

International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.1206

(04/2008)

SERIES X: DATA NETWORKS, OPEN SYSTEM
COMMUNICATIONS AND SECURITY

Telecommunication security

**A vendor-neutral framework for automatic
notification of security related information and
dissemination of updates**

Recommendation ITU-T X.1206



ITU-T X-SERIES RECOMMENDATIONS
DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.379
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.889
Generic applications of ASN.1	X.890–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
TELECOMMUNICATION SECURITY	X.1000–

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T X.1206

A vendor-neutral framework for automatic notification of security related information and dissemination of updates

Summary

Recommendation ITU-T X.1206 provides a framework for automatic notification of security related information and dissemination of updates. The key point of the framework is that it is a vendor-neutral framework. Once an Asset is registered, updates on vulnerabilities information and patches or updates can be automatically made available to the users or directly to applications regarding the Asset.

Source

Recommendation ITU-T X.1206 was approved on 18 April 2008 by ITU-T Study Group 17 (2005-2008) under the WTSA Resolution 1 procedure.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2009

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

CONTENTS

	Page
1 Scope	1
2 References.....	1
3 Definitions	1
3.1 Terms defined in this Recommendation.....	1
4 Abbreviations.....	2
5 Conventions	2
6 Introduction	2
7 Current situation regarding vulnerability information.....	3
8 Overview of vendor-neutral framework	5
8.1 Multiple sources of vulnerability information, updates and patches.....	5
8.2 Example application operation	6
8.3 Security and privacy considerations	7
9 Recommendation architecture	7
9.1 Message core layer	7
9.2 Message/application layer	8
9.3 Scalability	8
9.4 Extensibility.....	8
9.5 Platform independence	9
9.6 Client/Server communication.....	9
10 Components of the framework	9
10.1 Message container	9
10.2 Version message.....	13
11 Schemas	19
11.1 Message_Core	19
11.2 Message_Version	21
Bibliography.....	26

Recommendation ITU-T X.1206

A vendor-neutral framework for automatic notification of security related information and dissemination of updates

1 Scope

This Recommendation provides a framework of bidirectional flow of automatic notification and distribution of vulnerability information as well as the distribution of updates and/or patches. In addition, this Recommendation makes it possible for system administrators to know the condition of any Asset within their realm of responsibility.

Clauses 6 and 7 describe the problems of maintaining Assets from an Asset identification point of view, as well as information dissemination and systems/networks management points of view.

Clause 8 describes the overview of the vendor-neutral framework, which includes an example system supported by the adoption of the framework, compartments of the framework and an exemplary sequence of exchanges within the framework. Clause 8 also describes the security that should be considered in the vendor-neutral framework.

Clause 9 describes the functionalities and features of this Recommendation.

Clause 10 provides the definitions of the data structures of components of this Recommendation.

Clause 11 contains the XML schema defined and described in clause 10.

This Recommendation provides a framework that any vendor can use for notification, as well as the receiving of vulnerability information and dissemination of required patches/updates for covered Assets, and defines the format of the information that should be used in and between components implementing this framework.

This Recommendation does not define protocols to be used in the communication between components as many protocols are supported without special consideration.

Some common roles and responsibilities will be needed to be established for operation based on the vendor-neutral framework; however, a discussion regarding the establishment and operation of possible roles and their resulting responsibilities is not within the scope of this Recommendation.

2 References

None.

3 Definitions

3.1 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.1.1 agent: An implementation of this Recommendation operating in support of an installed asset on a given device, in support of server functionality or in support of local server functionality.

3.1.2 asset: A device, separately identifiable piece of hardware, application, operating system or instance of executable code.

3.1.3 client: A device which requests services from another device.

3.1.4 device: A system acting as either a client, server, or both, local server.

3.1.5 group: A number of devices operated on as a single unit.

3.1.6 local server: A client acting as a server node for additional downstream clients.

3.1.7 message: A request for a specific action to be performed, e.g., general actions such as "Register" an asset as being of a given version and/or contained components of given versions, "Request" existing or future available updates, patches or vulnerability information, etc. Messages extending the functionality of this Recommendation may be defined outside the scope of this Recommendation.

3.1.8 message data: Information provided in support of a given message. Among an almost infinite number of possibilities, specific examples defined in this Recommendation are data defining version information, vulnerability information pertaining to given versions as well as updates or patches for specific versions.

3.1.9 message set: A combination and association of a universally unique identifier, a message and the message's associated message data's definition, all defined within an XML schema derived from and extending *Message_Core* defined herein.

3.1.10 patch: A broadly released fix for a product-specific, security-related vulnerability. A method of updating a file that replaces only the parts being changed, rather than the entire file.

3.1.11 server: A device used to service requests from other devices.

3.1.12 vulnerability: Any weakness, administrative process or act, or physical exposure that makes a computer or network of computers susceptible to exploit by a threat.

4 Abbreviations

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
GUID	Globally Unique IDentifier
HTTP	HyperText Transfer Protocol
ISIRT	Information Security Incident Response Team
ISP	Internet Service provider
OS	Operating System
POAS	Platform/Operating System/Application/Service
URI	Uniform Resource Identifier

5 Conventions

None.

6 Introduction

As more people begin to use computers in their homes and workplaces, and fewer have any kind of official training in the operation of their computers, let alone security-related issues, one quickly approaches the point where security, not only becomes almost impossible to maintain, but it becomes more difficult for those responsible for maintaining security on a system level to know much about the condition of the systems they are responsible for and provide services until some breach or accident occurs, by which time it is already too late.

That is primarily due to the fact that so many different computers are in different states of maintenance and update. Where security-related issues are concerned, system management is much less of a preventative process than of a disaster management and recovery process.

Although a number of applications and even operating systems (OS) have their own update mechanisms, they all have a number of problems in common. One such problem is that all the update mechanisms rely on being enabled, in the first place, and, in the second, allowed to do their job when the user is notified of an update being available, assuming that the user has enabled the notifications.

Likely worst of all though is that these problems leave system administrators totally out of the picture, so that without installing their own monitoring systems on each computer under their responsibility, they have no idea as to the general level of security within the networks and systems they are responsible for.

Another consideration is that while updating software to the latest available it is often the case where updates alone are not the solution, but instead improved usage practices for which no update, other than the information being received by the end user, is of use. Even though various applications and OSs may have updating mechanisms in place, none of them has a uniform method of keeping users informed of the latest best practices leading to continued secure use.

Also of importance are the methods used to distribute updates. Currently, all updates through the various update mechanisms source the updates through dedicated channels, one for each update session. But where updates, or other important information made available to various redistribution centers, e.g., ISPs or corporate networks, are subsequently distributed within the networks in a trusted and secure manner, the bandwidth required for distribution could effectively be cut in half or at least reduced to a great extent just as in the case of hypertext transfer protocol (HTTP) proxies.

Another concern that is mismanaged in most cases is where users themselves find a problem regarding the use or actions of a given asset, without having anyone to refer the problem to. Even if some method is in place for users to contact system administrators or other support personnel, using such a system ends up more like a game of "20 questions" where the user and support person must go back and forth asking for information and replying usually with more questions.

It is often difficult for even an experienced support person to have a clear understanding of what exactly comprises many assets. Due to modular software architectures and various pieces of software actually comprising modules from different vendors, all with their own versioning systems, even if an update or relevant piece of information about a given product or sub-module is made available, knowing to what and to whom it applies can be a difficult task, often leading the less informed to ignore much of what their systems may rely for security.

In the end, one ends up with users being uninformed and those responsible for system wide security being essentially left out of the loop.

7 Current situation regarding vulnerability information

Vulnerability information is presently released by many vendors and many security-related organizations, such as the Information Security Incident Response Team (ISIRT), in an effort to make users aware of the security-related issues as well as providing updates and patches, when required. However, it is often the case that end users neither make use of the information, updates or patches or even know if, whatever is provided, applies to them.

