in the cmpversions function is 1 or 0.
		less than: The collected debian_evr_string value c is less than the specified debian_evr_string value s only if the result of the algorithm described in the cmpversions function is -1.
		less than or equal: The collected debian_evr_string value c is less than or equal to the specified debian_evr_string value s only if the result of the algorithm described in the cmpversions function is -1 or 0.
evr_ string	Data of this type conforms to the format EPOCH:VERSION-RELEASE and comparisons involving this type MUST follow the algorithm described in the rpmVersionCompare() function which is located in lib/psm.c of the RPM source code.	
		equals: The collected evr_string value c is equal to the specified evr_string value s only if the result of the algorithm described in the rpmVersionCompare(c,s) function is 0.
		not equal: The collected evr_string value c

		is not equal to the specified evr_string value s only if the result of the algorithm described in the rpmVersionCompare(c,s) function is -1 or 1. greater than: The collected evr_string value c is greater than the specified evr_string s value only if the result of the algorithm described in the rpmVersionCompare(c,s) function is 1. greater than or equal: The collected evr_string value c is greater than or equal to the specified evr_string value s only if the result of the algorithm described in the rpmVersionCompare(c,s) function is 1 or 0. less than: The collected evr_string value c is less than the specified evr_string value s only if the result of the algorithm described in the rpmVersionCompare(c,s) function is -1. less than or equal: The collected evr_string value c is less than or equal to the specified evr_string value s only if the result of the algorithm described in the rpmVersionCompare(c,s) function is -1 or 0.
fileset_ revision		Data of this type conforms to the version string related to filesets in HP-UX. An example would be 'A.03.61.00'. Please note that this needs further community review and discussion.
float		Data of this type conforms to the W3C Recommendation for float data [W3C-FLOAT]. equals: The collected float value is equal to the specified float value only if the collected float value and the specified float value are numerically equal. not equal: The collected float value is not equal to the specified float value only if the collected float value and the specified float value are not numerically equal.

	greater than: The collected float value is greater than the specified float value only if the collected float value is numerically greater than the specified float value. greater than or equal: The collected float value is greater than or equal to the specified float value only if the collected float value is numerically greater than or equal to the specified float value. less than: The collected float value is less than the specified float value only if the collected float value is numerically less than the specified float value. less than or equal: The collected float value is less than or equal to the specified float value only if the collected float value is numerically less than or equal to the specified float value.
ios_version	Data of this type conforms to Cisco IOS Train strings. These are in essence version strings for IOS. Please refer to Cisco's IOS Reference Guide for information on how to compare different Trains as they follow a very specific pattern [CISCO-IOS]. Please note that this needs further community review and discussion.
int	Data of this type conforms to the W3C Recommendation for integer data [W3C-INT]. equals: The collected integer value is equal to the specified integer value only if the collected integer value and the specified integer value are numerically equal. not equal: The collected integer value is not equal to the specified integer value only if the collected integer value and the specified integer value are not numerically equal. greater than: The collected integer value is greater than the specified integer value only

	if the collected integer value is numerically greater than the specified integer value. greater than or equal: The collected integer value is greater than or equal to the specified integer value only if the collected integer value is numerically greater than or equal to the specified integer value. less than: The collected integer value is less than the specified integer value only if the collected integer value is numerically less than the specified integer value. less than or equal: The collected integer value is less than or equal to the specified integer value only if the collected integer value is numerically less than or equal to the specified integer value. bitwise and: The collected integer satisfies the bitwise and operation with the specified integer value only if the result of performing the bitwise and operation on the binary representation of the collected integer value and the binary representation of the specified integer value is the binary representation of the specified value. bitwise or: The collected integer satisfies the bitwise or operation with the specified integer value only if the result of performing the bitwise or operation on the binary representation of the collected integer value and the binary representation of the specified integer value is the binary representation of the specified value.
ipv4_address	The ipv4_address [RFC791] datatype represents IPv4 addresses and IPv4 address prefixes. Its value space consists of the set of ordered pairs of integers where the first element of each pair is in the range [0,2^32) (the representable range of a 32-bit unsigned int), and the second is in the range [0,32]. The first element is an address, and the second is a prefix length.

