CVE, CME,. . . , CMSI? – Standardizing System Information

Bernd Grobauer*

Abstract

During the last few years, a clear trend towards standardized names and exchange formats could be observed in the world of IT security. For example:

- **Vulnerability Information**: CVE, a list of standard names for *Common Vulnerabilities and Exposures* [8] allows the IT-security community to cross-reference information about vulnerabilities. The EISPP/DAF format [3, 4] – in productive use by several CERTs within Europe – allows exchange of security-advisory information.
- **Incident Information**: The IODEF format [6] is used for exchanging incident information between CERTs.
- **Vulnerability Checks, Remediation and Avoidance**: OVAL [9] is a standardization effort regarding executable descriptions of vulnerability checks. OVAL descriptions can, for example be integrated into NIST's extensible configuration checklist description format (XCCDF) [12].
- **Malware Information**: Recently, the US-CERT announced an initiative to introduce CME, a *Common Malware Enumeration* to allow the cross-referencing between different names for the same malware artifact.

All these standardizations ease co-operation and re-use regarding security-related issues. A problem that has not been tackled so far is the standardization of system information. Similarly to CVE, system information is orthogonal to other information exchange formats: Which systems are affected by the vulnerability described in an advisory? What kind of system was involved in a security incident? For which kind of system is a vulnerability check or configuration setting applicable?

As CVE did, a common naming scheme for (machine-readable) system information would increase the potential of standards for information exchange: automated handling based on system information, e.g., for statistical purposes, correlation and filtering, becomes possible.

Can a common naming scheme for system information be established? This article describes the approach taken by a group of German CERTs towards a common model of system information (CMSI).

1 Introduction

At time of writing, suppliers of vulnerability information (security advisories, vulnerability databases, etc.) and incident response teams use proprietary models for specifying system information: suppliers of vulnerability information need to inform about which systems are affected by a new vulnerability, incident response teams need to communicate which type of system was attacked during an incident. Without a model of some kind, system information cannot be provided *consistently*: a product that was called "*Microsoft Explorer*

*Siemens AG, Corporate Technology, Information & Communications Division, Siemens CERT. Email: `bgrobauer@cert.siemens.de`

V6.0" in yesterday's advisory, should not be referred to as "*MS Internet Explorer (version 6.00)*" in today's advisory. Obviously, a record of established identifiers for systems must be kept and consulted whenever system information is to be specified. For many purposes an informal record may be enough: by cutting and pasting (human readable) system information from older advisories or incident reports, a certain degree of consistency can be kept. If, however, reliable methods for filtering or other automated manipulations of system information are to be used, a dictionary of identifiers and a set of syntactic rules are indispensable. Such a dictionary clarifies once and for all that Microsoft's browser product is to be called, say, "*Internet Explorer*" and maybe defines an additional purely machine-readable identifier for this product. Similarly, the model may prescribe a standard way for specifying more detailed product information such as the product version – for the example presented above, either the form "`Vx.y`" or "`version x.yz`" could be prescribed. In this article, we refer to such a dictionary and associated rules as a *model of system information.*

This article describes the first step of an approach taken by a group of German CERTs towards a *common* model of system information (CMSI): an XML-schema for describing a model of system information. The intended use of CMSI is the following:

- Using the CMSI XML schema, a model of system information can be defined and made available, e.g., by placing the resulting XML file on a web server for download.
- The XML file specifies recognized components of IT systems and informs about the format in which detailed information about such components (patch level, language version, etc.) must be described.
- Following the definitions made in the XML file, machine-readable system information can be included in structured information such as advisories following the EISPP/DAF [3, 4] advisory standard.
- Once machine-readable system information is available, advanced applications such as filtering out irrelevant advisories or creating user profiles for advisory services can be implemented.

It would be advantageous, if one instance of a common model of system information could be agreed upon – to be used, for example, for information exchange between CERT organizations. Using the CMSI XML schema, a working group of German CERT organizations has started to define a common model of system information, examples from which are cited within this paper. The common model defined by German CERTs could be a good starting point for defining a 'single' common model of system information. If that proves to be infeasible, the work on CMSI presented in this paper should still be useful to anybody with the need of making system information machine readable: CMSI offers a thought-out approach to standardizing system information.