There are various reasons for this situation, but first one should understand why end users might find it hard to make use of the information made available to them.

An end user normally uses version information of assets to determine whether they include the system, software or components that are affected by a newly-founded vulnerability. However, there are some difficulties for using version information to detect the affected system, software or components.

- Ambiguity of version notation

The rules of version notation vary from vendor to vendor. This variation may cause the end user's interpretation of the version information to be different from the intention of the vendor. Not understanding the vendor-specific version system, end users may not know that a given update, patch or information applies to them.

For example, in the case that there are 3 versions, "v1", "v1.1", and "v1.2", of the system that are described as a target in the vulnerability information, if the vulnerability information is described as below.

"Systems Affected
 * XXXX v1 the affected system "

Could have the following two different meanings:

- 1) v1 is just v1.0.
- 2) v1 means v1.x, so all sub-versions of version 1.

Trying to standardize the version notation is not a practical solution; so it is necessary to read and understand the vendor-specific rules of the version notation, but it is not realistic to hope that end users will do so.

- The version of a given product cannot always be used as a criteria for determining what may actually be vulnerable.

For some vendors, the version of the products that are generally known by the users might not be able to be used to detect whether the product has vulnerabilities or not. (Figure 1)

In most component-oriented products, the presence of a vulnerability may have to be determined by the version of each component that comprises the product, not the version of the product itself.

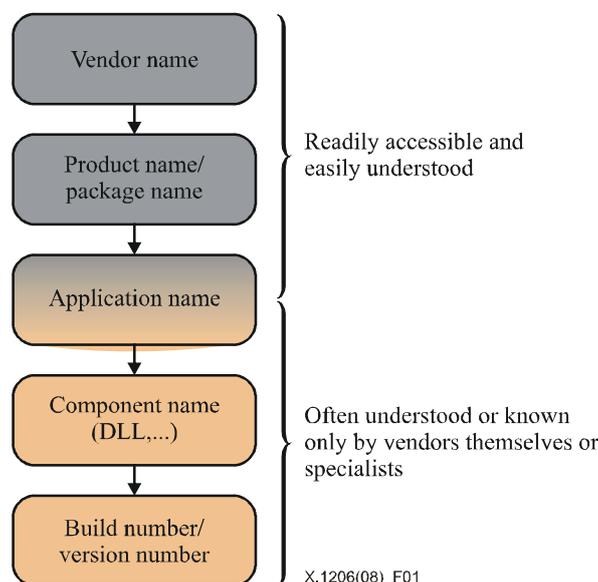


Figure 1 – Hierarchic structure of "version"

- Products that embed another vendor's products as components

Most vulnerability information usually refers to a given product. An end user may not use the specific product referred to, but instead may use the product through the use of another application that loads or otherwise makes use of a given vulnerable product's service. As a matter of fact, with open published application programming interfaces (APIs), a vendor of products used in other applications may have little to no knowledge of where their products are used or in what other applications they may be loaded or otherwise used.

- System administrators are unaware of the condition of assets under their responsibility

As system administrators are dependent on users maintaining their own systems and without personally verifying the status of each and every system themselves or relying on self reporting, it is impossible to know the status of the systems they are ultimately responsible for. Without self reporting, which is often inaccurate and incomplete for many of the reasons already mentioned regarding versioning or, custom monitoring applications, which require development and maintenance, those responsible for maintaining the security of corporate or Internet service provider (ISP) networks have few tools available.

8 Overview of vendor-neutral framework

This clause describes the overview of the vendor-neutral framework for the distribution of vulnerability, update and patch information.

By adopting the framework, vulnerability information, update and patch distribution systems, such as is shown in Figure 2, may be constructed using a simple subscription scheme. Each asset, device or local server may register with any and all available servers to receive requested (pull) or suggested (push) vulnerability information, updates and/or patches.

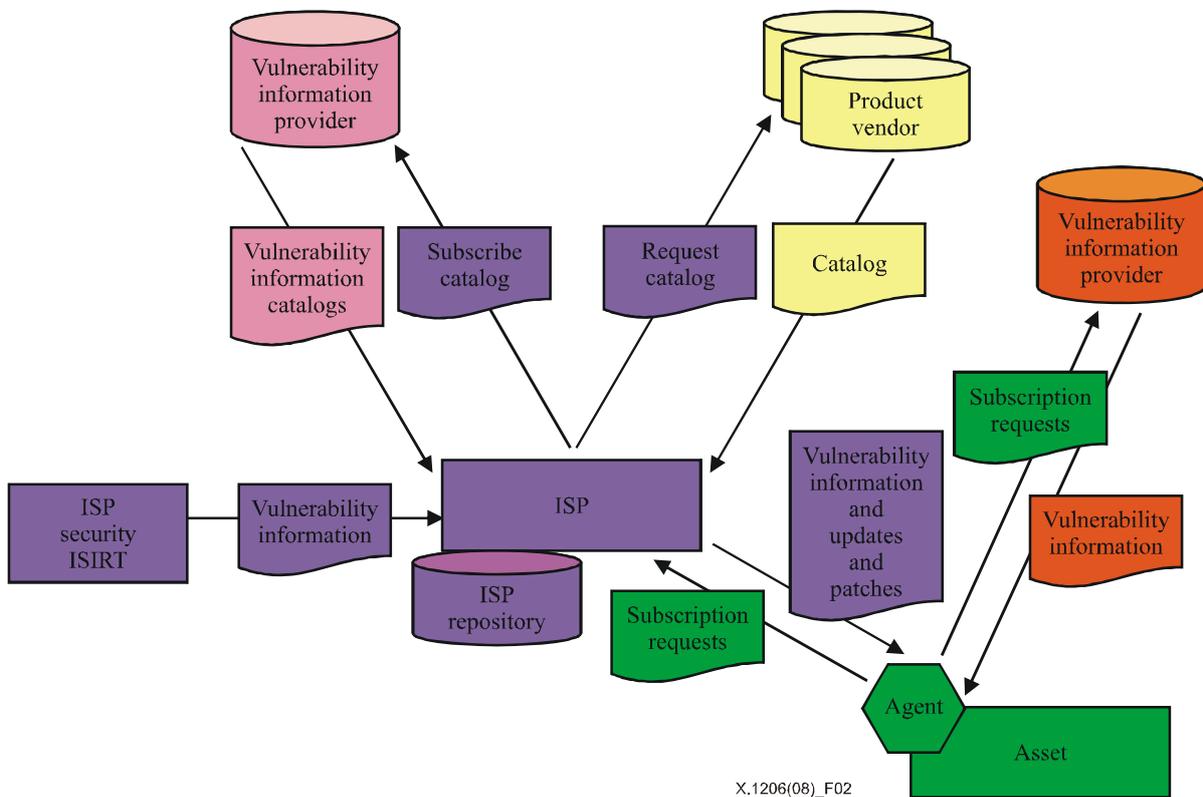


Figure 2 – Example application architecture

8.1 Multiple sources of vulnerability information, updates and patches

Using the same message scheme, any entity may make requests and/or provide vulnerability information, updates and/or patches to any other entity. In this specific example, an ISP is acting as a collection point to act as a source to all its subscribers. Additionally, individual subscribers can register with independent third-party sources for information updates, code analysis or even code updates or patches. Of course, it would be necessary for any application attempting to apply a given update or patch to accept the update or patch as authorized, but how this is accomplished, other than somehow using the supported inclusion of signatures, is beyond the scope of this Recommendation.

8.2 Example application operation

8.2.1 Process for requests and delivery of information and updates

The asset, for example a text editor, is installed and, during its installation, a previously installed agent is notified of the installation. From the information provided to the agent during the asset's installation, the agent then makes a subscription request to the user's ISP for any and all vulnerability information, updates and/or patches.

Previous or subsequent to receiving the subscription request from the user's agent, the ISP contacts various sources of vulnerability information updates and patches based on the information provided by the agent as to the source or vendor of the newly installed application.