	The lexical space is dotted-quad CIDR-like notation ('a.b.c.d' where 'a', 'b', 'c', and 'd' are integers from 0-255), optionally followed by a slash ('/') and either a prefix length (an integer from 0-32) or a netmask represented in the dotted-quad notation described previously. Examples of legal values are '192.0.2.0', '192.0.2.0/32', and '192.0.2.0/255.255.255.255'. Additionally, leading zeros are permitted such that '192.0.2.0' is equal to '192.000.002.000'. If a prefix length is not specified, it is implicitly equal to 32.
	All operations are defined in terms of the value space. Let A and B be ipv4_address values (i.e. ordered pairs from the value space). The following definitions assume that bits outside the prefix have been zeroed out. By zeroing the low order bits, they are effectively ignored for all operations. Implementations of the following operations MUST behave as if this has been done.
	Let P_addr mean the first element of ordered pair P and P_prefix mean the second element.
	equals: A equals B if and only if A_addr == B_addr and A_prefix == B_prefix.
	not equal: A is not equal to B if and only if they don't satisfy the criteria for operator "equals".
	greater than: A is greater than B if and only if A_prefix == B_prefix and A_addr > B_addr. If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported.
	greater than or equal: A is greater than or equal to B if and only if A_prefix == B_prefix and they satisfy either the criteria for operators "equal" or "greater than". If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported.

		less than: A is less than B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than or equal". If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported. less than or equal: A is less than or equal to B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than". If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported. subset of: A is a subset of B if and only if every IPv4 address in subnet A is present in subnet B. In other words, A_prefix >= B_prefix and the high B_prefix bits of A_addr and B_addr are equal. superset of: A is a superset of B if and only if B is a subset of A.
ipv6_address	The ipv6_address datatype represents IPv6 addresses and IPv6 address prefixes. Its value space consists of the set of ordered pairs of integers where the first element of each pair is in the range [0,2^128) (the representable range of a 128-bit unsigned int), and the second is in the range [0,128]. The first element is an address, and the second is a prefix length. The lexical space is CIDR notation given in IETF specification [RFC4291] for textual representations of IPv6 addresses and IPv6 address prefixes (see sections 2.2 and 2.3). If a prefix-length is not specified, it is implicitly equal to 128. All operations are defined in terms of the value space. Let A and B be ipv6_address values (i.e. ordered pairs from the value space). The following definitions assume that bits outside the prefix have been zeroed out. By zeroing the low order bits, they are effectively ignored for all operations. Implementations of the following operations	

	MUST behave as if this has been done. Let P_addr mean the first element of ordered pair P and P_prefix mean the second element. equals: A equals B if and only if A_addr == B_addr and A_prefix == B_prefix. not equal: A is not equal to B if and only if they don't satisfy the criteria for operator "equals". greater than: A is greater than B if and only if A_prefix == B_prefix and A_addr > B_addr. If A_prefix != B_prefix, an error MUST be reported. greater than or equal: A is greater than or equal to B if and only if A_prefix == B_prefix and they satisfy either the criteria for operators "equal" or "greater than". If A_prefix != B_prefix, an error MUST be reported. less than: A is less than B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than or equal". If A_prefix != B_prefix, an error MUST be reported. less than or equal: A is less than or equal to B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than". If A_prefix != B_prefix, an error MUST be reported. subset of: A is a subset of B if and only if every IPv6 address in subnet A is present in subnet B. In other words, A_prefix >= B_prefix and the high B_prefix bits of A_addr and B_addr are equal. superset of: A is a superset of B if and only if B is a subset of A.
string	Data of this type conforms to the W3C Recommendation for string data [W3C-STRING]. equals: The collected string value is equal