The remainder of this paper is structured as follows: Section 2 presents a concise introduction to CMSI, Section 3 gives a detailed explanation, Section 4 describes possible applications of CMSI, and Section 5 concludes.

2 A feasible Common Model

The most successful information model used within the IT-security community is CVE. Regarding structure and content, there are at least two significant differences between standardizing vulnerability names and standardizing system information.

Requirements on Structure The simple structure of CVE, which uses a flat list of vulnerability identifiers, is not suitable for specifying system information. System information requires more structure and the possibility to move between very coarse and very detailed information: a system may very well

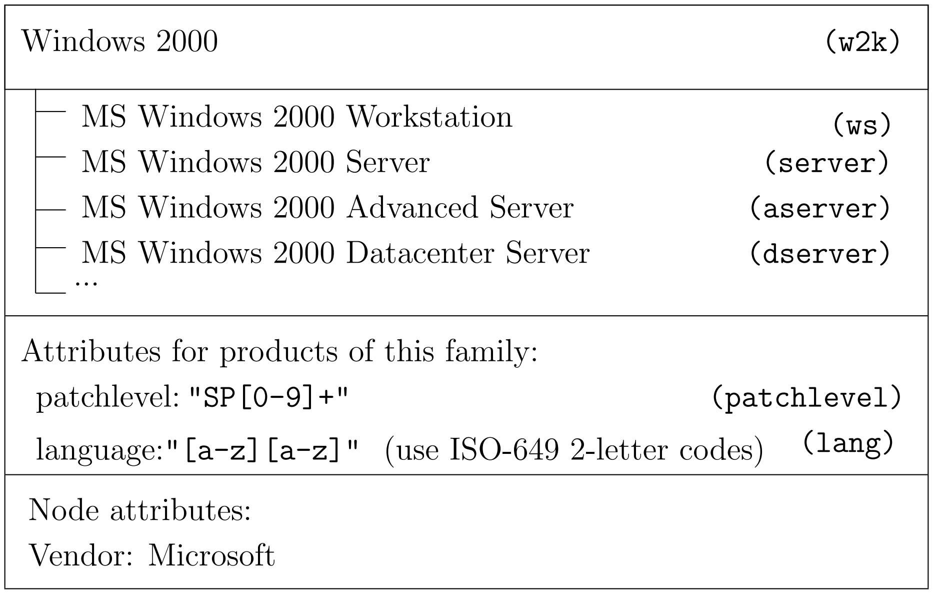
Windows 2000 (w2k)
MS Windows 2000 Workstation (ws)
MS Windows 2000 Server (server)
MS Windows 2000 Advanced Server (aserver)
MS Windows 2000 Datacenter Server (dserver)
...
Attributes for products of this family:
patchlevel: "SP[0-9]+" (patchlevel)
language:"[a-z][a-z]" (use ISO-649 2-letter codes) (lang)
Node attributes:
Vendor: Microsoft

Figure 1: Example of how the product family "Windows 2000" could be represented in CMSI.

be coarsely specified as "*Windows*", more accurately as "*Windows 2000*", and in a very detailed way as "*Windows 2000 Professional, SP2, German language version*". To be useful, a common model of system information must cater for all these levels. Equally important: if very detailed system information is received, but only coarse information (e.g., *Windows* vs. *Unix*) is wanted, the coarse information must be easily extractable.

Both requirements – additional structure and several levels of detail – can be implemented using a tree-like structure rather than a flat list.

Requirements on Content The world of products is more dynamic than the world of vulnerabilities, or rather dynamic in a different way: while every new vulnerability constitutes an entity of its own, existing products change by the release of new versions, name changes, etc. What is more: the way, in which version information and other attributes of interest are given varies from product to product. Thus, for a feasible model of system information, a way must be found to allow precise specification of such attributes, yet keep the model maintainable.

The CMSI Approach Based on the considerations presented above, CMSI has been designed as a *category tree*, whose leave nodes are formed by *product families*, which cluster closely related products. The model does not contain lists of acceptable values for attributes (e.g., a list of existing versions), but instead prescribes syntactic rules. Reusable syntax definitions for frequently used attributes – such as common styles of version numbering – are given. The clustering of closely related products into product families reduces complexity with respect to attribute definitions, because for closely related products, attributes such as version information are usually specified in the same way.