Additionally, the user's agent has, in this example, the ability to separately query other sources of vulnerability information and make any received information available to the user for viewing. How this latest process is initiated might be through a website promoting the vulnerability information source's services and through a download link, or through providing a message as defined in this Recommendation upon which the agent could then act.

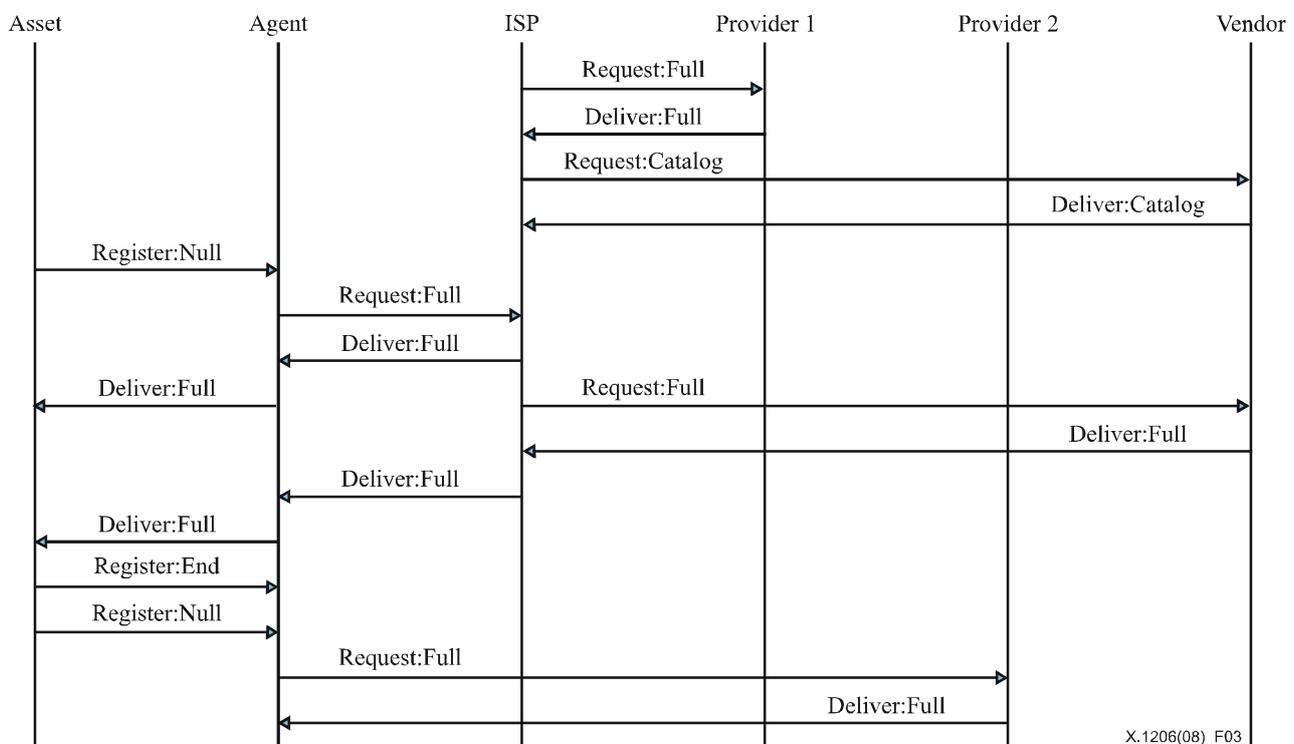


Figure 3 – Example application messages

8.2.2 Message for requests and delivery of information and updates

An ISP, which could just as easily be the administrators of a corporate or private network, makes requests from various providers of vulnerability information, updates and patches. The ISP may request either the full contents of what is available, so as to act as a local repository from the beginning of operation, or may simply request a list (Catalog) of available vulnerability information, updates and patches from which it can later request individual elements in their entirety. Although not shown here, a request for the full contents could be replied to with a "Major Message Not Supported" as well as a "Minor Message Not Supported", in which case the requesting service would try to request a list (Catalog) version instead. However, it is preferable for providers to make their request policies publicly known so that request negotiation is not necessary.

In this specific example, the ISP requests only a catalog from a vendor as the vendor's complete repository of vulnerability information, updates and/or patches might be quite extensive. But, since a catalog request also acts as a subscription for all future information, updates and patches, the ISP can be assured that anything it delivers to agents acting on behalf of its users will be the latest available. On the other hand, since the vulnerability information provider, e.g., Provider 1, likely has a lesser amount of data online, requesting the entirety of what is available may be practical.

During the installation of an asset, the installation process may pass a *Request:Full* (Major:Minor message types) message to a locally installed agent to request all current and future vulnerability information, updates and patches as they become available for the specific version of the asset or any of its contained components.

The agent then makes a *Request:Full* to the ISP on behalf of the newly installed asset.

Since the ISP, in this example, already carries vulnerability information from Provider 1 in its repository, it may deliver that instantly, or wait until a reply from the vendor is received and any and all information combined and then delivered.

The ISP having determined the vendor to be a source of information applicable to the asset, from its stored catalog, the ISP then checks its cache to see if the information is locally available or if not, makes a request from the vendor for the full information available which it will then deliver to the agent, as well as optionally caching the information in its own repository.

Through a user interface to the agent, the user may locate a useful source of vulnerability information independently and possibly using a pre-constructed request from a provider, e.g., provider 2, that the user may have downloaded, instructs the agent to request either a catalog or actual vulnerability information available or as it becomes available in the future.

After a given asset or component thereof has been upgraded, the requests for updates to the previous version are cancelled using *Register:End* and requests for applicable information updates for the new versions may be made using *Register:null*. Similarly, if an asset is removed from service, future information updates may be cancelled through the use of *Register:End* as well.

8.3 Security and privacy considerations

In the framework of this Recommendation, user information is not transmitted to the vendor side. However, it is necessary to consider authentication and integrity verification of delivered information and the following methods are suggested to be used:

- Sources should identify themselves using signatures in messages.
- Agents should verify authenticity and integrity of delivered messages.
- Updates have to be on a non-individualized basis, (non-personalized) so that no specific information of the end-user system is delivered into the network.
- Download or installation on end-user equipment requires the endorsement of the end-user.

9 Recommendation architecture

9.1 Message core layer

The message core provides platform/operating system/application/service (POAS) independent communication between compliant server and client applications built to POAS dependent requirements. The message core layer abstracts away to specific message compliant implementations the complexities involved in how to perform a given operation leaving an administrator only needing to concentrate on what operations need to be performed.

9.2 Message/application layer

The message/application layer consists of two main elements. The first element is POAS independent processes or functions normally carried out during asset or device operations, e.g., requesting vulnerability information, updates or patches. These operations are at a higher level and apply to systems and functions regardless of how a given system is implemented. For example, all systems need updating or patching periodically, or vulnerability information regarding them becomes available that users should be made aware of.

The problem is that virtually every different system performs those exact same high level functions in totally different ways at the implementation level. In the Recommendation architecture, it is at this layer, where POAS independent requirements clash with lower level implementations that the primary interface is created.

In general, an identifier is assigned to high level operations/processes that are required to be executed on systems regardless of the system type. Systems of different types then use these identifiers to communicate instructions to each other. Upon receiving a given instruction, the receiving system simply passes that instruction, along with any associated data, to a system specific handler assigned to process that given instruction.

In many ways, this is no different than processing and rendering a compressed video stream or reading a file, such as this document on a multiplicity of different platforms. Each platform dependent video player or doc file reader is responsible for processing the received data based on the standards involved.

9.3 Scalability

The scalability of systems implemented using this Recommendation is supported via the topographical server/client identification architecture which supports a client of one server being a server to other clients below. In effect, any topographical grouping of servers and clients are supported that can be described by nodes in a tree graph. Also, the ability to bundle messages for delivery according to belonging to a specific client or group allows any server at whatever level above a given client to be able to communicate and interact with that client.

