Abstract

This document specifies Version 5.11.1 of the Common Model of the Open Vulnerability and Assessment Language (OVAL). It contains definitions of the constructs and enumerations that are used throughout the other core models in the OVAL Language both eliminating duplication and facilitating reuse.

Table of Contents

1. Introduction .................................................. 3
   1.1. Requirements Language .................................. 3
2. GeneratorType ................................................ 3
3. MessageType ................................................... 4
4. CheckEnumeration ............................................. 5
5. ClassEnumeration ............................................. 6
6. SimpleDatatypeEnumeration ................................... 7
7. ComplexDatatypeEnumeration ................................ 10
8. DatatypeEnumeration ......................................... 11
9. ExistenceEnumeration ....................................... 11
10. FamilyEnumeration .......................................... 12
11. MessageLevelEnumeration .................................. 14
12. OperationEnumeration ...................................... 14
13. OperatorEnumeration ...................................... 16
14. Definition, Test, Object, State, and Variable Identifiers .. 16
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 a 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 and is intended to serve as a starting point for its endpoint posture assessment data modeling needs.

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. GeneratorType

The GeneratorType provides a structure for recording information about how and when the OVAL Content was created, for what version of the OVAL Language it was created, and any additional information at the discretion of the content author.
The extension_point property is not considered a part of the OVAL Language proper, but rather, an extension point that allows organizations to expand the OVAL Language to better suit their needs.

3. MessageType

The MessageType construct is used to relay messages from tools at run-time. The decision of how to use these messages is left to the tool developer as an implementation detail based upon the context in which the message is used.
The CheckEnumeration enumeration defines the acceptable values that can be used to determine the final result of an evaluation based on how many of the individual results that make up an evaluation are true. This enumeration is used in different contexts throughout the OVAL Language. See the Check Enumeration Evaluation section of [I-D.draft-haynes-sacm-oval-processing-model], for more information on how this enumeration is used.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>The final result is 'true' only if all of the individual results under consideration are 'true'.</td>
</tr>
<tr>
<td>at least one</td>
<td>The final result is 'true' only if one or more of the individual results under consideration are 'true'.</td>
</tr>
<tr>
<td>none exist</td>
<td>DEPRECATED (5.3) In Version 5.3 of the OVAL Language, the checking of existence and state were separated into two distinct checks CheckEnumeration (state) and ExistenceEnumeration (existence). Since CheckEnumeration is now used to specify how many objects should satisfy a given state for a test to return true, and no longer used for specifying how many objects must exist for a test to return true, a value of 'none exist' is no longer needed. The final result is 'true' only if zero of the individual results under consideration are 'true'.</td>
</tr>
<tr>
<td>none satisfy</td>
<td>The final result is 'true' only if zero of the individual results under consideration are 'true'.</td>
</tr>
<tr>
<td>only one</td>
<td>The final result is 'true' only if one of the individual results under consideration is 'true'.</td>
</tr>
</tbody>
</table>

Table 3: CheckEnumeration Construct

5. ClassEnumeration

The ClassEnumeration defines the different classes of OVAL Definitions where each class specifies the overall intent of the OVAL Definition.
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compliance</td>
<td>This class describes OVAL Definitions that check to see if a system's state is compliant with a specific policy. An evaluation result of 'true', for this class of OVAL Definitions, indicates that a system is compliant with the stated policy.</td>
</tr>
<tr>
<td>inventory</td>
<td>This class describes OVAL Definitions that check to see if a piece of software is installed on a system. An evaluation result of 'true', for this class of OVAL Definitions, indicates that the specified software is installed on the system.</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>This class describes OVAL Definitions that do not belong to any of the other defined classes.</td>
</tr>
<tr>
<td>patch</td>
<td>This class describes OVAL Definitions that check to see if a patch should be installed on a system. An evaluation result of 'true', for this class of OVAL Definitions, indicates that the specified patch should be installed on the system.</td>
</tr>
<tr>
<td>vulnerability</td>
<td>This class describes OVAL Definitions that check to see if the system is in a vulnerable state. An evaluation result of 'true', for this class of OVAL Definitions, indicates that the system is in a vulnerable state.</td>
</tr>
</tbody>
</table>

Table 4: ClassEnumeration Construct

6. SimpleDatatypeEnumeration

The SimpleDatatypeEnumeration defines the legal simple datatypes that are used to describe the values in the OVAL Language. Simple datatypes are those that are based upon a string representation without additional structure. Each value in the SimpleDatatypeEnumeration has an allowed set of operations listed in the table below. These operations are based upon the full list of operations which are defined in the OperationEnumeration.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>Data of this type conforms to the W3C Recommendation for hex-encoded binary data [W3C-HEX-BIN]. Valid operations are: &quot;equals&quot; and &quot;not equal&quot;.</td>
</tr>
<tr>
<td>boolean</td>
<td>Data of this type conforms to the W3C Recommendation for boolean data [W3C-BOOLEAN]. Valid operations are: &quot;equals&quot; and &quot;not equal&quot;.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>evr_string</td>
<td>Data of this type conforms to the format EPOCH:VERSION-RELEASE and comparisons involving this type MUST follow the algorithm of librpm's rpmvercmp() function. Valid operations are: &quot;equals&quot;, &quot;not equal&quot;, &quot;greater than&quot;, &quot;greater than or equal&quot;, &quot;less than&quot;, and &quot;less than or equal&quot;.</td>
</tr>
<tr>
<td>debian_evr_string</td>
<td>Data of this type conforms to the format EPOCH:UPSTREAM_VERSION-DEBIAN_REVISION and comparisons involving this datatype should follow the algorithm outlined in Chapter 5 of the &quot;Debian Policy Manual&quot; [DEBIAN-POLICY-MANUAL]. An implementation of this is the cmpversions() function in dpkg's enquiry.c. Valid operations are: &quot;equals&quot;, &quot;not equal&quot;, &quot;greater than&quot;, &quot;greater than or equal&quot;, &quot;less than&quot;, and &quot;less than or equal&quot;.</td>
</tr>
<tr>
<td>fileset_revision</td>
<td>Data of this type conforms to the version string related to filesets in HP-UX. An example would be 'A.03.61.00'. Valid operations are: &quot;equals&quot;, &quot;not equal&quot;, &quot;greater than&quot;, &quot;greater than or equal&quot;, &quot;less than&quot;, and &quot;less than or equal&quot;.</td>
</tr>
<tr>
<td>float</td>
<td>Data of this type conforms to the W3C Recommendation for float data [W3C-FLOAT]. Valid operations are: &quot;equals&quot;, &quot;not equal&quot;, &quot;greater than&quot;, &quot;greater than or equal&quot;, &quot;less than&quot;, and &quot;less than or equal&quot;.</td>
</tr>
<tr>
<td>ios_version</td>
<td>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] Valid operations are: &quot;equals&quot;, &quot;not equal&quot;, &quot;greater than&quot;, &quot;greater than or equal&quot;, &quot;less than&quot;, and &quot;less than or equal&quot;.</td>
</tr>
<tr>
<td>int</td>
<td>Data of this type conforms to the W3C Recommendation for integer data [W3C-INT]. Valid operations are: &quot;equals&quot;, &quot;not equal&quot;, &quot;greater than&quot;, &quot;greater than or equal&quot;, &quot;less than&quot;, &quot;less than or equal&quot;, bitwise and&quot; and &quot;bitwise or&quot;.</td>
</tr>
<tr>
<td>ipv4_address</td>
<td>The ipv4_address 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. [RFC791] Valid operations are: &quot;equals&quot;, &quot;not equal&quot;, &quot;greater than&quot;, &quot;greater than or equal&quot;, &quot;less than&quot;, &quot;less than or equal&quot;, bitwise and&quot; and &quot;bitwise or&quot;.</td>
</tr>
</tbody>
</table>
operations are: "equals", "not equal", "greater than", "greater than or equal", "less than", "less than or equal", "subset of", and "superset of".