	to the specified string value only if the collected string value and the specified string value are the same length and the collected string value and the specified string value contain the same characters in the same positions. not equal: The collected string value is not equal to the specified string value only if the collected string value is not the same length as the specified string value or the collected string value and specified string value do not contain the same characters in the same positions. case insensitive equals: The collected string value is equal to the specified string value only if the collected string value and the specified string value are the same length and the collected string value and the specified string value contain the same characters, regardless of case, in the same positions. case insensitive not equal: The collected string value is not equal to the specified string value only if the collected string value and the specified string value are not the same length or the collected string value and the specified string value do not contain the same characters, regardless of case, in the same positions. pattern match: The collected string value will match the specified string value only if the collected string value matches the specified string value when the specified string is interpreted as a Perl 5 Compatible Regular Expression (PCRE)[PERL5]. The support for PCRE in OVAL is documented in the [I-D.draft-cokus-sacm-oval-common-model.xml].
version	Data of this type represents a value that is a hierarchical list of non-negative integers separated by a single character delimiter. Any single non-integer character may be used as a delimiter and the delimiter may vary between the non-negative integers of a given

	version value. The hierarchical list of non-negative integers must be compared sequentially from left to right. When the version values, under comparison, have different-length lists of non-negative integers, zeros must be appended to the end of the values such that the lengths of the lists of non-negative integers are equal.
	equals: The collected version value is equal to the specified version value only if every non-negative integer in the collected version value is numerically equal to the corresponding non-negative integer in the specified version value.
	not equal: The collected version value is not equal to the specified version value if any non-negative integer in the collected version value is not numerically equal to the corresponding non-negative integer in the specified version value.
	greater than: The collected version value c is greater than the specified version value s only if the following algorithm returns true:
	c = c1,c2,...,cn where , is any non-integer character
	s = s1,s2,...,sn where , is any non-integer character
	for i = 1 to n if ci > si return true if ci < si return false if ci == si if i != n continue else return false
	greater than or equal: The collected version value c is greater than or equal to the specified version value s only if the following algorithm returns true:

```
|          |                                                |
|          | c = c1,c2,...,cn where , is any non-integer    |
|          | character                                      |
|          |                                                |
|          | s = s1,s2,...,sn where , is any non-integer    |
|          | character                                      |
|          |                                                |
|          | for i = 1 to n                                 |
|          |   if  ci > si                              |
|          |     return true                                |
|          |   if ci < si                               |
|          |     return false                               |
|          |   if ci == si                                  |
|          |     if i != n                                  |
|          |       continue                                 |
|          |     else                                       |
|          |       return true                              |
|          |                                                |
|          | less than: The collected version value c is    |
|          | less than the specified version value s only   |
|          | if the following algorithm returns true:       |
|          |                                                |
|          | c = c1,c2,...,cn where , is any non-integer    |
|          | character                                      |
|          |                                                |
|          | s = s1,s2,...,sn where , is any non-integer    |
|          | character                                      |
|          |                                                |
|          | for i = 1 to n                                 |
|          |   if  ci < si                              |
|          |     return true                                |
|          |   if ci > si                               |
|          |     return false                               |
|          |   if ci == si                                  |
|          |     if i != n                                  |
|          |       continue                                 |
|          |   else                                         |
|          |     return false                               |
|          |                                                |
|          | less than or equal: The collected version      |
|          | value c is less than or equal to the           |
|          | specified version value s only if the          |
|          | following algorithm returns true:              |
|          |                                                |
|          | c = c1,c2,...,cn where , is any non-integer    |
|          | character                                      |
|          |                                                |
|          | s = s1,s2,...,sn where , is any non-integer    |
```

```
|          | character                                        |
|          |                                                  |
|          | for i = 1 to n                                   |
|          |  if  ci < si                                |    |
|          |     return true                                  |
|          |   if ci > si                                |    |
|          |     return false                                 |
|          |   if ci == si                                    |
|          |     if i != n                                    |
|          |       continue                                   |
|          |     else                                         |
|          |       return true                                |
+----------+--------------------------------------------------+
| record   | Data of this type describes an entity with       |
|          | structured set of named fields and values        |
|          | as its content. The record datatype is           |
|          | currently prohibited from being used on          |
|          | variables.                                       |
|          |                                                  |
|          | equals: The collected record value is equal      |
|          | to the specified record value only if each       |
|          | specified OVAL Field has a corresponding         |
|          | collected OVAL Field with the same name          |
|          | property and that the collected OVAL Field       |
|          | value matches the specified OVAL Field           |
|          | value in the context of the datatype and         |
|          | operation as described above.                    |
+-------------------------------------------------------------+
```

Figure 19: Evaluation with Respect to Datatype and Operation

4.5.3.2.4.3.1.1. Variable Check Evaluation

It is often necessary to reference a variable from an OVAL Object or State Entity in order to specify multiple values or to use a value that was collected at runtime. When an OVAL Variable is referenced from an OVAL Object or State Entity using the var_ref property, the system state information will be compared to the every OVAL Variable value in the context of the specified datatype and operation. The final result of these comparisons are dependent on the value of the var_check property which specifies how many of the values, contained in OVAL Variable, must match the system state information to evaluate to a result of 'true'. The valid values for the var_check property are the defined in the CheckEnumeration.