- Clients and servers are identified by/assigned a 128-bit globally unique identifier (GUID)
- Clients may be pre-assigned a GUID or have one reassigned on registration with a given Server or both
- Depending on topographic policy, a client may be assigned/use different GUIDs for interacting with different Servers
- In cases where a client is itself a server for other clients (local server), the local server may "Shadow" the true GUID of the server it reports to or vice versa
- Clients may report/be assigned to more than one server with each server being responsible for providing a different range of services.

9.4 Extensibility

New platforms, applications, functions and services can be added simply by creating a schema deriving from and extending this Recommendation's message core schema.

From a defined message schema, modules can then be created for whatever platform desired with the assurance of full interoperability.

Based on a new message schema's messages and message data structures, user interfaces can easily be created on the fly as needed or desired.

Common messages can be "re-used" for new message sets either by importing the existing message definition, the message data structure or both, as needed.

9.5 Platform independence

- This Recommendation is the "Normative" interface among and between all participating entities.
- Modules implementing this Recommendation are created on a POAS dependent basis to implement messages, functions and services as required.
- Any participating device can issue any specified message or request to any other participating device independent of the platform either device is implemented on. In specific cases, contained message data may or likely will contain platform specific content, but in all cases, this Recommendation is designed to remove the need for cross platform knowledge. This Recommendation only supports one device to command another device what to do, not how to do it.

9.6 Client/Server communication

9.6.1 Public protocol

- XML based.

9.6.2 Server-client communication modes

- Server push
 - Server sends notifications and updates as they become available to client for which the client has registered for.
- Server pull
 - Server sends requests for operations, notifications and updates to client.
- Client push
 - Client sends notifications and updates to server.
- Client pull
 - Client sends requests for operations, notifications and updates to server.

10 Components of the framework

This Recommendation is defined using XML and consists of a messaging architecture supporting the addressing of sources and destinations of messages, and the carriage of messages as well as the associated message data.

This clause provides the definitions of the data format of the main components: the message container, the register message and the version message.

10.1 Message container

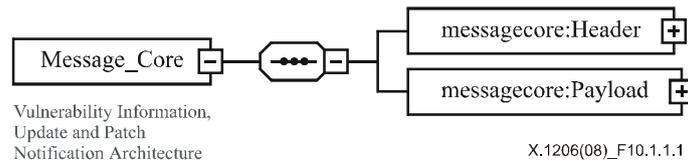
The message container acts as both an envelope for the carrying, routing and delivery of messages and replies as well as an abstract base upon which the other messages and replies may be created by extension and derivation.

10.1.1 Message_Core

All messages used in the Recommendation architecture are defined by importing, extending and deriving from *Message_Core*.

The root element of the message structure is *Message_Core*.

10.1.1.1 Syntax



10.1.1.2 Semantics

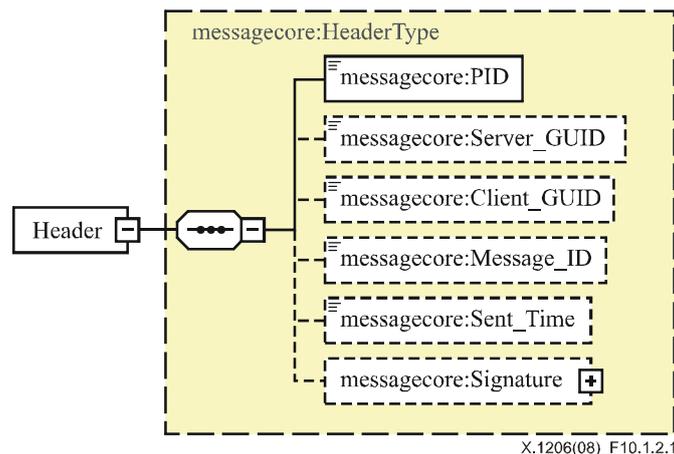
Header – Contains general routing and message identification information. [REQUIRED]

Payload – Contains an XML **Choice** of one or more bundled *Message_Core* messages, or a single message. For use in situations where messages may be cached at a given node (local server), or when a server controlling/servicing a local server may wish to receive messages and message data from clients controlled/serviced by the local server. [REQUIRED]

10.1.2 Header

The message header contains a protocol ID for extensibility and interoperability purposes, addressing information such as source(s) and destination(s) for a given message, a field to carry a message ID and a field to carry message origination time information.

10.1.2.1 Syntax



10.1.2.2 Semantics

PID – Protocol ID. A 128-bit value used to identify a specific version and/or extension to the message protocol, or a complete replacement thereof which at a minimum imports *Message_Core*, *Header*, and *PID*. To signal conformance with this Recommendation, applications may use a value of #h00000001. [REQUIRED]

Server_GUID – A 128-bit value used by clients to uniquely identify either a server or a local server to which the clients directly report. [OPTIONAL]

Client_GUID – A 128-bit value used by servers and local servers to uniquely identify clients reporting to them. [OPTIONAL]

Message_ID – A 128-Bit value used to uniquely identify a given message. Although the method to select an appropriate *Message_ID* is non-normative, it is expected that a given system peer will use the same value of *Message_ID* in subsequent replies. [OPTIONAL]

Sent_Time – Contains the time at which a given message was sent. [OPTIONAL]

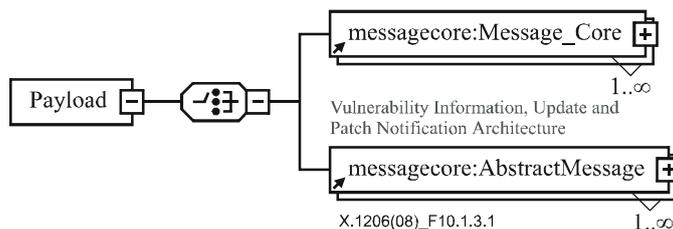
Signature – An enveloped XML signature. [OPTIONAL]

10.1.3 Payload

The functionality associated with the contents of a given *Payload* is defined by the definition and association of a *UUID* and/or *CID* with a *Message* and *Message* associated *Message_Data*. The *UUID/CID*, *Message* and *Message_Data* elements are the main points of extensibility in the Recommendation architecture and are what gives the Recommendation architecture its cross-platform independence. A defined *Message* and its associated *Message_Data* are referred to as a Message Set.

By defining and associating a *UUID* and/or *CID* with the definition of a message set XML schema, which imports and extends the *Message_Core* schema, and by the implementation of a module for a given message set which includes the *Message* and *Message_Data* schema, modules can be loaded into any Recommendation-compliant application with full interoperability assured.

10.1.3.1 Syntax



10.1.3.2 Semantics

Message_Core – Provides the ability to package and deliver multiple messages from multiple sources. If chosen [REQUIRED]

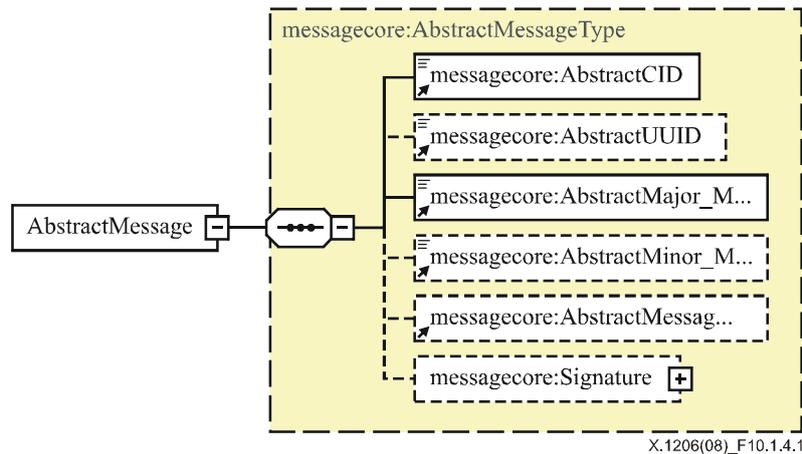
AbstractMessage – The relevant message and module identifier as well as specific message type indication and message specific data. [REQUIRED]

10.1.4 AbstractMessage

The *AbstractMessage* data structure is used to carry information to identify exactly which message and message mode is being requested, but also to carry any necessary data for the requested message to be carried out.