An Example Figure 1 gives an excerpt of the definition of the product family `Windows`

2000: the products belonging to the family are given and the relevant attributes are described. For the family as a whole, each product and each attribute, a machine-readable tag is defined. The syntax of how to specify each attribute is defined using regular expressions (Unix-style notation); if necessary, an explaining comment can be added. Thus, the machine-readable representation of a German-language Windows Server 2000 system with SP2 could be given as

```
platform "w2k:server"
 patchlevel "SP2"
 lang "de"
```

In practice, however, probably XML would be used to package the machine-readable system information. The default format to be used for CMSI is already part of the EISPP/DAF [3, 4] advisory standard (Section 4 gives an example) that could as well be integrated into other XML-based formats.

Figure 2 shows excerpts of the category tree that has been defined by a working group of German CERTs. The family `Windows 2000` is located under the category node `Windows`: thus, one can move from a very detailed to a rather coarse description simply by discarding attributes and walking up the tree. Assuming, that the machine-readable tag for the node `Windows` is '`win`,' coarse system information about the operating system could be given as '`platform "win"`.' That may be useful, for example, as part of a machine-readable representation of "*Apache 2.0 on Windows operating systems.*"

3 Detailed Explanation of CMSI

In the following, the proposal for a common model of system information as outlined in Section 2 is explained in more detail. Before turning to a formal definition of CMSI, we give some more information on contents and structure.

```
OS
   Mainframe
   Mobile Computing
   Network
     Cisco
   Telecommunication
   Unix-like
     Unix
     Linux
   Windows

Application
   Business
   Client
     Webbrowser
     Mail and Groupware
     Remote Access
     Multimedia
     File Transfer
   Office
     Adobe
   Security
     Antivirus
     Crypto
     Firewall
       Personal Firewalls
     IDS and Monitoring
       Host based IDS
     VPN
   Server
     Application
     Backup
     Database
       MS SQL Server
     Directory
     File
     Infrastructure
      DNS
     Mail and Groupware
     Network Management
     Proxy
     Remote Access
     System Management
     Web
```

Figure 2: Current contents of the top-level nodes `OS` and `Application` of the CMSI category tree. Product families are not shown. With the addition of further product families, subnodes may be added to cluster related product families. For example, the node `MS SQL Server` clusters the families `MS SQL Server (pre 2000)`, `MS SQL Server 2000`, and `MS SQL Server 2005`.

3.1 Contents and Structure

CMSI attempts to bring some order into the diverse world of IT products that form parts of IT systems using a *category tree*, whose leaves are formed by *product families*; the latter cluster closely related products.

3.1.1 Product families and Products

A product family comprises one or more closely related products, hence a vulnerability affecting one of these products is very likely to affect other members of the same family, as well. For practical reasons, a product family defined within the common model should not contain more than, say, a dozen or so products. For large product families, usually a sensible partition into several smaller families can be found. Consider the definition of a family `MS Windows` as an example. This family would contain several dozens products such as `Windows 2000 Advanced Server`, `Windows 98SE`, etc. Instead, several families such as `Windows 95/98/ME`, `Windows NT`, `Windows 2000`, and `Windows XP` should be defined.

The relationship between products and their respective families can be expressed with a single level of hierarchy within the common model: Each product is associated to a product family. Because the members of a product family are closely related, usually the same categories of information regarding version, patch levels, built, etc., will be applicable. Therefore, in CMSI, information about what kind of information can be associated with a product and corresponding syntactic rules is defined on the level of product families.

Let us revisit Figure 1: With the name of the family, the products belonging to the family are listed. Additionally, a set of attributes necessary to specify affected products in detail is defined. Because the products within the product family are rather similar, the same kind of additional information to completely specify a product is necessary, in this case the patch level in form of information about service packs and the language version of the product in question.

3.1.2 A (shallow) product tree

A common model based on product families will easily grow to at least several hundreds, probably thousands of families. The model will only be useful if families can be located easily. An obvious measure is to implement the dictionary of families and products such that search operations are possible – especially if alternative names are stored with families and products, most searches for products included in the common model will be successful.