Value	Description
all	The OVAL Object or State Entity matches the system state information only if the value of the OVAL Item Entity matches all of the values in the referenced the OVAL Variable in the context of the datatype and operation specified in the OVAL Object or State Entity.
at least one	The OVAL Object or State Entity matches the system state information only if the value of the OVAL Item Entity matches one or more of the values in the referenced OVAL Variable in the context of the datatype and operation specified in the OVAL Object or State Entity.
none satisfy	The OVAL Object or State Entity matches the system state information only if the OVAL Item Entity matches zero of the values in the referenced OVAL Variable in the context of the specified datatype and operation.
does not exist	-
only one	The OVAL Object or State Entity matches the system state information only if the OVAL Item Entity matches one of the values in the referenced OVAL Variable in the context of the specified datatype and operation.

Table 15: Variable Check Evaluation

4.5.3.2.4.3.1.1.1. Determining the Final Result of the Variable Check Evaluation

For more detailed information on how to combine the individual results of the comparisons between the OVAL object or State Entities and the system state information to determine the final result of applying the var_check property, see Section 4.5.3.2.4.1.

4.5.3.2.4.3.1.2. OVAL Entity Casting

In certain situations, it is possible that the datatype specified on the OVAL Entity is different from the datatype of the system state information. When this happens, it is required that an attempt is made to cast the system state information to the datatype specified by the OVAL Entity before the operation is applied. If the cast is

unsuccessful, the final result of the OVAL Entity Evaluation MUST be 'error'. Otherwise, the final result is dependent on the outcome of the Datatype and Operation Evaluation and the Variable Check Evaluation if an OVAL Variable is referenced. The process of casting a value of one datatype to a value of another datatype must conform to Section 4.5.3.4.

4.5.3.3. Masking Data

When the mask property is set to 'true' on an OVAL Entity or an OVAL Field, the value of that OVAL Entity or OVAL Field MUST NOT be present in the OVAL Results. Additionally, the mask property MUST be set to 'true' for any OVAL Entity or OVAL Field or corresponding OVAL Item Entity or OVAL Field in the OVAL Results where the system state information was omitted.

When the mask property is set to 'true' on an OVAL Entity with a datatype of 'record', each OVAL Field MUST have its operation and value or value omitted from the OVAL Results regardless of the OVAL Field's mask property value.

It is possible for masking conflicts to occur where one entity has mask set to 'true' and another entity has mask set to 'false'. Such a conflict will occur when the mask attribute is set differently on an OVAL Object and OVAL State or when more than one OVAL Objects identify the same OVAL Item(s). When such a conflict occurs the value MUST always be masked.

Values MUST NOT be masked in OVAL System Characteristics that are not contained within OVAL Results.

4.5.3.4. Entity Casting

Casting is performed whenever the datatype of a value, used during evaluation, differs from the specified datatype whether it be on an OVAL Entity or an OVAL Function. In most scenarios, it will be possible to attempt the cast of a value from one datatype to another.

4.5.3.4.1. Attempting to Cast a Value

When attempting to cast a value from one datatype to another, the value under consideration must be parsed according to the specified datatype. If the value is successfully parsed according to the definition of the specified datatype in the oval:DatatypeEnumeration, this constitutes a successful cast. If the value is not successfully parsed according to the definition of the specified datatype, this means that it is not possible to cast the value to the specified

datatype and an error MUST be reported for the construct attempting to perform the cast.

4.5.3.4.2. Prohibited Casting

In some scenarios, it is not possible to perform a cast from one datatype to another due to the datatypes, under consideration, being incompatible. When an attempt is made to cast two incompatible datatypes, an error MUST be reported. The following outlines the casts where the datatypes are incompatible:

- An attempt to cast a value of datatype 'record' to any datatype other than 'record'.
- An attempt to cast a value of datatype 'ipv4_address' to any datatype other than 'ipv4_address' or 'string'.
- An attempt to cast a value of datatype 'ipv6_address' to any datatype other than 'ipv6_address' or 'string'.
- An attempt to cast a value with a datatype other than 'ipv4_address' or 'string' to 'ipv4_address'.
- An attempt to cast a value with a datatype other than 'ipv6_address' or 'string' to 'ipv6_address'.

5. Intellectual Property Considerations

DHS, on behalf of the United States, owns the registered OVAL trademarks, identifying the OVAL STANDARDS SUITE and any component part, as that suite has been provided to the IETF Trust. A "(R)" will be used in conjunction with the first use of any OVAL trademark in any document or publication in recognition of DHS's trademark ownership.

6. Acknowledgements

The authors wish to thank DHS for sponsoring the OVAL effort over the years which has made this work possible. The authors also wish to thank the original authors of this document Jonathan Baker, Matthew Hansbury, and Daniel Haynes of the MITRE Corporation as well as the OVAL Community for its assistance in contributing and reviewing the original document. The authors would also like to acknowledge Dave Waltermire of NIST for his contribution to the development of the original document.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

While OVAL is just a set of data models and does not directly introduce security concerns, it does provide a mechanism by which to represent endpoint posture assessment information. This information could be extremely valuable to an attacker allowing them to learn about very sensitive information including, but, not limited to: security policies, systems on the network, criticality of systems, software and hardware inventory, patch levels, user accounts and much more. To address this concern, all endpoint posture assessment information should be protected while in transit and at rest. Furthermore, it should only be shared with parties that are authorized to receive it.

Another possible security concern is due to the fact that content expressed as OVAL has the ability to impact how a security tool operates. For example, content may instruct a tool to collect certain information off a system or may be used to drive follow-up actions like remediation. As a result, it is important for security tools to ensure that they are obtaining OVAL content from a trusted source, that it has not been modified in transit, and that proper validation is performed in order to ensure it does not contain malicious data.

9. Change Log

9.1. -00 to -01

There are no textual changes associated with this revision. This revision simply reflects a resubmission of the document so that it remains in active status.

10. References

10.1. Normative References

[CISCO-IOS]
CISCO, "Cisco IOS Reference Manual", 2014, <http://www.cisco.com/web/about/security/intelligence/ios-ref.html>.

[DEBIAN-POLICY-MANUAL]
Debian, "Debian Policy Manual", 2014,
<https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Version>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[RFC791] IETF, "Internet Protocol", 1981,
<https://tools.ietf.org/html/rfc791>.

[W3C-BOOLEAN]
W3C, "W3C Recommendation for Integer Data", 2004,
<http://www.w3.org/TR/xmlSchema-2/#boolean>.

[W3C-FLOAT]
W3C, "W3C Recommendation for Floating Point Data", 2004,
<http://www.w3.org/TR/xmlSchema-2/#float>.

[W3C-HEX-BIN]
W3C, "W3C Recommendation for Hex Binary Data", 2004,
<http://www.w3.org/TR/xmlschema-2/#hexBinary>.

[W3C-INT] W3C, "W3C Recommendation for Integer Data", 2004,
<http://www.w3.org/TR/xmlSchema-2/#integer>.

[W3C-STRING]
W3C, "W3C Recommendation for String Data", 2004,
<http://www.w3.org/TR/xmlSchema-2/#string>.

10.2. Informative References

[OVAL-WEBSITE]
The MITRE Corporation, "The Open Vulnerability and Assessment Language", 2015,
<http://ovalproject.github.io/>.

Authors' Addresses

Michael Cokus
The MITRE Corporation
903 Enterprise Parkway, Suite 200
Hampton, VA 23666
USA

Email: msc@mitre.org

Daniel Haynes
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
USA

Email: dhaynes@mitre.org

David Rothenberg
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
USA

Email: drothenberg@mitre.org

Juan Gonzalez
Department of Homeland Security
245 Murray Lane
Washington, DC 20548
USA

Email: juan.gonzalez@dhs.gov