**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 RFC 4291 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. [RFC4291] Valid operations are: "equals", "not equal", "greater than", "greater than or equal", "less than", "less than or equal", "subset of", and "superset of".

**string**

Data of this type conforms to the W3C Recommendation for string data [W3C-STRING]. Valid operations are: "equals", "not equal", "case insensitive equals", "case insensitive not equal", and "pattern match".

**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-number character may be used as a delimiter and the delimiter may vary between component of a given version string. Valid operations are: "equals", "not equal", "greater than", "greater than or equal", "less than", and "less than or equal".

---

Table 5: SimpleDatatypeEnumeration Construct

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

7. ComplexDatatypeEnumeration

The ComplexDatatypeEnumeration defines the complex datatypes that are supported the OVAL Language. These datatypes describe the values with some structure beyond simple string like content. One simple example of a complex datatype is an address. The address might be composed of a street, city, state, and zip code. These field together comprise the complete address.

Each value in the ComplexDatatypeEnumeration has an allowed set of operations listed in the table below. These operations are based upon the full list of operations which are defined in the OperationEnumeration.
Data of this type represents a collection of named fields and values. Valid operations are: * equals

Table 6: ComplexDatatypeEnumeration Construct

8. DatatypeEnumeration

The DatatypeEnumeration defines the complete set of all valid datatypes. This set is created as the union of the SimpleDatatypeEnumeration and the ComplexDatatypeEnumeration. This type is provided for convenience when working with the OVAL Language.

9. ExistenceEnumeration

The ExistenceEnumeration defines the acceptable values that can be used to specify the expected number of components under consideration must exist.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_exist</td>
<td>The final existence result is 'true' only if all of the components under consideration exist.</td>
</tr>
<tr>
<td>any_exist</td>
<td>The final existence result is 'true' only if zero or more of the components under consideration exist.</td>
</tr>
<tr>
<td>at_least_one_exists</td>
<td>The final existence result is 'true' only if one or more of the components under consideration exist.</td>
</tr>
<tr>
<td>none_exist</td>
<td>The final existence result is 'true' only if zero of the components under consideration exist.</td>
</tr>
<tr>
<td>only_one_exists</td>
<td>The final existence result is 'true' only if one of the components under consideration exist.</td>
</tr>
</tbody>
</table>

Table 7: ExistenceEnumeration Construct

10. FamilyEnumeration

The FamilyEnumeration defines the high-level family that an operating system belongs to.
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>android</td>
<td>The android value describes the Android mobile operating system.</td>
</tr>
<tr>
<td>asa</td>
<td>The asa value describes the Cisco ASA security devices.</td>
</tr>
<tr>
<td>apple_ios</td>
<td>The apple_ios value describes the iOS mobile operating system.</td>
</tr>
<tr>
<td>catos</td>
<td>This value describes Cisco CatOS operating systems.</td>
</tr>
<tr>
<td>ios</td>
<td>This value describes Cisco IOS operating systems.</td>
</tr>
<tr>
<td>iosxe</td>
<td>This value describes Cisco IOS XE operating systems.</td>
</tr>
<tr>
<td>junos</td>
<td>This value describes Juniper JunOS operating systems.</td>
</tr>
<tr>
<td>macos</td>
<td>This value describes Apple Mac OS operating systems.</td>
</tr>
<tr>
<td>pixos</td>
<td>This value describes Cisco PIX operating systems.</td>
</tr>
<tr>
<td>undefined</td>
<td>This value is reserved for operating systems where the high-level family is not available in the current enumeration.</td>
</tr>
<tr>
<td>unix</td>
<td>This value describes UNIX operating systems.</td>
</tr>
<tr>
<td>vmware_infrastructure</td>
<td>This value describes the VMWare Infrastructure.</td>
</tr>
<tr>
<td>windows</td>
<td>This value describes Microsoft Windows operating systems.</td>
</tr>
</tbody>
</table>

Table 8: FamilyEnumeration Construct
11. MessageLevelEnumeration

The MessageLevelEnumeration defines the different levels that can be associated with a message.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>This level is reserved for messages that should only be displayed when the tool is run in verbose mode.</td>
</tr>
<tr>
<td>error</td>
<td>This level is reserved for messages where an error was encountered, but the tool could continue execution.</td>
</tr>
<tr>
<td>fatal</td>
<td>This level is reserved for messages where an error was encountered and the tool could not continue execution.</td>
</tr>
<tr>
<td>info</td>
<td>This level is reserved for messages that contain informational data.</td>
</tr>
<tr>
<td>warning</td>
<td>This level is reserved for messages that indicate that a problem may have occurred.</td>
</tr>
</tbody>
</table>

Table 9: MessageLevelEnumeration Construct

12. OperationEnumeration

The OperationEnumeration defines the acceptable operations in the OVAL Language. The precise meaning of an operation is dependent on the datatype of the values under consideration. See the OVAL Entity Datatype and Operation Evaluation section of [I-D.draft-haynes-sacm-oval-processing-model] for additional information.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals</td>
<td>This operation evaluates to 'true' if the actual value is equal to the stated value.</td>
</tr>
<tr>
<td>not equal</td>
<td>This operation evaluates to 'true' if the actual value is not equal to the stated value.</td>
</tr>
<tr>
<td>case insensitive equals</td>
<td>This operation evaluates to 'true' if the actual value is equal to the stated value when performing a case insensitive comparison.</td>
</tr>
<tr>
<td>case</td>
<td>This operation evaluates to 'true' if the actual value is not equal to the stated value.</td>
</tr>
<tr>
<td>greater than</td>
<td>This operation evaluates to 'true' if the actual value is greater than the stated value.</td>
</tr>
<tr>
<td>less than</td>
<td>This operation evaluates to 'true' if the actual value is less than the stated value.</td>
</tr>
<tr>
<td>greater than or equal</td>
<td>This operation evaluates to 'true' if the actual value is greater than or equal to the stated value.</td>
</tr>
<tr>
<td>less than or equal</td>
<td>This operation evaluates to 'true' if the actual value is less than or equal to the stated value.</td>
</tr>
<tr>
<td>bitwise and</td>
<td>This operation evaluates to 'true' if the result of the BITWISE AND operation between the binary representation of the stated value and the actual value is equal to the binary representation of the stated value. This operation is used to determine if a specific bit in a value is set.</td>
</tr>
</tbody>
</table>
13. OperatorEnumeration

The OperatorEnumeration defines the acceptable logical operators in the OVAL Language. See the Operator Enumeration Evaluation section of [I-D.draft-haynes-sacm-oval-processing-model] for additional information.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>This operator evaluates to 'true' only if every argument is 'true'.</td>
</tr>
<tr>
<td>ONE</td>
<td>This operator evaluates to 'true' only if one argument is 'true'.</td>
</tr>
<tr>
<td>OR</td>
<td>This operator evaluates to 'true' only if one or more arguments are 'true'.</td>
</tr>
<tr>
<td>XOR</td>
<td>This operator evaluates to 'true' only if an odd number of arguments are 'true'.</td>
</tr>
</tbody>
</table>