Additionally, structure must be added to the flat name space of product families by defining a tree-like structure. Let us revisit Figure 2, which displays fragments of a possible tree structure; the leaves always correspond to product families. The tree is rather shallow: At the moment, no more than four levels are used – a more deeply nested structure is very likely to create more confusion than help organize product families.

Already with a shallow tree, it is not always quite clear, where a product family should be placed. Consider MS Exchange, which has the functionality of an email server, but much else besides. A user browsing the tree for MS Exchange therefore might not immediately search under `Server` and `Email`. For this reason, instead of a strict tree, a more relaxed representation is chosen: product families can be linked into the tree structure more than once. MS Exchange, for example, could also be a direct descendant of the `Application` node or some `Groupware` node that might exist. Thus, a user navigating the tree would have better chances of finding product families where she expects them.

If we examine the various nodes within the tree fragment, we encounter three types of nodes:

Structural Nodes are added to the tree to provide structure, such as `OS` and `Application`.

Relation Nodes group product families that are closely related. This happens especially with operating systems: consider the `Windows` and `Linux` nodes.

Profiling Nodes also provide structure, but additionally might be interesting for profiling purposes. For example, server administrators might be especially interested into product families to be found somewhere under the `Server` node.

As mentioned in Section 2, not only products and product families can be used for specifying system information: by walking up the tree, coarser information can be given, usually with relation nodes such as `Windows`.

3.1.3 Node Attributes

One aspect of information is still missing: what about information to be associated with nodes (category nodes or leaves, i.e., product families) rather than an actual instance of a product? Consider, for example, information that is fixed for all products within a product family such as vendor information. Obviously, defining a vendor attribute in the same way as the patch-level attribute and language-version attribute does not make sense: patch level and language version vary from case to case, while the vendor stays always the same.

The mechanism to integrate useful, (mostly) static information into a model of system information are node attributes, which are defined as part of the system's meta data. Node attributes must be filled in for those nodes for which the information is applicable. For example, a vendor attribute should be filled in for almost all product families. A platform attribute, on the other hand, would be meaningful only for category nodes or product families representing applications: this way, information about which application runs on which operating systems could be integrated into the model.

3.1.4 Attribute Type Definitions

Above, we have seen that the `Windows 2000` product family specifies, how information about service packs can be given. Obviously, the very same definition would be present in other product families such as `Windows NT`, `Windows XP`, and `Internet Explorer`. To reduce complexity of the model and support consistency within the model, such definitions must be reused rather than rewritten for every applicable product family. Therefore, CMSI allows the specification of *attribute type definitions* as reusable attribute definitions within product families.

3.2 The Formal Definition of CMSI

CMSI uses XML as underlying description language: the current XML-DTD is available via the CMSI home page [2] on the web site of the "Deutscher CERT-Verbund" (German CERT association). Here, we use a pseudo-formal notation in BNF to explain how CMSI is used. Figure 6 on page 11 shows an excerpt of the model defined by the German CERT working group in XML syntax.

3.2.1 Top-level structure

A CMSI-definition file has the following top-level structure:

```
SysInfoModel ::=  <type def.>*
                  <node attr.>*
                  <category tree>
                  <family>*
```

Before a category tree and product families can be described, meta data has to be defined:

- *Attribute type definitions* are used to constrain attributes – node attributes and attributes used within product families. For example, an attribute to specify a patch level (see the example in Figure 1) is certainly used in several families; its type should be defined once and for all:

the model is kept consistent and reuse reduces complexity.

- *Node attributes* can be used to define content that should be filled out for every category node and product family. In the example in Figure 1, the vendor attribute is filled out to be "*Microsoft.*" Not every node attribute will be applicable to every node: for the category node `OS`, the vendor attribute will remain empty, while for the category node `MS SQL Server` (see Figure 2), it is to be filled out as "*Microsoft*".

The meta data is followed by a definition of the category tree and the product families.