This data structure is intended to be extended by any specific message defined via a normative schema and, due to its being implemented in the core schema as an abstract class, cannot be used without an extending definition.

10.1.4.1 Syntax



10.1.4.2 Semantics

AbstractCID – A 128-bit value used to uniquely identify a single given functionality. A given implemented module, identified by a given *UUID* may implement support for more than one type of functionality identified by a *CID*. Conversely, more than one module may implement the same functionality but for different applications, e.g., a word processor version module and a video editor version module. [REQUIRED]

AbstractUUID – A 128-bit value used to uniquely identify an implemented module that can process *Messages* with which its ID is associated. [OPTIONAL]

AbstractMajor_Message – A string which is used to carry instructions either from servers to clients or vice versa. Such instructions are defined in schemas importing and extending the *Message_Core* schema, and may include, but are not limited to, "Register", "Request", etc. [REQUIRED]

AbstractMinor_Message – An additional message identifier string to further define the mode of the indicated message. Such instructions are defined in schemas importing and extending the *Message_Core* schema and modes may be, but are not limited to "Full", "Catalog", etc. [OPTIONAL]

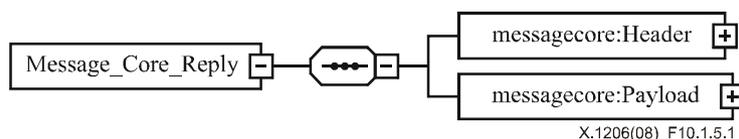
AbstractMessage_Data – Data required for the processing of the requested message. [OPTIONAL]

Signature – An enveloped XML signature. [OPTIONAL]

10.1.5 Message_Core_Reply

The *Message_Core_Reply* data structure is used to carry a status reply and any additionally desired data to whatever entity that initiated in a given communication stream.

10.1.5.1 Syntax



10.1.5.2 Semantics

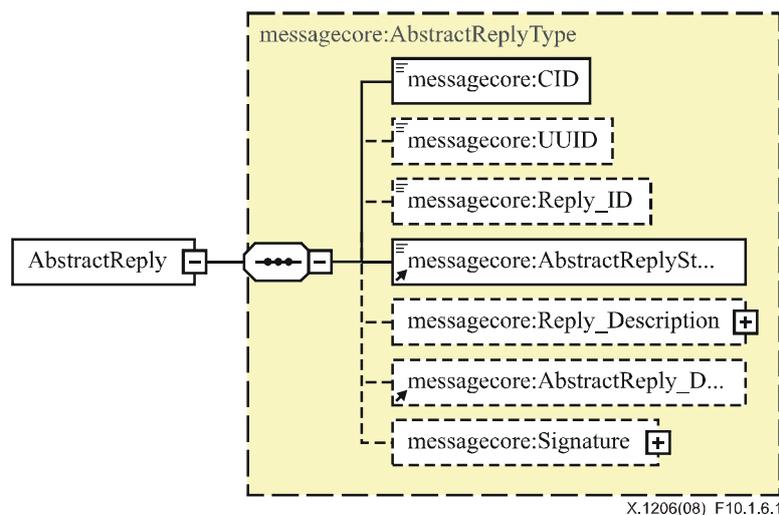
Header – Structure, data and requirements identical to *Message_Core:Header*.

Payload – Contains an XML **Choice** of one or more bundled *Message_Core_Reply* messages, or a single *Message_Core_Reply* message. For use in situations where messages may be cached at a given node (local server), or when a server controlling/servicing a local server may wish to receive messages and message data from clients controlled/serviced by the local server. [REQUIRED]

10.1.6 Abstract_Reply

The *Abstract_Reply* data structure is used to carry a status reply and any additionally desired data to whatever entity that initiated a given communication stream.

10.1.6.1 Syntax



10.1.6.2 Semantics

CID – A 128-bit value used to identify a specific version and/or extension to the message protocol or, a complete replacement thereof which at a minimum imports *Message_Core*, *Header*, and *PID*. To signal conformance with this Recommendation, applications may use a value of #h00000001. [REQUIRED]

UUID – A 128-bit value used to uniquely identify an implemented module that can process replies with which its ID is associated. [OPTIONAL]

Reply_ID – A 128-bit value used to uniquely identify a given reply. Although the method to select an appropriate *Reply_ID* is non-normative, it is expected that a given system peer will use the same value as *Message_ID* in received requests to which this is a reply. [OPTIONAL]

AbstractReplyString – Extended by messages importing the *Message_Core* schema and contains strings indicating the status of the previous request to which the reply is in response to. Such strings may be but are not limited to, "Failed", "Succeeded", "Message not supported", etc. [REQUIRED]

Reply_Description – Plain text or HTML giving further information regarding the *AbstractReplyString*. [OPTIONAL]

AbstractReply_Data – Any implementation dependant data required. [OPTIONAL]

Signature – An enveloped XML signature. [OPTIONAL]

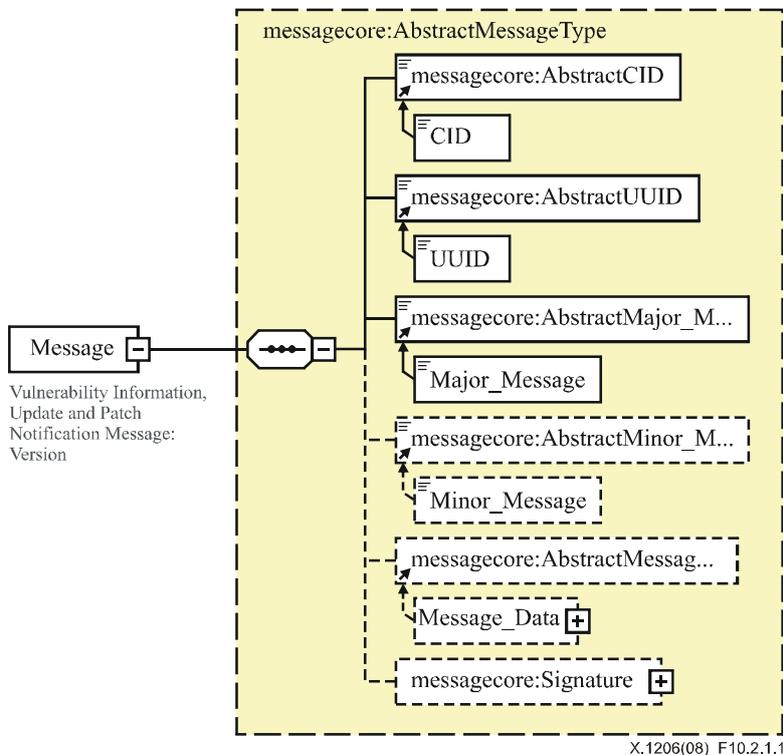
10.2 Version message

The version message is used between components of a system to request and make available vulnerability information as well as updates and/or patches.

10.2.1 Version:Message

Version:Message is designed to directly substitute for any location where *AbstractMessage* exists, and its data structure is both restricted as well as extended to uniquely support the process of make available, request and deliver vulnerability information, updates and/or patches.