Table 11: OperatorEnumeration Construct

14. Definition, Test, Object, State, and Variable Identifiers

14.1. DefinitionIDPattern

The DefinitionIDPattern defines the URN format associated with OVAL Definition identifiers. All OVAL Definition identifiers MUST conform to the following regular expression:

```
oval:[A-Za-z0-9._-\.]+:def:[1-9][0-9]*
```

14.2. ObjectIDPattern

The ObjectIDPattern defines the URN format associated with OVAL Object identifiers. All OVAL Object identifiers MUST conform to the following regular expression:

```
oval:[A-Za-z0-9._-\.]+:obj:[1-9][0-9]*
```

14.3. StateIDPattern

The StateIDPattern defines the URN format associated with OVAL State identifiers. All OVAL State identifiers MUST conform to the
following regular expression:

```
14.4.  TestIDPattern

The TestIDPattern defines the URN format associated with OVAL Test
identifiers.  All OVAL Test identifiers MUST conform to the following
regular expression:

   oval:[A-Za-z0-9-\.\_\-\.]\+:ste:[1-9][0-9]*
```

```
14.5.  VariableIDPattern

The VariableIDPattern defines the URN format associated with OVAL
Variable identifiers.  All OVAL Variable identifiers MUST conform to
the following regular expression:

   oval:[A-Za-z0-9-\.\_\-\.]\+:var:[1-9][0-9]*
```

```
15.  ItemIDPattern

The ItemIDPattern defines the format associated with OVAL Item
identifiers.  All OVAL Item identifiers are unsigned integer values.
```

```
16.  EmptyStringType

The EmptyStringType defines a string value with a maximum length of
zero.
```

```
17.  NonEmptyStringType

The NonEmptyStringType defines a string value with a length greater
than zero.
```

```
18.  Any

The Any datatype represents an abstraction that serves as the basis
for other user defined datatypes.  This Any datatype does not
constrain its data in anyway.  This type is used to allow for
extension with the OVAL Language.
```

```
19.  Signature

The Signature type provides a structure for applying a digital
signature to OVAL Content.  Any binding or representation of the OVAL
Language MUST specify the format and structure of this type.  This
type is defined in an external namespace and when referenced in this
document will be prefix with the external namespace alias as follows,
```

```
<xsd:schema
```

```
<xsd:annotation>
```
description of the common types that are shared across the different schemas within Open Vulnerability and Assessment Language (OVAL). Each type is described in detail and should provide the information necessary to understand what each represents. This document is intended for developers and assumes some familiarity with XML. A high level description of the interaction between these type is not outlined here.}

<!---- ELEMENTS ---->
<xsd:element name="deprecated_info" type="oval:DeprecatedInfoType">
  <xsd:annotation>
    <xsd:documentation>The deprecated_info element is used in documenting deprecation information for items in the OVAL Language. It is declared globally as it can be found in any of the OVAL schemas and is used as part of the appinfo documentation and therefore it is not an element that can be declared locally and based off a global type.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="element_mapping" type="oval:ElementMapType">
  <xsd:annotation>
    <xsd:documentation>The element_mapping element is used in documenting which tests, objects, states, and system characteristic items are associated with each other. It provides a way to explicitly and programatically associate the test, object, state, and item definitions.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="notes" type="oval:NotesType">
  <xsd:annotation>
    <xsd:documentation>Element for containing notes; can be replaced using a substitution group.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<!---- TYPES ---->
<xsd:complexType name="ElementMapType">
  <xsd:annotation>
    <xsd:documentation>The ElementMapType is used to document the association between OVAL test, object, state, and item entities.</xsd:documentation>
  </xsd:annotation>
</xsd:complexType>
<xsd:element name="test" type="oval:ElementMapItemType" minOccurs="1">
  <xsd:annotation>
    <xsd:documentation>The local name of an OVAL test.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="object" type="oval:ElementMapItemType" minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>The local name of an OVAL object.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="state" type="oval:ElementMapItemType" minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>The local name of an OVAL state.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="item" type="oval:ElementMapItemType" minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>The local name of an OVAL item.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:complexType name="ElementMapItemType">
  <xsd:annotation>
    <xsd:documentation>Defines a reference to an OVAL entity using the schema namespace and element name.</xsd:documentation>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="xsd:NCName">
      <xsd:attribute name="target_namespace" type="xsd:anyURI" use="optional">
        <xsd:annotation>
          <xsd:documentation>The target_namespace attributes indicates what XML namespace the element belongs to. If not present, the namespace is that of the document in which the ElementMapItemType instance element appears.</xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="DeprecatedInfoType">
  <xsd:annotation>
    <xsd:documentation>The DeprecatedInfoType complex type defines a structure that will be used to flag schema-defined constructs as deprecated. It holds information related to the version of OVAL when the
The required version child element details the version of OVAL in which the construct became deprecated.

The required reason child element is used to provide an explanation as to why an item was deprecated and to direct a reader to possible alternative structures within OVAL.

The optional comment child element is used to supply additional information regarding the element's deprecated status.

The GeneratorType complex type defines an element that is used to hold information about when a particular OVAL document was compiled, what version of the schema was used, what tool compiled the document, and what version of that tool was used. Additional generator information is also allowed although it is not part of the official OVAL Schema. Individual organizations can place generator information that they feel are important and these will be skipped during the validation. All OVAL really cares about is that the stated generator information is there.
<xsd:element name="product_version"
    type="xsd:string" minOccurs="0"
    maxOccurs="1">
    <xsd:annotation>
      <xsd:documentation>The optional product_version specifies the version of the application used to generate the file.</xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="schema_version"
    maxOccurs="unbounded"
    type="oval:SchemaVersionType">
    <xsd:annotation>
      <xsd:documentation>The required schema_version specifies the version of the OVAL Schema that the document has been written in and that should be used for validation. The versions for both the Core and any platform extensions used should be declared in separate schema_version elements.</xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="timestamp"
    type="xsd:dateTime">
    <xsd:annotation>
      <xsd:documentation>The required timestamp specifies when the particular OVAL document was compiled. The format for the timestamp is yyyy-mm-ddThh:mm:ss. Note that the timestamp element does not specify when a definition (or set of definitions) was created or modified but rather when the actual XML document that contains the definition was created. For example, the document might have pulled a bunch of existing OVAL Definitions together, each of the definitions having been created at some point in the past. The timestamp in this case would be when the combined document was created.</xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="asset_identification"
    minOccurs="0" maxOccurs="unbounded"
    processContents="lax">
    <xsd:annotation>
      <xsd:documentation>The Asset Identification specification (http://scap.nist.gov/specifications/ai/) provides a standardized way of reporting asset information across different organizations.</xsd:documentation>
    </xsd:annotation>
</xsd:element>
document, such as persons or organizations.</xsd:documentation>
<xsd:documentation>To support greater interoperability, an ai:assets element describing assets used to produce an OVAL document may appear at this point in an OVAL document.</xsd:documentation>
</xsd:annotation>
</xsd:any>
</xsd:sequence>
</xsd:complexType>

To support greater interoperability, an ai:assets element describing assets used to produce an OVAL document may appear at this point in an OVAL document. The platform name of the OVAL document is returned along with the schema version.
attribute is available to indicate the URI of the target namespace for any platform extension being included. This platform attribute is to be omitted when specifying the core schema version.</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
</xsd:extension>
</xsd:complexType>
</xsd:complexType>