3.2.2 CMSI meta data

```
type def ::= <type id>
             <descr.>?
             (   (grammar <grammar>)
               | (LoV  <value list>)
               |  nodes
             )

  where

   value list ::= (<value tag>
                   <name>
                   <descr.>?)+

node attr ::=  <attr. id>
               <type id>
               <name>
               <descr.>?
```

Figure 3: Explanation of CMSI syntax in pseudo-formal notation: CMSI meta data.

Figure 3 details the definition of CMSI meta data. Every attribute type definition contains a unique type identifier for cross-reference within the model, a description (optional) and information about which kind of type is defined:

- A type can be defined *grammatically* as a regular expression as used, for example, in the programming language Perl. The type of the language attribute used within the product family `Windows 2000` (see Figure 1), for example, specifies the regular expression `[a-z][a-z]`, i.e., a two-letter code. The description field of the type definition is used to inform about the semantics of the code: a ISO-639 two-letter language code is to be used.
- A type can be defined by a *list of values* (LoV). To fill in an attribute of such a type, one or more items from a predefined list of values must be chosen. An item in such a value list consists of a machine-readable tag, a human-readable name, and an optional description.
  For example, the type of the vendor attribute is defined as a *list of values.* One item in this list has the name "*Microsoft*" and machine-readable tag `ms`. In this case, a description probably is not necessary – for smaller vendors, though, a comment with a pointer to the vendor's web site could be useful.
- A type can be defined as a set of *nodes.* For example, the platform attribute given above as an example for a possible node attribute could be realized by referencing nodes representing operating systems. At the moment, CMSI offers no way to constrain the set of nodes that can be used: further experiences with CMSI will show, whether the possibility to give such constraints is necessary.

The list of node attributes makes use of the defined attribute types. Each node attribute is specified with a unique identifier, a name, a type identifier referencing the attribute type to be used, and – optionally – a description.

3.2.3 Category tree and product families

Figure 4 details the definition of the category tree and the product families.

The definition of the category tree is

```
category tree ::= <category node>*

category node ::= <node tag>
                  <name>+
                  <descr.>
                  <node attr. value>*
                  <children>

node attr. value ::= <attr. id>
                     <value>+

children ::= <category node>*

family ::= <family tag>
           <parents>
           <name>+
           <descr.>?
           <node attr. value>*
           <product>*
           <product attr.>+

parents ::= <node tag>+

product ::= <product tag>
            <name>+
            <descr.>?

product attr. ::= <attr. tag>
                  <type id>
                  <name>
                  <descr.>?
```

Figure 4: Explanation of CMSI syntax in pseudo-formal notation: category tree and product families.

straightforward: a machine-readable tag is followed by one or more names (we allow synonyms), an optional description, a list of attribute values for this node, and a list of category nodes that are the node's children. An attribute value is given by referencing the unique identifier of a node attribute and listing one or more values of the proper type.

The definition of a product family starts with a unique machine-readable tag and continues with a list of category nodes that are the parents of a product family. The rationale behind allowing product families to be located under more than one category node is that a perfect categorization in which there would be one and only one logical place for a given product family is unrealistic. Although the possibility to associate a product family with more than one parent node should not be overused, ambiguities in the categorization can be resolved this way.

Just as for category nodes, we allow more names for a product family, which is useful especially in the case of changing product names: the old name, which may still be in common use, does not have to be discarded but can be kept in the model. Similarly to category nodes, a description can be given and values for applicable node attributes can be specified.

A product family contains products defined by a machine-readable tag unique within the product family, one or more product names and an optional product description. The machine-readable tag of a product is derived by joining the product family's tag with the product's tag using a colon ":". Single products are represented by a product family containing no product definitions: the product corresponds to the product family.

Similarly to the top-level definition of node attributes, product-family-specific attributes can be defined.

4 Applications of CMSI

As has been argued in the introduction, even a single organization dealing with system information needs some model of system information. The model may very well be informal and still be sufficient for applications such as issuing advisories that only contain human-readable information. Applications such as automated filtering based on user profiles or configuration-management data, however, are out of the question without a formal model of system information.

CMSI has grown out of an effort to standardize security-advisory information within the EISPP [4] project, now continued as DAF [3]. Figure 5 shows an example of how system information based on the model defined by a work group of German CERTs could be used within a DAF-advisory: The system information says that Apache 1.3.x and 2.0 on Windows 2000 and Windows XP, and Apache 2.x on Unix-like systems are affected.

Note that abbreviations such as `1.3.x` must be given a well-defined semantics fit for the implementation of filter mechanisms recognizing, e.g., the version `1.3.24` as instance of the shorthand notation `1.3.x`. The necessity to define proper semantics for all attributes underlines the importance of reusing attribute definitions as implemented by attribute type definitions within CMSI.

```
<system_list cat_model="german_cert_wg">
 <system>
  <system_part type="platform">
   <instance tag="w2k"/>
   <instance tag="wxp"/>
  </system_part>
  <system_part type="software">
   <instance tag="apache">
    <attribute_value tag="version">
     <value>1.3.x</value>
     <value>2.x</value>
    </attribute_value>
   </instance>
  </system_part>
 </system>
 <system>
  <system_part type="platform">
   <instance tag="unix"/>
  </system_part>
  <system_part type="software">
   <instance tag="apache">
    <attribute_value tag="version">
     <value>2.x</value>
    </attribute_value>
   </instance>
  </system_part>
 </system>
</system_list>
```

Figure 5: Example of machine-readable system information as part of a DAF-advisory. The system information says that Apache 1.3.x and 2.0 on Windows 2000 and Windows XP, and Apache 2.x on Unix-like systems are affected.

Also other standards could make good use of a common model of system information:

- The CVE-based databases ICAT [5] and Cassandra [1] use proprietary models of system information. The use of an established common model to express system information would certainly provide added value to the users of these databases.
- IODEF [6] is used for the exchange of incident information. With the use of machine-readable system information in IODEF, automated correlation and generation of statistics becomes possible.
- XCCDF [12], a standardized language for specifying configuration checklists, recognizes the need to provide structured system information: a subcomponent of the XCCDF XML schema, the CIS platform schema [11], specifies how local system definitions can be undertaken in a given checklist. A common model of system information could tie together different checklists written in XCCDF.
- Data collected as part of the projected common malware enumeration (CME) would certainly contain system information. Again, providing such information based on a common model would provide added value and enable automated techniques such as correlation, filtering and generation of statistics.

Another promising application area for CMSI is configuration management. Complex software systems may contain 50 and more OEM software components. Vendors of such systems must keep track of the exact configuration, for example to react to vulnerability information concerning components used within the system. Ideally, configuration data maintained in a CMSI-compatible format could be used as profile for filtering advisory information containing system information based on CMSI.

5 Conclusion

So far, the following milestones regarding CMSI have been achieved:

- An XML schema for describing the contents of the common model (category tree and product families) has been defined. It is described in this paper and available from the CMSI home page [2].
- An XML schema for including system information based on CMSI has been defined and is already part of the EISPP/DAF [3] advisory exchange formats.
- CMSI has been tightly integrated into the open-source development of the incident handling system SIRIOS [7, 10], a project of CERT Bund. SIRIOS also supports IODEF and EISPP/DAF.
- A standard category tree has been agreed upon for use within the German CERT community. At time of writing, this category tree is being filled with the most important product families.

Already at this stage, CMSI – whose structure is based on a careful analysis of requirements and constraints regarding models of system information – can be applied for proprietary models. Now, future work must focus on establishing a common model of system information for a user-base that is as broad as possible. Co-operation of interested parties regarding the design of a standardized category tree and standardized product families would provide the chance to close a gap left by all standardization efforts in the field of information security so far.

The working group of German CERTs will continue to build up a workable model for common use within the German CERT association. However, a broader base of maintainers and users would of course be highly desirable – one way to achieve this could be the maintenance of a common model under the auspices of an organization such as FIRST.

Acknowledgments

I am indebted to the members of the *Advisory Format Interest Group* of the *Deutscher CERT-Verbund*, participants of the EISPP project, and my colleagues from Siemens CERT for fruitful discussions on the subject. Special thanks are due to Andreas Bunten of DFN-CERT and Thomas Klingmüller from CERT-Bund for their timely comments on an earlier draft of this article.

References

[1] Cassandra homepage. See `https://cassandra.cerias.purdue.edu`.

[2] CMSI homepage. See `http://www.cert-verbund.de/cmsi`.

[3] DAF Advisory Format Description. Available from `http://www.cert-verbund.de/daf/daf_description.html`.

[4] EISPP Common Advisory Format Description, v2.0. Available from `http://www.eispp.org`.

[5] ICAT Metabase. See `http://icat.nist.gov/`.

[6] Incident Object Description and Exchange Format. See `http://www.ietf.org/html.charters/inch-charter.html`.

[7] Thomas Klingmüller. SIRIOS – a framework for CERTs. In *Proceedings of the 17th Annual FIRST Conference*, 2005.

[8] David E. Mann and Steven M. Christey. Towards a common enumeration of vulnerabilities, January 1999. Available from `http://cvs.mitre.org/docs/cerias.html`.

[9] OVAL – Open Vulnerability and Assessment Language. See `http://oval.mitre.org/index.html`.

[10] SIRIOS homepage. See `http://www.cert-verbund.de/sirios`.

[11] David Waltermire and Neal Ziring. CIS platform schema, 2004. Available from `http://csrc.nist.gov/checklists/docs/platform-0.2.3.xsd.txt`.

[12] Neil Ziring. Specification for the extensible configuration checklist description format (XCCDF). Technical Report NISTIR 7188, NIST, 2005. Available from `http://csrc.nist.gov/checklists/xccdf.html`.

```
...
<TypeDefs>                                                                  (* Type Definitions *)
  <TypeDef id="vendor_lov" kind="LOV">                              (* Type for specifying vendors *)
   <ValueList>
    <ListElement tag="ms" name="Microsoft"/>
    <ListElement tag="suse" name="SuSE"/>
     ...
   </ValueList>
  </TypeDef>
  <TypeDef id="os_nodes" kind="nodes">                             (* Type for specifying OS nodes *)
   <Description>Category nodes or product families that
                refer to operating systems</Description>
  </TypeDef>
  <TypeDef id="service_pack" kind="grammar">       (* Type for specifying patch level as service pack *)
   <Description>The patch level, e.g., of Windows OSes is specified
                with a service pack number of form
                "SP1", "SP1a", "SP2", "SP3", possibly followed by a single lower-case letter.
                No leading zeros (e.g., "SP01") are used!</Description>
   <Grammar><![CDATA[SP[0-9]+[a-z]?]]></Grammar>
  </TypeDef>
</TypeDefs>
...
<NodeAttributes>                                                            (* Node Attributes *)
  <NodeAttribute id="vendor" type_id="vendor_type" name="Vendor">      (* Node attribute 'Vendor' *)
   <Description>This attribute specifies the vendor of a product family.</Description>
  </NodeAttribute>
  <NodeAttribute id="os" type_id="os_nodes" name="Platform">         (* Node attribute 'Platform' *)
   <Description>This attribute only applies to applications. It specifies all
            the platforms a given application runs on.</Description>
  </NodeAttribute>
  ...
</NodeAttributes>
...
<Families>                                                                  (* Product Families *)
 ...
  <Family tag="w2k">                                          (* Product Familiy Windows 2000 *)
   <Parents>
     <Parent tag="win"/>
   </Parents>
   <Names>
     <Name>Windows 2000</Name>
   </Names>
   <Description>The Windows 2000 product family</Description>
   <Products>
     <Product tag="server">                                    (* Product 'Windows 2000 Server *)
       <Names>
         <Name>Windows 2000 Server</Name>
       </Names>
       <Description></Description>
     </Product>
     ...
   </Products>
   <AttributeValues>
     <AttributeValue attribute_id="vendor">              (* Filled-in value for node attribute 'Vendor' *)
       <Value>ms</Value>
     </AttributeValue>
   </AttributeValues>
   <ProductAttributes>                         (* Specification of family-specific attribute 'Service Pack' *)
     <ProductAttribute tag="patchlevel" type_id="service_pack" name="Service Pack"/>
     ...
   </ProductAttributes>
  </Family>
  ...
</Families>
...
```

Figure 6: Excerpts from the model of system information defined by a working group of German CERTs. The only component of which nothing is shown is the category tree, whose definition is rather straightforward.