10.2.1.1 Syntax



10.2.1.2 Semantics

CID – Overrides and restricts the *Message_Core:AbstractCID* base class with a specific assigned value of #h02. It uniquely identifies this message as belonging to the version message class. [REQUIRED]

UUID – Overrides the message *Message_Core:AbstractUUID* base class and is used to uniquely identify an implemented module that can process *Messages* with which its ID is associated. [REQUIRED]

Major_Message – Overrides and restricts the *Message_Core:AbstractMajor_Message* base class with an enumerated list of possible values "Register", "Request", "Deliver", "Analyse". [REQUIRED]

The values are used as follows:

- "Register" – Notify the receiving client that the asset and its version as described in the *Version:Message_Data* is in use and that vulnerability information, updates and patches should be delivered as they are made available.
- "Request" – Request available vulnerability information, updates and patches for any products minimally satisfying the requirements specified in *Version:Message_Data*.
- "Deliver" – Vulnerability information, one or more updates or patches are included in *Version:Message_Data*.
- "Analyse" – The information contained in *Version:Message_Data* is requested to be analysed. The reason for the request may be known *a priori* or an explanation may be given in *version:Info – version:Description* or *version:Info – version:Container – version:Description* depending on which is more appropriate for the application.

Minor_Message – Overrides and restricts the *Message_Core:AbstractMinor_Message* base class with an enumerated list of possible values "Full", "Catalog" and "End". [OPTIONAL]

The values are used as follows:

- "Full" – Request all available and applicable updates and/or patches.
- "Catalog" – Deliver a list of vulnerability information, updates and/or patches available for products specified in *Version:Message_Data*.
- "End" – End a subscription according to information contained in *Version:Message_Data*.

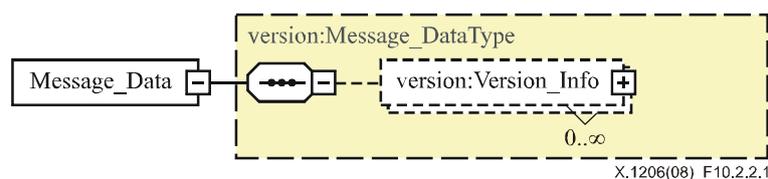
Message_Data – Overrides and restricts the *Message_Core:AbstractMessage_Data* base class with a specific data structure used to carry the identification of product version as well as carry any applicable vulnerability information, updates and/or patches. [OPTIONAL]

Signature – An enveloped XML signature. [OPTIONAL]

10.2.2 Version:Message_Data

Version:Message_Data is designed to directly substitute for any location where *AbstractMessage_Data* exist and its data structure is both restricted, as well as extended to uniquely support the process of carrying product version, vulnerability information, updates and/or patches.

10.2.2.1 Syntax



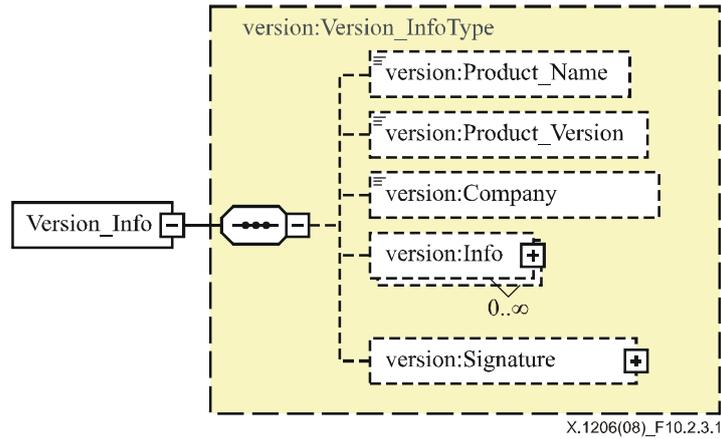
10.2.2.2 Semantics

Version_Info – Information used to identify, to the required level of specificity, products and their versions as well as supporting the carriage of vulnerability information, updates and/or patches.

10.2.3 Version_Info

Used to not only identify products, along with their vendor, version and version related information, but also to carry vulnerability information, updates and patches, as required for distribution to applications requiring them.

10.2.3.1 Syntax



10.2.3.2 Semantics

Product_Name – The name of the top level product the identified version information pertains to, e.g., for a given DVD drive which contains a specific flash system version, the *Product_Name* would refer to the DVD drive product name. [OPTIONAL]

Product_Version – The version of the top level product the identified version information pertains to. Referring to the previous example, the *Product_Version* would reply to the specific version of the DVD drive. [OPTIONAL]

Company – The vendor, provider or author of the identified product or asset. [OPTIONAL]

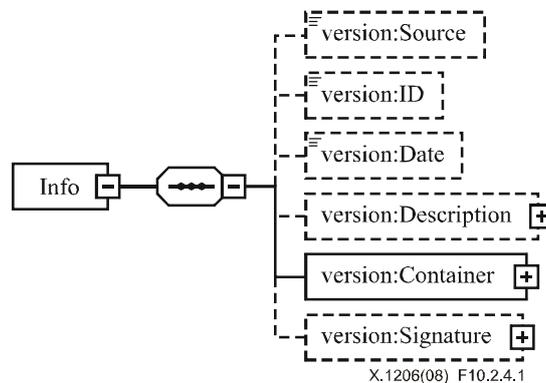
Info – A data structure used to carry vulnerability information for actual implementations of products or assets, as indicated by the previous field values. [OPTIONAL]

Signature – An enveloped XML signature. [OPTIONAL]

NOTE – The information contained in *Info* is always to be used to identify a given asset with the *Product_Name* and *Product_Version* being used only as a matter of convenience; however, where the asset identified in *Info* is a top level product, *Product_Name* and *Product_Version* as suggested to be used to clearly indicate that no "parent" for the identified asset exists.

10.2.4 Info

10.2.4.1 Syntax



10.2.4.2 Semantics

Source – A textual description, e.g., name, of the vendor or vulnerability information provider providing the included code or vulnerability information. [OPTIONAL]

ID – An ID by which the given set of *Info* data may be identified. Although the method of assignment is not specified, it is suggested that IDs assigned need only be unique within a series for a given Asset from a given *Source*. [OPTIONAL]

Date – An XML date value relevant to the *Info* data. [OPTIONAL]

Description – A textual or HTML description of either the code contained within an optional instance of *Info_Container*, or a uniform resource identifier (URI) reference thereof for the specific Asset version described in a specific instance of this data structure. [OPTIONAL]

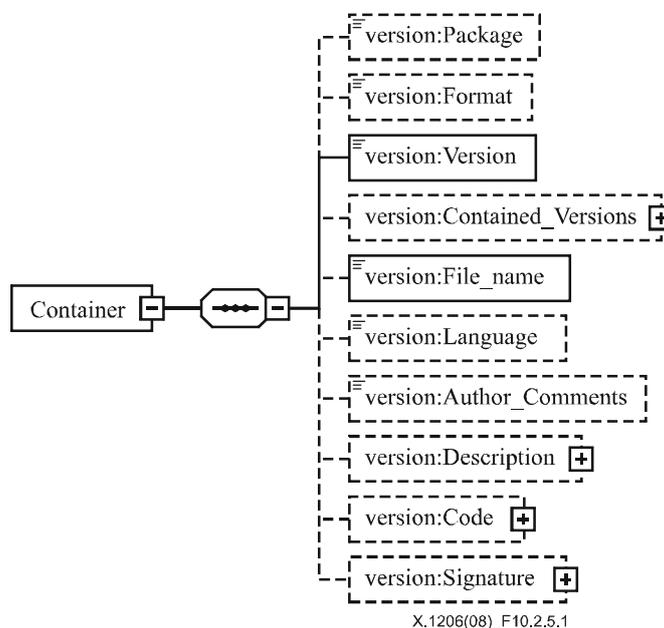
Container – Data structure for the carriage of the code in the form of an update or patch. [REQUIRED]

Signature – An enveloped XML signature. [OPTIONAL]

10.2.5 Container

A container for version identification information, as well as vulnerability information, an update or a patch.