<xsd:complexType name="MessageType">

<xsd:annotation>
<xsd:documentation>The MessageType complex type defines the structure for which messages are relayed from the data collection engine. Each message is a text string that has an associated level attribute identifying the type of message being sent. These messages could be error messages, warning messages, debug messages, etc. How the messages are used by tools and whether or not they are displayed to the user is up to the specific implementation. Please refer to the description of the MessageLevelEnumeration for more information about each type of message.</xsd:documentation>
</xsd:annotation>
<xsd:extension base="xsd:string">
<xsd:attribute name="level" type="oval:MessageLevelEnumeration" use="optional" default="info"/>
</xsd:extension>
</xsd:complexType>

<xsd:complexType name="NotesType">

<xsd:annotation>
<xsd:documentation>The NotesType complex type is a container for one or more note child elements. Each note contains some information about the definition or tests that it references. A note may record an unresolved question about the definition or test or present the reason as to why a particular approach was taken.</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="note" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

<!-- ====================================================== -->
<!-- ================  ENUMERATIONS  ====================== -->
<!-- ====================================================== -->

<xsd:simpleType name="CheckEnumeration">

<xsd:annotation>
<xsd:documentation>The CheckEnumeration simple type defines acceptable check values, which are used to determine the final result of something based on the results of individual components, when used to define the relationship between
objects and states, each check value defines how many of the matching objects (items except those with a status of does not exist) must satisfy the given state for the test to return true. When used to define the relationship between instances of a given entity, the different check values defines how many instances must be true for the entity to return true. When used to define the relationship between entities and multiple variable values, each check value defines how many variable values must be true for the entity to return true.

Below are some tables that outline how each check attribute effects evaluation. The far left column identifies the check attribute in question. The middle column specifies the different combinations of individual results that the check attribute may bind together. (T=true, F=false, E=error, U=unknown, NE=not evaluated, NA=not applicable) For example, a 1+ under T means that one or more individual results are true, while a 0 under U means that zero individual results are unknown. The last column specifies what the final result would be according to each combination of individual results. Note that if the individual test is negated, then a true result is false and a false result is true, all other results stay as is.

<table>
<thead>
<tr>
<th>check attr is</th>
<th>num of individual results</th>
<th>final result is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>ALL</td>
<td>0+</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>


Internet-Draft OVAL Common Model September 2016
<table>
<thead>
<tr>
<th>num of individual results</th>
<th>final result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>0</td>
<td>1+</td>
</tr>
<tr>
<td>1+</td>
<td>0+</td>
</tr>
<tr>
<td>0</td>
<td>0+</td>
</tr>
<tr>
<td>0</td>
<td>0+</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>


Internet-Draft  OVAL Common Model  September 2016

given if all the individual results under consideration are true.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="at least one">  
<xsd:annotation>
<xsd:documentation>A value of 'at least one' means that a final result of true is given if at least one of the individual results under consideration is true.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="none exist">  
<xsd:annotation>
<xsd:documentation>A value of 'none exists' means that a test evaluates to true if no matching object exists that satisfy the data requirements.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:appinfo>
<oval:deprecated_info>
<oval:version>5.3</oval:version>
<oval:reason>Replaced by the 'none satisfy' value. In version 5.3 of the OVAL Language, the checking of existence and state were separated into two distinct checks CheckEnumeration (state) and ExistenceEnumeration (existence). Since CheckEnumeration is now used to specify how many objects should satisfy a given state for a test to return true, and no longer used for specifying how many objects must exist for a test to return true, a value of 'none exist' is no longer needed. See the 'none satisfy' value.</oval:reason>
<oval:comment>This value has been deprecated and will be removed in version 6.0 of the language.</oval:comment>
</oval:deprecated_info>
<sch:pattern id="oval_none_exist_value_dep">  
<sch:rule
<xsd:simpleType name="ClassEnumeration">
  <xsd:documentation>The ClassEnumeration simple type defines the different classes of definitions. Each class defines a certain intent regarding how an OVAL Definition is written and what that definition is describing. The specified class gives a hint about the definition so a user can know what the definition writer is trying to say. Note that the class does not make a statement about whether a true result is good or bad as this depends on the use of an OVAL Definition. These classes are also used to group definitions by the type of system state they are describing. For example, this allows users to find all the vulnerability (or patch, or inventory, etc) definitions.</xsd:documentation>
  <xsd:annotation>
    <xsd:documentation>A compliance definition describes the state of a machine as it complies with a specific policy. A definition of this class will evaluate to true when the system is found to be compliant with the stated policy. Another way of thinking about this is that a compliance
definition is stating "the system is compliant if ...".
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="inventory">
  <xsd:annotation>
    <xsd:documentation>An inventory definition describes whether a specific piece of software is installed on the system. A definition of this class will evaluate to true when the specified software is found on the system. Another way of thinking about this is that an inventory definition is stating "the software is installed if ...".
    </xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="miscellaneous">
  <xsd:annotation>
    <xsd:documentation>The 'miscellaneous' class is used to identify definitions that do not fall into any of the other defined classes.
    </xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="patch">
  <xsd:annotation>
    <xsd:documentation>A patch definition details the machine state of whether a patch executable should be installed. A definition of this class will evaluate to true when the specified patch is missing from the system. Another way of thinking about this is that a patch definition is stating "the patch should be installed if ...". Note that word SHOULD is intended to mean more than just CAN the patch executable be installed. In other words, if a more recent patch is already installed then the specified patch might not need to be installed.
    </xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="vulnerability">
  <xsd:annotation>
    <xsd:documentation>A vulnerability definition describes the conditions under which a machine is vulnerable. A definition of this class will evaluate to true when the system is found to be vulnerable with the stated issue. Another way of thinking about this is that a vulnerability definition is stating "the system is vulnerable if ...".
    </xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SimpleDatatypeEnumeration">
  <xsd:annotation>
    <xsd:documentation>The SimpleDatatypeEnumeration simple type defines the legal datatypes that are used to describe the values of individual entities that can be represented in a XML string field. The value may have structure and a pattern, but it is represented as string content.
    </xsd:documentation>
  </xsd:annotation>
</xsd:annotation>
This datatype conforms to the W3C Recommendation for binary data meaning that each binary octet is encoded as a character tuple, consisting of two hexadecimal digits \([0-9a-fA-F]\) representing the octet code. Expected operations within OVAL for binary values are 'equals' and 'not equal'.

The boolean datatype represents standard boolean data, either true or false. This datatype conforms to the W3C Recommendation for boolean data meaning that the following literals are legal values: \{true, false, 1, 0\}. Expected operations within OVAL for boolean values are 'equals' and 'not equal'.

The evr_string datatype represents the epoch, version, and release fields as a single version string. It has the form "EPOCH:VERSION-RELEASE". Comparisons involving this datatype should follow the algorithm of librpm's rpmvercmp() function. Expected operations within OVAL for evr_string values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.

The debian_evr_string datatype represents the epoch, upstream_version, and debian_revision fields, for a Debian package, as a single version string. It has the form "EPOCH:UPSTREAM_VERSION-DEBIAN_REVISION". Comparisons involving this datatype should follow the algorithm outlined in Chapter 5 of the "Debian Policy Manual" (https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-version). An implementation of this is the cmpversions() function in dpkg's inquiry.c. Expected operations within...
OVAL for debian_evr_string values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.

OVAL for fileset_revision values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.

OVAL for float values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.

OVAL for ios_version values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.

OVAL for int values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.
Expected operations within OVAL for int values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', 'less than or equal', 'bitwise and', and 'bitwise or'.

The ipv4_address 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.

The expected operations within OVAL for ipv4_address values are 'equals', 'not equal', 'greater than', 'greater than or equal', 'less than', 'less than or equal', 'subset of', and 'superset of'. 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.

The following defines how to perform each operation for the ipv4_address datatype. Let A_addr mean the first element of ordered pair A and A_prefix mean the second element.
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 and they satisfy either the criteria for operators "equal" or "greater than", 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 don't satisfy the criteria for operator "equal". 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 A_addr < B_addr. 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 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.
of. 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.

The following defines how to perform each operation for the ipv6_address datatype. Let P_addr mean the first element of ordered pair P and P_prefix mean the second element.

<enumeration value="equals"> A equals B if and only if A_addr == B_addr and A_prefix == B_prefix.</enumeration>
<enumeration value="not equal"> A is not equal to B if and only if they don't satisfy the criteria for operator "equals".</enumeration>
<enumeration value="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.</enumeration>
<enumeration value="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.</enumeration>
<enumeration value="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.</enumeration>
<enumeration value="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.</enumeration>
<enumeration value="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.</enumeration>
<enumeration value="superset of"> A is a superset of B if and only if B is a subset of A.</enumeration>
The string datatype describes standard string data. This datatype conforms to the W3C Recommendation for string data. Expected operations within OVAL for string values are 'equals', 'not equal', 'case insensitive equals', 'case insensitive not equal', and 'pattern match'.

The version datatype represents a value that is a hierarchical list of non-negative integers separated by a single character delimiter. Note that any non-number character can be used as a delimiter and that different characters can be used within the same version string. So '1.2.3' is the same as '1.2.3' or '1-2-3' where '#' is any non-negative integer. Expected operations within OVAL for version values are 'equals', '.not equal', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.

For example '1.2.3' or '1-2-3' where the numbers to the left are more significant than the numbers to the right. When performing an 'equals' operation on a version datatype, you should first check the left most number for equality. If that fails, then the values are not equal. If it succeeds, then check the second left most number for equality. Continue checking the numbers from left to right until the last number has been checked. If, after testing all the previous numbers, the last number is equal then the two versions are equal. When performing other operations, such as 'less than', 'less than or equal', 'greater than, or greater than or equal', similar logic as above is used. Start with the left most number and move from left to right. For each number, check if it is less than the number you are testing against. If it is, then the version in question is less than the version you are testing against. If the number is equal, then move to check the next number to the right. For example, to test if 5.7.23 is less than or equal to 5.8.0 you first compare 5 to 5. They are equal so you move on to compare 7 to 8. 7 is less than 8 so the entire test succeeds and 5.7.23 is 'less than or equal' to 5.8.0. The difference between the 'less than' and 'less than or equal' operations is how the last number is handled. If the last number is reached, the check should use the given operation (either 'less than' and 'less than or equal') to test the number. For example, to test if 4.23.6 is greater than 4.23.6
you first compare 4 to 4. They are equal so you move on to compare 23 to 23. They are equal so you move on to compare 6 to 6. This is the last number in the version and since 6 is not greater than 6, the entire test fails and 4.23.6 is not greater than 4.23.6.

Version strings with a different number of components shall be padded with zeros to make them the same size. For example, if the version strings '1.2.3' and '6.7.8.9' are being compared, then the short one should be padded to become '1.2.3.0'.

ComplexDatatypeEnumeration simple type defines the complex legal datatypes that are supported in OVAL. These datatypes describe the values of individual entities where the entity has some complex structure beyond simple string like content.

The record datatype describes an entity with structured set of named fields and values as its content. The only allowed operation within OVAL for record values is 'equals'. Note that the record datatype is not currently allowed when using variables.

The DatatypeEnumeration simple type defines the legal datatypes that are used to describe the values of individual entities. A value should be interpreted according to the specified type. This is most important during comparisons. For example, is '21' less than '123'? will evaluate to true if the datatypes are 'int', but will evaluate to 'false' if the datatypes are 'string'. Another example is applying the 'equal' operation to '1.0.0.0' and '1.0'. with datatype 'string' they are not equal, with datatype 'version' they are.
<xsd:documentation>The ExistenceEnumeration simple type defines acceptable existence values, which are used to determine a result based on the existence of individual components. The main use for this is for a test regarding the existence of objects on the system.</xsd:documentation>

<evaluation_documentation>Below are some tables that outline how each ExistenceEnumeration value effects evaluation of a given test. Note that this is related to the existence of an object(s) and not the object(s) compliance with a state. The left column identifies the ExistenceEnumeration value in question. The middle column specifies the different combinations of individual item status values that have been found in the system characteristics file related to the given object. (EX=exists, DE=does not exist, ER=error, NC=not collected) For example, a 1+ under EX means that one or more individual item status attributes are set to exists, while a 0 under NC means that zero individual item status attributes are set to not collected. The last column specifies what the result of the existence piece would be according to each combination of individual item status values.</evaluation_documentation>

<evaluation_chart xml:space="preserve">

<table>
<thead>
<tr>
<th>attr value</th>
<th>item status value count</th>
<th>existence piece is</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_exist</td>
<td>1+ 0 0 0</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>0+ 1+ 0+ 0+</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>0+ 0 1+ 0+</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>0+ 0 0 1+</td>
<td>Unknown</td>
</tr>
<tr>
<td></td>
<td>-- -- -- --</td>
<td>Not Evaluated</td>
</tr>
<tr>
<td></td>
<td>-- -- -- --</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

<evaluation_chart xml:space="preserve">

<table>
<thead>
<tr>
<th>attr value</th>
<th>item status value count</th>
<th>existence piece is</th>
</tr>
</thead>
<tbody>
<tr>
<td>any_exist</td>
<td>0+ 0+ 0 0+</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>1+ 0+ 1+ 0+</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>-- -- -- --</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>0 0+ 1+ 0+</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>-- -- -- --</td>
<td>Unknown</td>
</tr>
<tr>
<td></td>
<td>-- -- -- --</td>
<td>Not Evaluated</td>
</tr>
<tr>
<td></td>
<td>-- -- -- --</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

<evaluation_chart xml:space="preserve">

<table>
<thead>
<tr>
<th>attr value</th>
<th>item status value count</th>
<th>existence piece is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EX</td>
<td>DE</td>
</tr>
</tbody>
</table>
### at_least_one_exists

<table>
<thead>
<tr>
<th></th>
<th>0+</th>
<th>0+</th>
<th>1+</th>
<th>0+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1+</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

**True**

**False**

**Error**

**Unknown**

**Not Evaluated**

**Not Applicable**

### none_exists

<table>
<thead>
<tr>
<th></th>
<th>0+</th>
<th>0+</th>
<th>1+</th>
<th>0+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0+</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

**Error**

**Unknown**

**Not Evaluated**

**Not Applicable**

---


Internet-Draft OVAL Common Model September 2016
<xsd:enumeration value="only_one_exists">
  <xsd:annotation>
    <xsd:documentation>A value of
    'only_one_exists' means that only one
    object defined by the description
    exists on the
    system.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="FamilyEnumeration">
  <xsd:annotation>
    <xsd:documentation>The FamilyEnumeration
    simple type is a listing of families that
    OVAL supports at this time. Since new
    family values can only be added with new
    version of the schema, the value of
    'undefined' is to be used when the desired
    family is not available. Note that use of
    the undefined family value does not target
    all families, rather it means that some
    family other than one of the defined
    values is targeted.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="android">
      <xsd:annotation>
        <xsd:documentation>The android value
        describes the Android mobile operating
        system.</xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="asa">
      <xsd:annotation>
        <xsd:documentation>The asa value
        describes the Cisco ASA security
        devices.</xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="apple_ios">
      <xsd:annotation>
        <xsd:documentation>The apple_ios value
        describes the iOS mobile operating
        system.</xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="catos">
      <xsd:annotation>
        <xsd:documentation>The catos value
        describes the Cisco CatOS operating
        system.</xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="ios">
      <xsd:annotation>
        <xsd:documentation>The ios value
        describes the Cisco IOS operating
        system.</xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="iosxe">
      <xsd:annotation>
        <xsd:documentation>The iosxe value
        describes the Cisco IOSXe operating
        system.</xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
  </xsd:restriction>
</xsd:simpleType>
describes the Cisco IOS XE operating system.
</xsd:enumeration>
</xsd:annotation>
<xsd:enumeration value="junos">
  <xsd:annotation>
    <xsd:documentation>The junos value describes the Juniper JunOS operating system.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="macos">
  <xsd:annotation>
    <xsd:documentation>The macos value describes the Mac operating system.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="pixos">
  <xsd:annotation>
    <xsd:documentation>The pixos value describes the Cisco PIX operating system.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="undefined">
  <xsd:annotation>
    <xsd:documentation>The undefined value is to be used when the desired family is not available.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="unix">
  <xsd:annotation>
    <xsd:documentation>The unix value describes the UNIX operating system.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="vmware_infrastructure">
  <xsd:annotation>
    <xsd:documentation>The vmware_infrastructure value describes VMWare Infrastructure.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="windows">
  <xsd:annotation>
    <xsd:documentation>The windows value describes the Microsoft Windows operating system.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="MessageLevelEnumeration">
  <xsd:annotation>
    <xsd:documentation>The MessageLevelEnumeration simple type defines the different levels associated with a message. There is no specific criteria about which messages get assigned which level. This is completely arbitrary and up to the content producer to decide what is an error message and what is a debug message.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="debug">
      <xsd:annotation>
        <xsd:documentation>Debug messages should
only be displayed by a tool when run in some sort of verbose mode.

<xsd:documentation>
<xsd:enumeration value="error">
<xsd:annotation>
  <xsd:documentation>Error messages should be recorded when there was an error that did not allow the collection of specific data.</xsd:documentation>
</xsd:enumeration>
<xsd:enumeration value="fatal">
<xsd:annotation>
  <xsd:documentation>A fatal message should be recorded when an error causes the failure of more than just a single piece of data.</xsd:documentation>
</xsd:enumeration>
<xsd:enumeration value="info">
<xsd:annotation>
  <xsd:documentation>Info messages are used to pass useful information about the data collection to a user.</xsd:documentation>
</xsd:enumeration>
<xsd:enumeration value="warning">
<xsd:annotation>
  <xsd:documentation>A warning message reports something that might not correct but information was still collected.</xsd:documentation>
</xsd:enumeration>
</xsd:enumeration>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="OperationEnumeration">
<xsd:annotation>
<xsd:documentation>The OperationEnumeration simple type defines acceptable operations. Each operation defines how to compare entities against their actual values.</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="equals">
<xsd:annotation>
<xsd:documentation>The 'equals' operation returns true if the actual value on the system is equal to the stated entity. When the specified datatype is a string, this results in a case-sensitive comparison.</xsd:documentation>
</xsd:enumeration>
<xsd:enumeration value="not equal">
<xsd:annotation>
<xsd:documentation>The 'not equal' operation returns true if the actual value on the system is not equal to
the stated entity. When the specified
datatype is a string, this results in
a case-sensitive
comparison.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration
value="case insensitive equals">
<xsd:annotation>
<xsd:documentation>The 'case insensitive
equals' operation is meant for string
data and returns true if the actual
value on the system is equal (using a
case insensitive comparison) to the
stated entity.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration
value="case insensitive not equal">
<xsd:annotation>
<xsd:documentation>The 'case insensitive
not equal' operation is meant for
string data and returns true if the
actual value on the system is not
equal (using a case insensitive
comparison) to the stated
entity.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="greater than">
<xsd:annotation>
<xsd:documentation>The 'greater than'
operation returns true if the actual
value on the system is greater than
the stated entity.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="less than">
<xsd:annotation>
<xsd:documentation>The 'less than'
operation returns true if the actual
value on the system is less than the
stated entity.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="greater than or equal">
<xsd:annotation>
<xsd:documentation>The 'greater than or
equal' operation returns true if the
actual value on the system is greater
than or equal to the stated
entity.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="less than or equal">
<xsd:annotation>
<xsd:documentation>The 'less than or
equal' operation returns true if the
actual value on the system is less
than or equal to the stated
entity.</xsd:documentation>
</xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="bitwise and">
<xsd:annotation>
<xsd:documentation>The 'bitwise and'
operation is used to determine if a
specific bit is set. It returns true
if performing a BITWISE AND with the
binary representation of the stated
entity against the binary
representation of the actual value on
the system results in a binary value that is equal to the binary representation of the stated entity. For example, assuming a datatype of 'int', if the actual integer value of the setting on your machine is 6 (same as 0110 in binary), then performing a 'bitwise and' with the stated integer 4 (0100) returns 4 (0100). Since the result is the same as the state mask, then the test returns true. If the actual value on your machine is 1 (0001), then the 'bitwise and' with the stated integer 4 (0100) returns 0 (0000). Since the result is not the same as the stated mask, then the test fails.

The 'bitwise or' operation is used to determine if a specific bit is not set. It returns true if performing a BITWISE OR with the binary representation of the stated entity against the binary representation of the actual value on the system results in a binary value that is equal to the binary representation of the stated entity. For example, assuming a datatype of 'int', if the actual integer value of the setting on your machine is 6 (same as 0110 in binary), then performing a 'bitwise or' with the stated integer 14 (1110) returns 14 (1110). Since the result is the same as the state mask, then the test returns true. If the actual value on your machine is 1 (0001), then the 'bitwise or' with the stated integer 14 (1110) returns 15 (1111). Since the result is not the same as the stated mask, then the test fails.

The 'pattern match' operation allows an item to be tested against a regular expression. When used by an entity in an OVAL Object, the regular expression represents the unique set of matching items on the system. OVAL supports a common subset of the regular expression character classes, operations, expressions and other lexical tokens defined within Perl 5's regular expression specification. For more information on the supported regular expression syntax in OVAL see: http://oval.mitre.org/language/about/re_support_5.6.html

The 'bitwise and' operation is used to determine if a specific bit is set. It returns true if performing a BITWISE AND with the binary representation of the stated entity against the binary representation of the actual value on the system results in a binary value that is equal to the binary representation of the stated entity. For example, assuming a datatype of 'int', if the actual integer value of the setting on your machine is 6 (same as 0110 in binary), then performing a 'bitwise and' with the stated integer 4 (0100) returns 4 (0100). Since the result is the same as the state mask, then the test returns true. If the actual value on your machine is 1 (0001), then the 'bitwise and' with the stated integer 14 (1110) returns 0 (0000). Since the result is not the same as the stated mask, then the test fails.

The 'bitwise or' operation is used to determine if a specific bit is set. It returns true if performing a BITWISE OR with the binary representation of the stated entity against the binary representation of the actual value on the system results in a binary value that is equal to the binary representation of the stated entity. For example, assuming a datatype of 'int', if the actual integer value of the setting on your machine is 6 (same as 0110 in binary), then performing a 'bitwise or' with the stated integer 14 (1110) returns 14 (1110). Since the result is the same as the state mask, then the test returns true. If the actual value on your machine is 1 (0001), then the 'bitwise or' with the stated integer 14 (1110) returns 15 (1111). Since the result is not the same as the stated mask, then the test fails.