10.2.5.1 Syntax



10.2.5.2 Semantics

Package – Method of packaging used by the specific version and/or the contents of *Code*. [OPTIONAL]

Format – Format of the specific version. [OPTIONAL]

Version – Version as applicable, either included for requests or used to identify the contents of *Version_Control* and/or *Code*. [REQUIRED]

Contained_Versions – A list, defined using the *Version_Info* (see above) data structure which contains descriptions of all components of the current asset or component. [OPTIONAL]

File_Name – File name. [OPTIONAL]

Language – For assets for which different versions are available for different countries, based on language, this field is used to identify the language. [OPTIONAL]

Author_Comments – Any textual content as desired. [OPTIONAL]

Description – A textual or HTML description of the specific version contained in *Code*, or a textual or HTML description of vulnerability information pertaining to the asset identified by *Version*

and/or *File_Name* used by or within *Version_Info* : *Product_Name* and *Version_Info* : *Product_Version*.

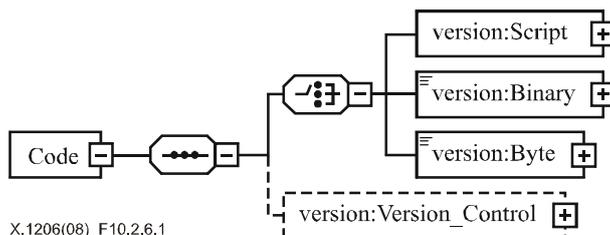
Code – A Container for executable code as well as version control information.

Signature – An enveloped XML signature. [OPTIONAL]

10.2.6 Code

A container for actual code or strings of bytes, e.g., as memory dump, or URL references thereto as well as information useful for controlling version updates.

10.2.6.1 Syntax



10.2.6.2 Semantics

Script – Code in scripted form. [OPTIONALLY REQUIRED]

Binary – Code in compiled binary form. [OPTIONALLY REQUIRED]

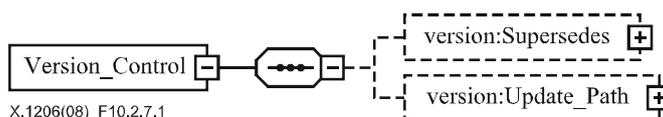
Byte – Data provided for analysis. [OPTIONALLY REQUIRED]

Version_Control – A data structure used to indicate relationships between successive versions and their possible update paths. [OPTIONAL]

10.2.7 Version_Control

A container for carrying lists of sequences of versions superseded by a given version as well as a list identifying to which versions the identified update or patch may be applied.

10.2.7.1 Syntax



10.2.7.2 Semantics

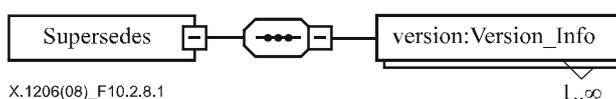
Supersedes – A sequenced list of versions superseded by the *Version_Info*. [OPTIONAL]

Update_Path – A list of versions to which the current update or patch may be applied. [OPTIONAL]

10.2.8 Supersedes

A list of assets which are superseded by the specified version.

10.2.8.1 Syntax



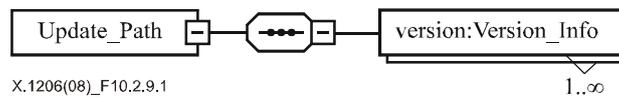
10.2.8.2 Semantics

Version_Info – (see *info*).

10.2.9 Update_Path

A list of assets to which the contained update or patch may be applied.

10.2.9.1 Syntax

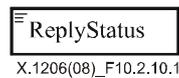


10.2.9.2 Semantics

Version_Info – (see *info*).

10.2.10 ReplyStatus

10.2.10.1 Syntax



10.2.10.2 Semantics

ReplyStatus – Contains one of the following:

"Failed" – The requested operation could not be completed.

"Succeeded" – The requested operation succeeded.

"Major message Not Supported" – The requested major message identifier or its use in the given context is not supported.

"Minor message Not Supported" – The requested minor message identifier or its use in the given context is not supported.

"Handler not available" – The message type identified is not supported.

11 Schemas

11.1 Message_Core

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:messagecore="http://www.itu.int/xml-namespace/itu-t/x.1206/CORE/"  
xmlns:ns1="http://www.w3.org/2000/09/xmldsig#" targetNamespace="http://www.itu.int/xml-namespace/itu-t/x.1206/CORE/"  
elementFormDefault="qualified" attributeFormDefault="unqualified">
```

```
<xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-schema.xsd"/>
```

```
<xs:element name="Message_Core">
```

```
<xs:annotation>
```

```
<xs:documentation>Vulnerability Information, Update and Patch Notification Architecture</xs:documentation>
```

```
</xs:annotation>
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="Header" type="messagecore:HeaderType"/>
```

```
<xs:element name="Payload">
```

```
<xs:complexType>
```

```
<xs:choice>
```

```
<xs:element ref="messagecore:Message_Core" maxOccurs="unbounded"/>
```

```
<xs:element ref="messagecore:AbstractMessage" maxOccurs="unbounded"/>
```

```
</xs:choice>
```

```

        </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name="HeaderType">
    <xs:sequence>
        <xs:element name="PID" type="xs:string"/>
        <xs:element name="Server_GUID" type="xs:string" minOccurs="0"/>
        <xs:element name="Client_GUID" type="xs:string" minOccurs="0"/>
        <xs:element name="Message_ID" type="xs:string" minOccurs="0"/>
        <xs:element name="Sent_Time" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="Signature" type="ns1:SignatureType" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="AbstractMessage" type="messagecore:AbstractMessageType" abstract="true"/>
<xs:complexType name="AbstractMessageType">
    <xs:sequence>
        <xs:element ref="messagecore:AbstractCID"/>
        <xs:element ref="messagecore:AbstractUUID" minOccurs="0"/>
        <xs:element ref="messagecore:AbstractMajor_Message"/>
        <xs:element ref="messagecore:AbstractMinor_Message" minOccurs="0"/>
        <xs:element ref="messagecore:AbstractMessage_Data" minOccurs="0"/>
        <xs:element name="Signature" type="ns1:SignatureType" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="AbstractCID" type="messagecore:AbstractCIDType" abstract="true"/>
<xs:complexType name="AbstractCIDType">
    <xs:simpleContent>
        <xs:extension base="xs:string"/>
    </xs:simpleContent>
</xs:complexType>
<xs:element name="AbstractUUID" type="messagecore:AbstractUUIDType" abstract="true"/>
<xs:complexType name="AbstractUUIDType">
    <xs:simpleContent>
        <xs:extension base="xs:string"/>
    </xs:simpleContent>
</xs:complexType>
<xs:element name="AbstractMajor_Message" type="messagecore:AbstractMajorMessageType" abstract="true"/>
<xs:complexType name="AbstractMajorMessageType">
    <xs:simpleContent>
        <xs:extension base="xs:string"/>
    </xs:simpleContent>
</xs:complexType>
<xs:element name="AbstractMinor_Message" type="messagecore:AbstractMinorMessageType" abstract="true"/>
<xs:complexType name="AbstractMinorMessageType">
    <xs:simpleContent>
        <xs:extension base="xs:string"/>
    </xs:simpleContent>
</xs:complexType>

```