The 'pattern match' operation allows an item to be tested against a regular expression. When used by an entity in an OVAL Object, the regular expression represents the unique set of matching items on the system. OVAL supports a common subset of the regular expression character classes, operations, expressions and other lexical tokens defined within Perl 5's regular expression specification. For more information on the supported regular expression syntax in OVAL see: http://oval.mitre.org/language/about/re_support_5.6.html
The 'subset of' operation returns true if the actual set on the system is a subset of the set defined by the stated entity.

The 'superset of' operation returns true if the actual set on the system is a superset of the set defined by the stated entity.

### OperatorEnumeration

The OperatorEnumeration simple type defines acceptable operators. Each operator defines how to evaluate multiple arguments.

Below are some tables that outline how each operator affects evaluation. The far left column identifies the operator in question. The middle column specifies the different combinations of individual results that the operator may bind together. (T=true, F=false, E=error, U=unknown, NE=not evaluated, NA=not applicable) For example, a 1+ under T means that one or more individual results are true, while a 0 under U means that zero individual results are unknown. The last column specifies what the final result would be according to each combination of individual results. Note that if the individual test is negated, then a true result is false and a false result is true, all other results stay as is.

#### Evaluation Chart: AND

<table>
<thead>
<tr>
<th>num of individual results</th>
<th>T</th>
<th>F</th>
<th>E</th>
<th>U</th>
<th>NE</th>
<th>NA</th>
<th>final result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>True</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>False</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>Error</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>Unknown</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>Not Evaluated</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

#### Evaluation Chart: ONE

<table>
<thead>
<tr>
<th>num of individual results</th>
<th>T</th>
<th>F</th>
<th>E</th>
<th>U</th>
<th>NE</th>
<th>NA</th>
<th>final result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0+</td>
<td>0+</td>
<td>True</td>
</tr>
<tr>
<td>2+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td><strong>False</strong></td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td><strong>False</strong></td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>Error</td>
</tr>
<tr>
<td>0.1</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>Unknown</td>
</tr>
<tr>
<td>0.1</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>Not Evaluated</td>
</tr>
<tr>
<td>operator is</td>
<td>final result is</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>E</td>
<td>U</td>
<td>NE</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0+</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0+</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>0+</td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0+</td>
<td>0</td>
<td>1+</td>
<td>0+</td>
<td>0+</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0+</td>
<td>0</td>
<td>0</td>
<td>1+</td>
<td>0+</td>
<td>Not Evaluated</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1+</td>
<td>Not Applicable</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operator is</th>
<th>final result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>odd</td>
<td>0+</td>
</tr>
<tr>
<td>even</td>
<td>0+</td>
</tr>
</tbody>
</table>

XOR

<table>
<thead>
<tr>
<th>operator is</th>
<th>final result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
</tr>
<tr>
<td>0+</td>
<td>0+</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>


Internet-Draft OVAL Common Model September 2016

xsd:restriction base="xsd:string">
<xsd:enumeration value="AND">
  <xsd:annotation>
    <xsd:documentation>The AND operator produces a true result if every argument is true. If one or more arguments are false, the result of the AND is false. If one or more of the arguments are unknown, and if none of the arguments are false, then the AND operator produces a result of unknown.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="ONE">
  <xsd:annotation>
    <xsd:documentation>The ONE operator produces a true result if one and only one argument is true. If there are more than argument is true (or if there are no true arguments), the result of the ONE is false. If one or more of the arguments are unknown, then the ONE operator produces a result of unknown.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="OR">
  <xsd:annotation>
    <xsd:documentation>The OR operator produces a true result if one or more arguments is true. If every argument is false, the result of the OR is false. If one or more of the arguments are unknown and if none of arguments are true, then the OR operator produces a result of unknown.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="XOR">
  <xsd:annotation>
    <xsd:documentation>XOR is defined to be true if an odd number of its arguments are true, and false otherwise. If any of the arguments are unknown, then the XOR operator produces a result of unknown.</xsd:documentation>
  </xsd:annotation>
</xsd:enumeration>

<!-- === ID PATTERNS === -->
<xsd:simpleType name="DefinitionIDPattern">
  <xsd:annotation>
    <xsd:documentation>Define the format for acceptable OVAL Definition ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'def', and ending with an integer.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="oval:[A-Za-z0-9_\-.]+:def:[1-9][0-9]*"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ObjectIDPattern">
  <xsd:annotation>
    <xsd:documentation>Define the format for acceptable OVAL Object ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'obj', and ending with an integer.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="oval:[A-Za-z0-9_\-.]+:obj:[1-9][0-9]*"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="StateIDPattern">
  <xsd:annotation>
    <xsd:documentation>Define the format for acceptable OVAL State ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'ste', and ending with an integer.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="oval:[A-Za-z0-9_\-.]+:ste:[1-9][0-9]*"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TestIdPattern">
  <xsd:annotation>
    <xsd:documentation>Define the format for acceptable OVAL Test ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'tes', and ending with an integer.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="oval:[A-Za-z0-9_\-.]+:tes:[1-9][0-9]*"/>
  </xsd:restriction>
</xsd:simpleType>
by the three letter code 'tst', and ending
with an integer.</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:string">
  <xsd:pattern
    value="oval:[A-Za-z0-9_\-\.]+:tst:[1-9][0-9]*"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="VariableIDPattern">
  <xsd:annotation>
    <xsd:documentation>Define the format for
    acceptable OVAL Variable ids. An urn
    format is used with the id starting with
    the word oval followed by a unique string,
    followed by the three letter code 'var',
    and ending with an
    integer.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern
      value="oval:[A-Za-z0-9_\-\.]+:var:[1-9][0-9]*"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ItemIDPattern">
  <xsd:annotation>
    <xsd:documentation>Define the format for
    acceptable OVAL Item ids. The format is an
    integer. An item id is used to identify
    the different items found in an OVAL
    System Characteristics
    file.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>
<xsd:simpleType name="SchemaVersionPattern">
  <xsd:annotation>
    <xsd:documentation>Define the format for
    acceptable OVAL Language version
    strings.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern
  </xsd:restriction>
</xsd:simpleType>
<!---------------------------------------------------------------------->
<!- OTHER TYPES ------------------------------------------------------->
<!---------------------------------------------------------------------->
<xsd:simpleType name="EmptyStringType">
  <xsd:annotation>
    <xsd:documentation>The EmptyStringType
    simple type is a restriction of the
    built-in string simpleType. The only
    allowed string is the empty string with a
    length of zero. This type is used by
    certain elements to allow empty content
    when non-string data is accepted. See the
    EntityIntType in the OVAL Definition
    Schema for an example of its
    use.</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="0"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="NonEmptyStringType">
  <xsd:annotation>
    <xsd:documentation>The NonEmptyStringType
simple type is a restriction of the built-in string simpleType. Empty strings are not allowed. This type is used by comment attributes where an empty value is not allowed.</xsd:documentation>
</xsd:annotation>
</xsd:restriction>
</xsd:simpleType>
<!-- ====================================================== -->
<!-- ====================================================== -->
<!-- ====================================================== -->
</xsd:schema>

21. Intellectual Property Considerations

Copyright (C) 2010 United States Government. All Rights Reserved.

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.

22. 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.

23. IANA Considerations

This memo includes no request to IANA.

24. 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.
25. Change Log

25.1. -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.

26. References

26.1. Normative References


[RFC791]IETF, "Internet Protocol", 1981,

26.2. Informative References


Appendix A. Terms and Acronyms
### Table 12: Terms and Acronyms Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVAL</td>
<td>An action that can further specify the set of OVAL Behavior Items that matches an OVAL Object.</td>
</tr>
<tr>
<td>OVAL Test</td>
<td>An OVAL Test is the standardized representation of an assertion about the state of a system.</td>
</tr>
<tr>
<td>OVAL Object</td>
<td>An OVAL Object is a collection of OVAL Object Entities that can uniquely identify a single OVAL Item on the system.</td>
</tr>
<tr>
<td>OVAL Item</td>
<td>An OVAL Item is a single piece of collected system state information.</td>
</tr>
<tr>
<td>OVAL Component</td>
<td>An OVAL Construct that is specified in the oval-def:ComponentGroup.</td>
</tr>
<tr>
<td>OVAL Function</td>
<td>An OVAL Function is a capability used in OVAL Variables to manipulate a variable's value.</td>
</tr>
<tr>
<td>OVAL Variable</td>
<td>An OVAL Variable represents a collection of values that allow for dynamic substitutions and reuse of system state information.</td>
</tr>
<tr>
<td>OVAL Object Entity</td>
<td>An OVAL Object Entity is a standardized representation for specifying a single piece of system state information.</td>
</tr>
<tr>
<td>OVAL State Entity</td>
<td>An OVAL State Entity is a standardized representation for checking a single piece of system state information.</td>
</tr>
<tr>
<td>OVAL Item Entity</td>
<td>An OVAL Item Entity is a standardized representation for a single piece of system state information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCE</td>
<td>Common Configuration Enumeration</td>
</tr>
<tr>
<td>CPE</td>
<td>Common Platform Enumeration</td>
</tr>
</tbody>
</table>
The OVAL Language supports a common subset of the regular expression character classes, operations, expressions, and other lexical tokens defined within Perl 5's regular expression specification. This common subset was identified through a survey of several regular expression libraries in an effort to ensure that the regular expression elements supported by OVAL will be compatible with a wide variety of regular expression libraries. A listing of the surveyed regular expression libraries is provided later in this document.

### B.1. Supported Regular Expression Syntax

Perl regular expression modifiers (m, i, s, x) are not supported. These modifiers should be considered to always be 'OFF, unless specifically permitted by documentation on an OVAL Language construct.

Character matching assumes a Unicode character set. Note that no syntax is supplied for specifying code points in hex; actual Unicode characters must be used instead.

The following regular expression elements are specifically identified as supported in the OVAL Language. For more detailed definitions of the regular expression elements listed below, refer to their descriptions in the Perl 5.004 Regular Expression documentation. A copy of this documentation has been preserved for reference purposes [10]. Regular expression elements that are not listed below should be avoided as they are likely to be incompatible or have different meanings with commonly used regular expression libraries.

Please note that while only a subset of the Perl 5 regular expression syntax is supported, content can be written that may still run in some OVAL interpreter tools. This practice should be avoided in order to maintain the portability of content across multiple tools. In the event that an attempt was made to evaluate a string against a malformed regular expression, an error must be reported. An example of a malformed regular expression is the pattern "+". An unsupported regular expression should only be reported as an error if the evaluating tool is not capable of analyzing the pattern. A malformed
A regular expression may remain ignored if the preceding existence check can determine the evaluation flag.

### Table 14: Metacharacters

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>Quote the next metacharacter</td>
</tr>
<tr>
<td>^</td>
<td>Match the beginning of the line</td>
</tr>
<tr>
<td>.</td>
<td>Match any character (except newline)</td>
</tr>
<tr>
<td>$</td>
<td>Match the end of the line (or before newline at the end)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>()</td>
<td>Grouping</td>
</tr>
<tr>
<td>[]</td>
<td>Character class</td>
</tr>
</tbody>
</table>

### Table 15: Greedy Quantifiers

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Match 0 or more times</td>
</tr>
<tr>
<td>+</td>
<td>Match 1 or more times</td>
</tr>
<tr>
<td>?</td>
<td>Match 1 or 0 times</td>
</tr>
<tr>
<td>{n}</td>
<td>Match exactly n times</td>
</tr>
<tr>
<td>{n,}</td>
<td>Match at least n times</td>
</tr>
<tr>
<td>{n, m}</td>
<td>Match at least n but not more than m times</td>
</tr>
<tr>
<td>Quantifier</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>*?</td>
<td>Match 0 or more times</td>
</tr>
<tr>
<td>+?</td>
<td>Match 1 or more times</td>
</tr>
<tr>
<td>??</td>
<td>Match 0 or 1 time</td>
</tr>
<tr>
<td>{n}?</td>
<td>Match exactly n times</td>
</tr>
<tr>
<td>{n,}?</td>
<td>Match at least n times</td>
</tr>
<tr>
<td>{n,m}?</td>
<td>Match at least n but not more than m times</td>
</tr>
</tbody>
</table>

Table 16: Reluctant Quantifiers

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tab (HT, TAB)</td>
</tr>
<tr>
<td>\n</td>
<td>newline (LF, NL)</td>
</tr>
<tr>
<td>\r</td>
<td>return (CR)</td>
</tr>
<tr>
<td>\f</td>
<td>form feed (FF)</td>
</tr>
<tr>
<td>\033</td>
<td>octal char (think of a PDP-11)</td>
</tr>
<tr>
<td>\x1B</td>
<td>hex char</td>
</tr>
<tr>
<td>\c[</td>
<td>control char</td>
</tr>
</tbody>
</table>

Table 17: Escape Sequences

<table>
<thead>
<tr>
<th>Character Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\w</td>
<td>Match a &quot;word&quot; character (alphanumeric plus &quot;_&quot;)</td>
</tr>
<tr>
<td>\W</td>
<td>Match a non-word character</td>
</tr>
<tr>
<td>\s</td>
<td>Match a whitespace character</td>
</tr>
<tr>
<td>\S</td>
<td>Match a non-whitespace character</td>
</tr>
<tr>
<td>\d</td>
<td>Match a digit character</td>
</tr>
<tr>
<td>\D</td>
<td>Match a non-digit character</td>
</tr>
</tbody>
</table>

Table 18: Character Classes

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Match a word boundary</td>
</tr>
<tr>
<td>\B</td>
<td>Match a non-(word boundary)</td>
</tr>
</tbody>
</table>
### Table 19: Zero Width Assertions

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?:regexp)</td>
<td>Group without capture</td>
</tr>
<tr>
<td>(?=regexp)</td>
<td>Zero-width positive lookahead assertion</td>
</tr>
<tr>
<td>(?!regexp)</td>
<td>Zero-width negative lookahead assertion</td>
</tr>
</tbody>
</table>

### Table 20: Extensions

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[chars]</td>
<td>Match any of the specified characters</td>
</tr>
<tr>
<td>![chars]</td>
<td>Match anything that is not one of the specified characters</td>
</tr>
<tr>
<td>[a-b]</td>
<td>Match any character in the range between &quot;a&quot; and &quot;b&quot;, inclusive</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>\n</td>
<td>When 'n' is a single digit: the nth capturing group matched</td>
</tr>
</tbody>
</table>

### Table 21: Version 8 Regular Expressions

<table>
<thead>
<tr>
<th>Authors' Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael Cokus</td>
</tr>
<tr>
<td>The MITRE Corporation</td>
</tr>
<tr>
<td>903 Enterprise Parkway, Suite 200</td>
</tr>
<tr>
<td>Hampton, VA 23666 USA</td>
</tr>
<tr>
<td>Email: <a href="mailto:msc@mitre.org">msc@mitre.org</a></td>
</tr>
</tbody>
</table>

|                  |
| 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