```

    </xs:simpleContent>
</xs:complexType>
<xs:element name="AbstractMessage_Data" type="messagecore:AbstractMessage_DataType" abstract="true"/>
<xs:complexType name="AbstractMessage_DataType"/>
<xs:element name="Message_Core_Reply">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Header" type="messagecore:HeaderType"/>
      <xs:element name="Payload">
        <xs:complexType>
          <xs:choice>
            <xs:element ref="messagecore:Message_Core_Reply" maxOccurs="unbounded"/>
            <xs:element ref="messagecore:AbstractReply"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="AbstractReply" type="messagecore:AbstractReplyType"/>
<xs:complexType name="AbstractReplyType">
  <xs:sequence>
    <xs:element name="CID" type="messagecore:AbstractCIDType"/>
    <xs:element name="UUID" type="messagecore:AbstractUUIDType" minOccurs="0"/>
    <xs:element name="Reply_ID" type="xs:string" minOccurs="0"/>
    <xs:element ref="messagecore:AbstractReplyStatus"/>
    <xs:element name="Reply_Description" type="messagecore:Description_Type" minOccurs="0"/>
    <xs:element ref="messagecore:AbstractReply_Data" minOccurs="0"/>
    <xs:element name="Signature" type="ns1:SignatureType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="AbstractReplyStatus" type="messagecore:AbstractReplyStatusType" abstract="true"/>
<xs:complexType name="AbstractReplyStatusType">
  <xs:simpleContent>
    <xs:extension base="xs:string"/>
  </xs:simpleContent>
</xs:complexType>
<xs:element name="AbstractReply_Data" type="messagecore:AbstractReply_DataType" abstract="true"/>
<xs:complexType name="AbstractReply_DataType"/>
<xs:complexType name="Description_Type">
  <xs:attribute name="ref" type="xs:anyURI" use="optional"/><![CDATA[]]></xs:complexType>
</xs:schema>

```

11.2 Message_Version

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:messagecore="http://www.itu.int/xml-namespact/itu-t/x.1206/CORE/"
  xmlns:version="http://www.itu.int/xml-namespact/itu-t/x.1206/CORE/MESSAGE/VERSION/"
  xmlns:xmldsig="http://www.w3.org/2000/09/xmldsig#" targetNamespace="http://www.itu.int/xml-namespact/itu-t/x.1206/CORE/MESSAGE/VERSION/" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="http://www.example.com/CORE" schemaLocation="Message_Core.xsd"/>

```

```

<xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-schema.xsd"/>
<xs:element name="Message" type="messagecore:AbstractMessageType" substitutionGroup="messagecore:AbstractMessage">
  <xs:annotation>
    <xs:documentation>Vulnerability Information, Update and Patch Notification Message : Version</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="CID" type="messagecore:AbstractCIDType" substitutionGroup="messagecore:AbstractCID"/>
<xs:element name="UUID" type="messagecore:AbstractUUIDType" substitutionGroup="messagecore:AbstractUUID"/>
<xs:element name="Major_Message" type="version:Major_MessageType" substitutionGroup="messagecore:AbstractMajor_Message"/>
<xs:complexType name="Major_MessageType">
  <xs:simpleContent>
    <xs:restriction base="messagecore:AbstractMajorMessageType">
      <xs:enumeration value="Register"/>
      <xs:enumeration value="Request"/>
      <xs:enumeration value="Deliver"/>
      <xs:enumeration value="Analyse"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
<xs:element name="Minor_Message" type="version:Minor_MessageType" substitutionGroup="messagecore:AbstractMinor_Message"/>
<xs:complexType name="Minor_MessageType">
  <xs:simpleContent>
    <xs:restriction base="messagecore:AbstractMinorMessageType">
      <xs:enumeration value="Full"/>
      <xs:enumeration value="Catalog"/>
      <xs:enumeration value="End"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
<xs:element name="Message_Data" type="version:Message_DataType" substitutionGroup="messagecore:AbstractMessage_Data"/>
<xs:complexType name="Message_DataType">
  <xs:complexContent>
    <xs:extension base="messagecore:AbstractMessage_DataType">
      <xs:sequence>
        <xs:element name="Version_Info" type="version:Version_InfoType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Reply" type="messagecore:AbstractReplyType" substitutionGroup="messagecore:AbstractReply"/>
<xs:element name="ReplyStatus" substitutionGroup="messagecore:AbstractReplyStatus">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="messagecore:AbstractReplyStatusType">
        <xs:enumeration value="Failed"/>
        <xs:enumeration value="Succeeded"/>
        <xs:enumeration value="Major Message Not Supported"/>
        <xs:enumeration value="Minor Message Not Supported"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>

```

```

        <xs:enumeration value="Handler not available"/>
    </xs:restriction>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="Reply_Data" type="messagecore:AbstractReply_DataType" substitutionGroup="messagecore:AbstractReply_Data"/>
<xs:complexType name="Version_InfoType">
    <xs:complexContent>
        <xs:extension base="messagecore:AbstractReply_DataType">
            <xs:sequence>
                <xs:element name="Product_Name" type="xs:string" minOccurs="0"/>
                <xs:element name="Product_Version" type="xs:string" minOccurs="0"/>
                <xs:element name="Company" type="xs:string" minOccurs="0"/>
                <xs:element name="Info" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Source" type="xs:string" minOccurs="0"/>
                            <xs:element name="ID" type="xs:string" minOccurs="0"/>
                            <xs:element name="Date" type="xs:string" minOccurs="0"/>
                            <xs:element name="Description" minOccurs="0">
                                <xs:complexType>
                                    <xs:attribute name="ref" type="xs:anyURI" use="optional"/><![CDATA[]]></xs:complexType>
                                </xs:element>
                                <xs:element name="Container">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="Package" type="xs:string" minOccurs="0"/>
                                            <xs:element name="Format" type="xs:string" minOccurs="0"/>
                                            <xs:element name="Version" type="xs:string"/>
                                            <xs:element name="Contained_Versions" type="version:Version_InfoType" minOccurs="0"/>
                                            <xs:element name="File_name" type="xs:string" minOccurs="0"/>
                                            <xs:element name="Language" type="xs:string" minOccurs="0"/>
                                            <xs:element name="Author_Comments" type="xs:string" minOccurs="0"/>
                                            <xs:element name="Description" type="messagecore:Description_Type" minOccurs="0"/>
                                            <xs:element name="Code" minOccurs="0">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:choice>
                                                            <xs:element name="Script">
                                                                <xs:complexType>
                                                                    <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
                                                                </xs:complexType>
                                                            </xs:element>
                                                            <xs:element name="Binary">
                                                                <xs:complexType>
                                                                    <xs:simpleContent>
                                                                        <xs:extension base="xs:base64Binary">
                                                                            <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
                                                                        </xs:extension>
                                                                    </xs:simpleContent>
                                                                </xs:complexType>
                                                            </xs:choice>
                                                        </xs:sequence>
                                                    </xs:complexType>
                                                </xs:element>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

```

        </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="Byte">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:base64Binary">
                <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
</xs:choice>
<xs:element name="Version_Control" minOccurs="0">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Supersedes" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Version_info"
type="version:Version_InfoType" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Update_Path" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Version_Info"
type="version:Version_InfoType" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:sequence>
                <xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Signature" type="xmldsig:SignatureType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Signature" type="xmldsig:SignatureType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Signature" type="xmldsig:SignatureType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>

```

```
</xs:extension>  
</xs:complexContent>  
</xs:complexType>  
</xs:schema>
```

Bibliography

- [b-ITU-T X.667] Recommendation ITU-T X.667 (2004) | ISO/IEC 9834-8:2005, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components.*
- [b-IETF RFC 3023] IETF RFC 3023 (2001), *XML Media Types*.
<<http://www.ietf.org/rfc/rfc3023.txt?number=3023>>
- [b-IETF RFC 3075] IETF RFC 3075 (2001), *XML-Signature Syntax and Processing*.
<<http://www.ietf.org/rfc/rfc3075.txt?number=3075>>
- [b-XML Datatypes] W3C Datatypes:2001, *XML Schema Part 2: Datatypes*, W3C Recommendation, Copyright © [2 May 2001] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University),
<http://www.w3.org/TR/2001/RECxmlschema-2-20010502/>.
- [b-XML Signature] W3C Signature Schema:2001, *XML Signature Schema*, W3C Recommendation, Copyright © [1 March 2001] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University),
<http://www.w3.org/TR/xmlsigcore/xmlsig-core-schema.xsd>.
- [b-XML Structures] W3C XML Schema Part 1:2001, *XML Schema Part 1: Structures*, W3C Recommendation, Copyright © [2 May 2001] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University),
<http://www.w3.org/TR/2001/RECxmlschema-1-20010502/>.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems