Requirement
Security Requirements on Web Services / XML

Internal

Document information		
Document number	S.7.1.2:2007-12en	
Release date	21.12.2007	
Document status	Release	

Document responsibilities			
Task	Name	Function	Date
Author	Dr. Uwe Wilhelm	GGS Project MAS	20.12.2007
Auditor	Andreas Sensen	Products and Innovation	26.11.2007
Approved by	Dr. Thomas Breitbach	GGS	21.12.2007
Review cycle		1 year	
Author's contact e-mail:		WilhelmU@telekom.de	

Co-authors and co-signers			
Name	Function	Task	Document subject area

Keywords

GGS, MAS, Web Services Security

Change history			
Version	Date	Name	Change
1.0	20.12.2007	U.Wilhelm / A. Sensen	Final changes before release

Table of Contents

1 Introduction

1.1 Purpose

An increasing number of services offered by Deutsche Telekom AG to external third parties are realized in the form of web services. These can roughly be described by the use of a certain set of technologies: SOAP, XML, WSDL, and UDDI. This document defines the security requirements, which need to be fulfilled in order to ensure the security of web services. The requirements are intended to prescribe an acceptable level of security for web services operated by or on behalf of Deutsche Telekom AG and offered to external third parties or also to other parties within the Deutsche Telekom Group. For a particular web service, the actual threat level needs to be considered in order to assess whether some requirements that are given as optional (should-requirements) actually need to be implemented. In any case, for every optional requirement that is not implemented, a brief rationale must be provided that explains why the requirement was not implemented.

With the goal of realizing a service oriented architecture (SOA), web services are also increasingly used internally within Deutsche Telekom AG for the fast and easy creation of new service interfaces between various system components. The security requirements for these internal web services may in some respects be less stringent than those for external services (e.g., oftentimes no confidentiality is required for internal communication). Nevertheless, Deutsche Telekom AG – Global Group Security recommends, wherever appropriate, the application of the security requirements given in this document also to internal web services.

Whenever Deutsche Telekom AG uses web services that are offered by external third parties, these requirements may also be used to assess the security of the used service. If the level of security of such a service is lower than the level prescribed in this document, it may be necessary to assess whether this lack of security constitutes a threat for Deutsche Telekom AG (e.g., can an external attacker impersonate Deutsche Telekom AG and thus use services on our account). If this is the case, then Deutsche Telekom AG must require the external third party to adhere to the relevant security requirements given in this document.

In addition to the security requirements given in this document, the other relevant security requirement documents from Global Group Security (i.e., [WebApps], [WebServer], and [SecGuide]) should be consulted whenever appropriate.

1.2 Scope

This document applies to all web services that are developed and operated by or on behalf of Deutsche Telekom AG and its subsidiaries.

2 Architecture of Web Services

[Req 2.1] Secure Architecture

This document does not make detailed prescriptions with respect to the overall architecture, deployment, and/or employed technology of a web service. Nevertheless, with respect to security, the architecture of a web service MUST follow an N-tier architecture (see Fig. 1), where the access and validation tier, which decouples the client from the actual web service application, is deployed in a DMZ and all subsequent tiers (e.g., the application and/or data tier) are deployed in one or several MZs. The DMZ MUST be separated from the MZ with a firewall; additional firewalls MAY further separate the application from the data tier.

The following functions SHOULD be realized in the access and validation tier (i.e., in the DMZ):

- some coarse grained authentication and authorization for the request (i.e., is the apparent sender of the request potentially allowed to access the web service in question);
- some basic validation of the request;
- other pre- and post-processing tasks (such as terminating an SSL/TLS encryption or security relevant logging);
- termination of the tcp traffic and thus preventing all direct access to web service end points in the MZ.

The client tier is not trusted to manage or protect security relevant data (other than a session identifier, see [Req 5.1]). Therefore, all security relevant data MUST be stored and managed by the web service in an MZ (no such data sent by the client can be relied upon).

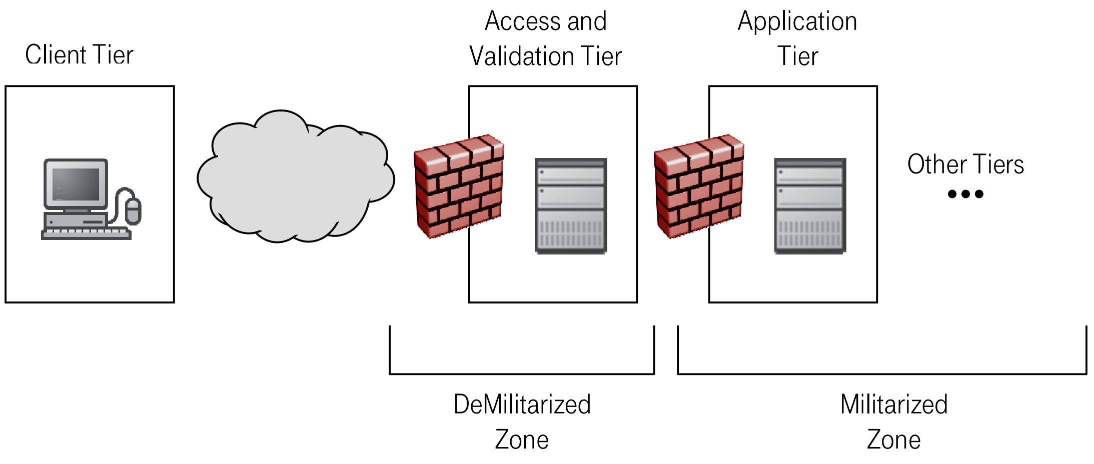

Fig. 1 Typical N-Tier Architecture

[Req 2.2] Security Appliance

In order to implement the required security functionality in the access and validation tier, it is RECOMMENDED to funnel all access to web service end points through a specialized security appliance (in the form of a web (services) proxy), which facilitates the following tasks:

- centralized enforcement of a unified security,
- certificate management (see [Req 4.3]),
- centralized validation of requests (see [Req 7.1] and [Req 7.4]),
- blocking of potentially insecure XML and Web Technologies (see [Req 6.2]),
- logging (see [Req 3.7]),
- security mechanisms on the transport layer (e.g., SSL/TLS, VPN, and/or NAT).

It is RECOMMENDED to use the system architecture that is outlined in [SecProxy], which provides an appropriate solution for many of the requirements outlined in this document.

Motivation: *Specialized security appliances are more robust and efficient than the actual web service end points in particular in the context of malicious attacks. Also, since the security appliance is mostly transparent to the web service implementation, it is more easily upgradeable in the case of new security threats.*

[Req 2.3] Trust Relationships

In most cases, a web service request only has a sender and a receiver. In these simple cases there MUST be a direct trust relationship between the receiver and the sender, where the receiver verifies that the web service request was actually sent by the sender and the sender takes appropriate measures to ensure that the web service request will be received by the proper receiver (see [Req 4.1] and [Req 4.2]). In more complex cases where there is one (or several) intermediate node(s) between the sender and the receiver, we require this intermediate node to either belong to the trust domain of the sender or the trust domain of the receiver. In the former case the receiver MUST trust the intermediate node as it would trust the sender; in the latter case, the sender MUST trust the intermediate node as it would trust the receiver.

Motivation: *Complex web service scenarios with multiple intermediate nodes lead to an increase in security problems, e.g. complex forwarding rules might be exploited by an attacker.*

3 General Guidelines

This chapter documents general guidelines with regard to the realization of secure web services.

[Req 3.1] Fail Safe

If the web service detects some sort of misbehaviour or incomprehensible behaviour it MUST fail safe; that is the web service denies the service or drops the request. If the web service was addressed in the context of a session, the corresponding session MUST be invalidated. Any such event MUST be logged (according to [Req 3.7]).

Motivation: *In a machine-to-machine interaction we can expect the communication partner to behave correctly according to a given specification. Otherwise security critical side-effects may occur where an attacker gains knowledge on a particular web service through experimentation and observing the results when sending unexpected requests. If the web service simply drops every request that it can not correctly handle this exploration becomes much more difficult for an attacker.*

[Req 3.2] Confidentiality Protection

If a web service request is routed over public networks, the content of the request MUST be protected from potential eavesdroppers on the unprotected communication path. This can be accomplished through the use of SSL/TLS on the transport layer, XML encryption on the application layer, or other appropriate mechanisms (e.g., VPN). On protected backend networks within an MZ, web service requests SHOULD NOT be confidentiality protected in order to allow for intrusion detection or intrusion prevention.

If XML encryption is used for confidentiality protection, the initial vectors (IV) of the utilised cipher algorithm SHOULD be random and SHOULD NOT be used repeatedly. Furthermore digests or signatures of encrypted data should be encrypted too. XML encryption MUST NOT be used for integrity protection (see [Req 3.4]) since the IV are stored in the message in plain text and could be manipulated by an attacker.[1]

Motivation: *The payload of a web service request is very likely to contain sensitive data, which must not be accessible to any entities other than the communication partners. The use of SSL/TLS may be preferable over XML encryption since this approach also protects the http header information, which may also contain sensitive data*

[Req 3.3] Confidentiality Protection of Highly Sensitive Data

If a web service processes highly sensitive data (e.g., passwords or credit card data), then these data items SHOULD be individually protected using end-to-end application level mechanisms such as XML encryption and XML signature.

Motivation: *Due to special agreements within the industry, particular data items require confidentiality protection even when communicated over secure networks. This can only be ensured by using end-to-end mechanisms that decrypt data using secure keys only when needed. The application level encryption should not hamper IDS or IPS.*

[1] Refer to http://www.w3.org/TR/xmlenc-core/#sec-Security for additional guidelines on the use of XML encryption.

[Req 3.4] Integrity Protection

The content of a web service request MUST be protected from manipulation while on the unprotected communication path. This can be accomplished through the use of XML signature (XMLdsig) on the application layer, SSL/TLS on the transport layer, or other appropriate mechanisms (e.g., VPN). Global Group Security strongly recommends to use XML signature on the entire XML message / SOAP Body. Only the signed elements of the XML message MUST be trusted by the web service.

If XML signature is used, the public/private key pair and the corresponding certificate need to be different from the public/private key pair used for SSL/TLS. According to the XML signature standard XML files have to be in unicode normalized state before the signing process and should remain in unicode normalized state after the signing process. Therefore signed documents, that are not in unicode normalized form MUST be rejected by the web service. The signature of encrypted data has to be encrypted itself according to the requirements described in the Basic Security Profile (see [Req 6.1]).

Motivation: *Non-repudiation mechanisms are assumed to become more and more accepted in legal proceedings as well as for accounting and auditing purposes. Therefore, it is highly desirable to not rely on SSL/TLS as ephemeral integrity protection mechanism on the transport layer, but rather to solve this issue more appropriately on the application layer with durable XML signatures. Due to the different time scales and goals pursued with XML signatures, as compared to SSL/TLS integrity protection, the use of a special purpose public/private key pair for signatures becomes inevitable. XML signatures are the only mechanism that allows to prove to an external third party (e.g., a legal entity such as a court) the fact that a message was received from a particular entity. The timestamp and/or message identifier can be used to counter replay attacks (see [Req 3.6]). The requirement to sign the entire XML message / SOAP Body is important to counter some sophisticated attacks on the XML signature.*

[Req 3.5] Secure Message Buffer

If a SOAP request or reply is not handled immediately and needs to be buffered in the DMZ the confidentiality and integrity of the buffered message MUST be ensured.

Motivation: *Otherwise it might be possible for an attacker to access or modify these buffered messages.*

[Req 3.6] Prevention of Replay Attacks

If the web service application is potentially vulnerable to replay attacks, appropriate prevention mechanisms MUST be implemented either in the application or in the access and validation tier. In the access and validation tier this may be done with a list of stored hashes of processed messages or a unique message identifier as well as an appropriate usage of timestamps. Data items used for this purpose MUST be properly integrity protected (see [Req 3.4]).

Motivation: *There are many ways to counter replay attacks. This document does not intend to prescribe a particular mechanism to overcome this type of attacks, but leaves this with the application developers.*

[Req 3.7] Logging of Security Related Events

All security related events (e.g., failed authentication, validation failures, locking of accounts, etc.) MUST be logged. Security related events above a particular threshold SHOULD be logged in the MZ and SHOULD be reported to an operator (the thresholds for these actions may be different). All logging SHOULD be done in a secure manner. It SHOULD be possible to uncover the IP address of a potential attacker based on the logged data, which may have to be correlated with log data from other systems or layers.

Motivation: *Logs are the only means to detect and comprehend attacks.*

[Req 3.8] Monitor and Analyze Security Related Events

Procedures MUST be established to adequately monitor the logged security related events on a regular basis. The actual analysis will always require human intelligence and must therefore be done manually. However, due to the sheer amount of security events, a completely manual analysis will not be feasible. Tools for automatic pre-processing (e.g., based on misuse and anomaly detection) have to be considered. A risk assessment may be helpful to determine the effort that needs to be devoted to this task.

Motivation: *Monitoring and analysis of security events are the only means to discover an attack at an early stage. If an attacker has enough time to compromise a system, he may even remove the traces of his attack, which may then become very difficult to uncover.*

[Req 3.9] No Disclosure Of Unnecessary Information

All output created by the web service MUST exclusively consist of information required by the client, in order to prevent accidental disclosure of information that may be used by an attacker. This includes error messages, version information on used software, etc.

Motivation: *Such information may help an attacker to determine potential security vulnerabilities or implementation details.*

[Req 3.10] Estimate and minimize the risks of Denial of Service

All applications publicly accessible from the Internet are potentially susceptible to denial of service (DoS) attacks. Web services in particular are likely to be vulnerable to XML-based DoS attacks due to the high complexity of processing XML documents. Therefore, for each web service a risk assessment SHOULD be performed identifying the potential damage due to a successful DoS attack. The measures taken to counter DoS attacks MUST be chosen according to the risk and potential damage.

Motivation: *Due to the high cost of effective DoS counter measures, we request each web service deployment to consider this issue individually. This document does not intend to prescribe a particular mechanism to overcome this type of attacks, but leaves this with the application developers.*

4 Authentication and Authorization

[Req 4.1] Strong Sender Authentication

The sender of a web service request MUST be properly authenticated. The mechanism used SHOULD rely on strong cryptographic algorithms (e.g., XML signature or SSL/TLS client certificates); the use of a simple username/password authentication is strongly discouraged[2]. The result of the authentication is an identity (e.g., a distinguished name) that can be used for subsequent authorization. If SSL/TLS client certificates are to be used for authentication, it MUST be ensured that the identifying information obtained by SSL/TLS from the client certificate is securely transferred to the application layer. If anonymous access to the web service is inevitable, only read access SHOULD be granted for these requests. Furthermore, there SHOULD be only one way for authentication of a client, e.g. there should be no authentication possible over SOAP and HTML at the same time.

Other means for identifying the origin of a web service request (e.g., originating IP address) may be used to further strengthen the authentication, but do not replace a strong cryptographic authentication mechanism.

Motivation: *The use of a shared secret (the password in username/password authentication) is considered too weak for the realization of business grade services. Even though username/password authentication with SSL/TLS can be made secure, the potential for operational weaknesses is considered a high risk (weak passwords, password theft, complex password management, etc.). The use of cryptographic algorithms requires the use of an appropriate PKI, which may be managed locally.*

[Req 4.2] Recipient Authentication

The recipient of a web service request MUST be authenticated by the sender of this request. This authentication can be done implicitly by using XML encryption on the request with the appropriate public key from the certificate of the intended recipient, explicitly in the communication setup of a SSL/TLS connection (also tied to a certificate), or another appropriate mechanism (e.g., VPN).

Motivation: *In order to prevent man-in-the-middle attacks the sender of a request must ensure that it communicates with the intended recipient of its request.*

[Req 4.3] Proper Certificate Handling

All certificates that are processed during confidentiality and integrity protection as well as authentication MUST be validated. This validation includes verification of the signature against a set of trusted root certificates, the validity period (i.e., “Not Before” and “Not After” fields in the certificate) based on trustworthy time data and the current revocation status of this certificate.

Motivation: *If an attacker can trick the web service to accept invalid certificates he may be able to impersonate another system or user.*

[Req 4.4] Protect Security Relevant Data

All security relevant data that is required for the proper operation of a web service MUST be adequately protected. We can distinguish between disclosure sensitive security relevant data which MUST NOT be accessible (i.e., neither readable nor writeable) to an attacker (e.g., private keys, shared keys, or password data) and manipulation sensitive security relevant

[2] If username/password authentication is used, the communication MUST be encrypted, the recipient of the request MUST be authenticated before sending the authentication credentials, the password MUST be cryptographically strong (i.e., on the order of 160 bits of random data), and additional methods for sender authentication (such as source IP verification of the original request) SHOULD be used.

data which MUST NOT be altered by an attacker (e.g., digital certificates, authorization data, web service meta data, or time data).

The primary (persistent) storage location for all security relevant data MUST be the MZ and it MUST NOT be possible to modify security relevant data from the DMZ. In general all processing of security relevant data SHOULD exclusively be done in the MZ (e.g., authorization decisions can be processed in the MZ and the result is then enforced in the DMZ). Whenever processing in the MZ is not possible due to performance or functional issues the security relevant data that needs to be processed and stored in the DMZ MUST be adequately protected with appropriate mechanisms (e.g., the data SHOULD only be held in the main memory of the executing system, preferably encrypted).

- For disclosure sensitive security relevant data this includes that such data MUST NOT be accessible from the DMZ via an API; rather this data must actively be pushed from the managing system in the MZ to the system in the DMZ that requires this data (e.g., upon starting the system in the DMZ). The trigger for pushing this data MUST NOT initiate in the DMZ. The communication path for transmitting this data as well as its storage in the DMZ MUST be adequately protected (e.g., the data should not be visible as command line parameter upon executing a 'ps').
- For manipulation sensitive security relevant data this includes that all such data in the DMZ MUST always be considered as cached data that MUST be refreshed with the primary data located in the MZ whenever appropriate.

If the system in the DMZ that requires security relevant data contains specific high protection mechanisms for managing the security relevant data (e.g., trusted hardware module), then this MUST be examined by Global Group Security to assess whether these protection mechanisms are adequate. Additionally, a secure backup of the security relevant data MUST be held in the MZ.

Motivation: *A successful attack on a server within the DMZ must not result in a corruption, disclosure or loss of the highly sensitive security relevant data. In the wors case, a manipulation of security relevant data in the DMZ can be identified using the data from the primary storage in the DMZ.*

[Req 4.5] Authentication Data in the MZ

Motivation: The local data that is used for authentication purposes (X.509 certificates or username/password data) is security relevant data and MUST be adequately protected as explained in [Req 4.4]. This functionality can be combined with the authorization data in a so-called IAM (Identity and Access Management) module located in the MZ.

Motivation: *If an attacker can access or modify authentication data he can use the regular functions of the attacked system to compromise it.*

[Req 4.6] Communication of Authenticated Identity

If some of the authentication, authorization or logging responsibilities are implemented in the application tier (i.e., not everything is done in the access and validation tier/DMZ) the identity that is authenticated as sender of a web service request in the DMZ MUST be securely communicated to subsequent entities in the communication path. This can be done in any manner appropriate to the application (http/smtp header enrichment, in form of a SOAP Header/Body element, or other). In particular it MUST be ensured that an attacker can not inject data into a request that may overwrite or prepopulate the identity information obtained in the access and validation tier/DMZ.

Motivation: *In order to perform the more fine-grained access decision mentioned in [Req 4.9], the authenticated identity must be communicated in the forwarded web service request.*

[Req 4.7] Forwarding of Signature Data

If a web service request is protected with XML signatures, the signature data MUST NOT be removed from the request by an intermediate processor.

Motivation: *Since the signature data can be used to prove reception of a message from a particular entity, it may be necessary to log such data in late processing stages of the web service request. A particular strength of XML is the possibility to simply ignore XML elements that are not needed by a processing entity. Leaving the signature in the request incurs only the acceptable disadvantage of having a slightly larger message to process and communicate.*

[Req 4.8] Authorization Data in the MZ

The authorization data of a web service is manipulation sensitive security relevant data that MUST be adequately protected as explained in [Req 4.4]. This functionality can be combined with the authentication data in a so-called IAM (Identity and Access Management) module located in the MZ.

Motivation: *If an attacker can modify authorization data he can use the regular functions of the attacked system to compromise it.*

[Req 4.9] Policy based Authorization

The decision to allow the access of an entity, identified in the authentication, to a particular web service MUST be based on a well-defined policy. The access decision MUST exclusively be based upon the identifier obtained in the authentication, the web service request in question, and the state of the web service application. We recommend the use of role based authorization policies where each identifier is associated with a role that encodes the access rights.

Motivation: *Access control can and should be performed at various points within the architecture. In the access and validation tier within the DMZ, at least a coarse grained authentication and authorization check should be performed, determining if the authenticated entity has access to any of the functionality offered by the addressed web service. More fine-grained access decisions based on the authenticated identity (see [Req 4.6]) can then be made closer to the application.*

[Req 4.10] Key Data in the MZ

In order to sign outgoing data (in a web service reply) with the XML signature mechanism or to set up a server-side SSL/TLS connection, a highly sensitive private key is required. This key data is disclosure sensitive security relevant data that MUST be adequately protected as explained in [Req 4.4].

Motivation: *If an attacker can access private or shared key data, he can use the regular functions of the attacked system to compromise it.*

5 Session Management

Most web services will not rely on an explicit session management based on session identifiers, but perform authorization checks based on the verification of XML signatures and proper authorization policies. Nevertheless, the porting of legacy applications may require the use of session identifiers.

[Req 5.1] Protect Session Identifiers

If the web service relies on session identifiers, these MUST be confidentiality protected while on the unprotected communication path. The session state MUST NOT be administrated on the client side (see [Req 2.1]) .

Motivation: *Some web service frameworks allow the management of sessions by http mechanisms (e.g., cookies). This data is sent outside the XML message and can only be protected by transport level mechanisms (e.g., SSL/TLS).*

[Req 5.2] Cryptographically Strong Session Identifiers

If the web service relies on session identifiers these MUST be created with cryptographically strong algorithms. The requirements on a session identifier are:

- o The length of the session identifiers MUST be at least 160 bit, which means > 49 digits [0...9] or 26 characters [A..Z, a..z, 0..9].
- o Session identifiers MUST be generated randomly each time a session starts.
- o Session identifiers MUST be collision free.
- o Session identifiers MUST not contain plain text user related information like name, MSISDN, credit card numbers, etc.

Motivation: *Otherwise it may be possible to determine or guess valid session identifiers.*

[Req 5.3] Session State Management Outside of Access and Validation Tier

Session state (i.e., session state and other session related information) is disclosure sensitive security relevant data that MUST be adequately protected as explained in [Req 4.4]. This functionality can be implemented in a so-called IAM (Identity and Access Management) module (see [Req 4.5] and [Req 4.6]).

Motivation: *Session state information maintained in the access and validation tier cannot be trusted.*

[Req 5.4] Session Identifier Verification

The session identifier MUST be verified upon each web service request. This MUST include that the web service application verifies if the session identifier is valid and if the identity associated with the session identifier is authorized to execute this specific function.

Motivation: *Otherwise*

- o *unauthenticated or unauthorized actions may be carried out,*
- o *the business logic may be circumvented, or*
- o *replay attacks may be possible.*

[Req 5.5] Unique Session Identifier

Each session identifier MUST be unique within the session state management at any single given point in time.

Motivation: *Otherwise this could lead to a privilege escalation.*

[Req 5.6] Fresh Session Identifier After Authentication

After a client has been authenticated, the unauthenticated session, if such a session exists, MUST be terminated and a new, fresh session identifier, completely unrelated to the previous unauthenticated session identifier must be created and sent to the client.

Motivation: *This prevents potential session fixation attacks.*

[Req 5.7] Invalidate Session State

When a session is terminated the session state MUST be actively invalidated at the server.

Motivation: *If the server holds stale session state this may be abused by an attacker for session hijacking.*

[Req 5.8] Session Timeout

If the web service relies on session identifiers, the web service application MUST invalidate the session after a certain time of inactivity. This time should be chosen in accordance to the usage of the web application. A typical time is 30 min and the timeout should be not longer than 60 min. If a longer timeout is required the reasons MUST be documented.

Motivation: *If a web service client does not explicitly terminate the session it must be terminated after an appropriate time.*

6 SOAP and XML

This chapter describes security requirements regarding XML documents in general and SOAP messages in particular. There are some security requirements not explicitly mentioned here, e.g. requirements regarding proper signature or SOAP header element handling, which are summarized in [Req 6.1].

[Req 6.1] Conformance to Basic Profile and Basic Security Profile

All web services requests MUST stay conform to the "Basic Profile"[3] and the "Basic Security Profile"[4] specifications of the Web Services-Interoperability Organization (WS-I). A web service MUST NOT process requests that are not compliant to these specifications.

Motivation: *The original web service standards left space for improvements regarding compatibility and security of web services. A common set of improvements is specified in the "Basic Profile" and "Basic Security Profile."*

[Req 6.2] Secure Use of XML and Web Technologies

A Web Service Request MUST NOT make use of the following technologies, unless the web service is explicitly capable of handling such data in a secure way: Xpointer, Xpath, Xquery, Stylesheets, DTDs and CDATA elements (see also [Req 2.2]).

Motivation: *The listed technologies may lead to security risks when handled without care. Document Type Definitions (DTDs), for example, are an outdated technologie and should be replaced by XML schema definitions. XPath, XPointer and XQuery may lead to injection attacks, similar to SQL-Injection attacks, if input from insecure sources is used to build the respective expressions. The following XPath expression, for example, could be manipulated by an attacker by providing the malicious input "' or ''='" as the password and thus:*

```
//user[name='Joe' and passwd='unbreakable']
```

leads to

```
//user[name='Joe' and passwd='' or ''='']
```

CDATA elements can be used by an attacker to obfuscate malicious content which is accepted by the XML parser. The following XML element, for example:

```
<TAG1>

  <![CDATA[<]]>SCRIPT<![CDATA[>]]>

    alert('Cross-site Scripting Attack!')

  <![CDATA[<]]>/SCRIPT<![CDATA[>]]>

</TAG1>
```

would be modified by the parser and lead to the following XML element:

```
<TAG1>
  <SCRIPT>
    alert('Cross-site Scripting Attack!')
  </SCRIPT>
```

[3] http://www.ws-i.org/Profiles/BasicProfile-1.0.html
[4] http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

```
</TAG1>
```

which itself could lead to an JavaScript attack on the application level.

[Req 6.3] Parser and Resource Restrictions

A SAX parser SHOULD be used rather than a DOM parser to process XML documents. Resource restrictions MUST be specified for the parser and the evaluation of messages MUST be stopped if the specified restrictions are exceeded.

Motivation: *DOM parsers process the complete XML message and create a tree structure in main memory out of it. SAX parsers process XML messages sequentially as token streams and are therefore normally more resource-saving and hence more resistent to DoS attacks based on complex XML structures.*

[Req 6.4] Canonicalization Methods (XMLC14N and EXCC14N respectively)

A web service MUST only use one of the canonicalization methods XMLC14N[5] or EXCC14N[6]. XMLC14N SHOULD be used if the signed XML elements are not moved into another XML context or document. EXCC14N SHOULD be used if the signed XML elements have to be moved into another XML context or document. The parameter "InclusiveNamespacePrefixList" MUST be used to identify security critical namespaces, which have to be included in the canonicalization. All incoming requests MUST be normalized and canonicalized before performing any validation on the request and before passing the request on to later processing stages.

Refer to http://www.w3.org/TR/xml-exc-c14n and http://www.w3.org/TR/xmldsig-core/ for further details on canonicalization problems.

Motivation: *A canonicalization method from an insecure source can transform an XML document or SOAP message in a malicious way. Even without malicious intent, a self-developed canonicalization method can easily contain mistakes, which may destroy signatures in the XML document. An XML element can be specified differently in different namespaces. Therefore a signature should ideally span over all namespaces relevant for the respective XML element. The standard canonicalization method XMLC14N spans over the relevant namespaces and thus prohibits moving the signed element into another XML context without invalidating the signature.To counter this disadvantage the W3C specified the "Exclusive XML Canonicalization EXCC14N" which regards only the minimal necessary namespace information.*

[Req 6.5] Avoid Reencryption of Encrypted XML Data

An XML document SHOULD NOT be reencrypted or moved into another context if the underlying structure of the encrypted data is unknown. If the encrypted data uses IDREFs or makes implicit assumptions about namespaces of the outer context, it SHOULD NOT be reencrypted or moved into another context, too.

Motivation: *The meaning of an XML message depends on the context of the namespaces in which the XML document is interpreted. Changes in a surrounding XML context can influence the meaning of the decrypted data. If for example the encrypted data contains references to surrounding, unecnrypted elements of the XML message (IDREFs / "Same-Document References"), which are redefined in the unencrypted part of the message, the meaning of the encrypted data could be changed. There can also be a major difference between*

```
<sample> <EncryptedData>…</EncryptedData> </sample>
```

and

[5] W3C: *Canonical XML Version 1.0.* (http://www.w3.org/TR/xml-c14n)
[6] W3C: *Exclusive XML Canonicalization Version 1.0.* (http://www.w3.org/TR/xml-exc-c14n/)

```
<sample xmlns="…some namespace …"> <EncryptedData>…</EncryptedData> </sample>
```

regarding the meaning of the XML message.

[Req 6.6] Prefix the Encryption Key to the Encrypted Data

Encrypted XML elements can only be decrypted if the necessary key is available. The encryption key SHOULD therefore prefix the encrypted XML elements to support stream processing of the message, e.g. using a SAX parser.

Motivation: *The implementation of this requirement supports an efficient validation according to a WS SecurityPolicy specification, thus making DoS attacks more difficult.*

[Req 6.7] HTTP »SOAPAction« Header Element Security

The `SOAPAction` HTTP header element MUST NOT be used by the web service for security relevant decisions. If a `SOAPAction` element must be specified for compatibility reasons it SHOULD be specified as empty string.

Motivation: *Confidentiality and integrity protection measures like XML encryption and XML signature only protect SOAP requests, i.e. the HTTP body. HTTP header elements are thus not protected and might be manipulated by an attacker.*

7 WSDL and UDDI

The following requirements are particularly important since they go beyond what is currently expected from web service security. However, Deutsche Telekom AG – Global Group Security believes that these will be the differentiating factors that will ensure that web services operated by or on behalf of Deutsche Telekom AG will prove resilient to at least the first wave of attacks expected when web services will come in the focus of capable attackers. Therefore, we urge our partners to strive for the realization of these requirements.

[Req 7.1] Web Service Metadata in the MZ

The WSDL and XML Schema files that are used for the WSDL and Schema validation (see [Req 7.4]) are manipulation sensitive security relevant data that MUST be adequately protected as explained in [Req 4.4]. To further prevent misuse of the web service, the WSDL and XML Schema files SHOULD only be accessible to actual users of the web service. If some part of the web service API should be published (e.g. in a UDDI), this MAY be limited to a subset of APIs for non-business-critical transactions.

Motivation: *If an attacker can modify these metadata, he may be able to launch sophisticated attacks on the web service in question (e.g., overcome schema validation).*

[Req 7.2] Secure UDDI

If a UDDI is offered to external third parties to facilitate automatic web service management this service has to be operated securely. This means that at least the following issues MUST be considered:

- only authorized individuals can publish or change information in the registry;
- changes or deletions can only be made by the originator of the information;
- each instance of a UDDI registry can define its own user authentication mechanism;
- if the UDDI is a web service all the requirements from this document have to be considered.

Motivation: *If an attacker can modify the data in the UDDI he can easily deviate clients to non-Deutsche-Telekom services.*

[Req 7.3] Detailed WSDL Specification

The WSDL that defines the web service operated by Deutsche Telekom MUST specify the element and attribute values of input parameters as detailed as possible in order to allow request validation. In particular, each element and attribute value communicated in a web service request SHOULD be described with the expected data type (e.g., integer, string, etc.), the expected length and/or range (e.g. min value, max value, min length, max length – avoid "minOccur=1" and "maxOccur=unbounded"), and whenever appropriate a formal specification describing the acceptable data. An example for such a formal specification are regular expressions, e.g.

```
“([A-Za-z0-9]+[-._+&#%/=~])*[A-Za-z0-9]+@([-A-Za-z0-9]+[.])+[A-Za-z]{2,6}”
```

to specify a valid e-mail address. The base type "int" SHOULD be replaced by an application specific subtype which exactly defines the needed integer range. Equivalently the base type "string" should be replaced by an application specific subtype which limits the string's length through "maxLength" and "minLength".

Motivation: *The detailed WSDL specification (including regular expressions) allows to much better describe the element and/or attribute values that are expected in an input parameter of the web service request. In addition to improving the security due to content validation (see [Req 7.4]), this measure shoud also simplify web service development in general (design by contract).*

[Req 7.4] WSDL and Schema Validation

All web service requests MUST be validated in the access and validation tier against the detailed WSDL specification (see [Req 7.3]) describing this request. This verification comprises verification of both the structure of the web service request (i.e., size, nesting depth, or forbidden XML tags) as well as the content (i.e., element and attributes values of input parameters). This validation MUST be performed on the normalized and canonicalized representation of the request (see [Req 6.4]). Only correctly validated requests, which satisfy all the requirements given in the detailed WSDL specification may be forwarded to subsequent processing tiers.

If the WSDL specification does not contain a formal specification of input data, other appropriate measures need to be taken to prevent injection attacks, such as generic white (or black) lists of allowed characters and/or regular expressions. These white (or black) lists need to be agreed with the implementer of the actual web service. If no such agreement can be made, the web service itself MUST be hardened against possible injection attacks.

Motivation: *Any data that is not expected by subsequent parts of the application may cause unexpected and undesired effects, such as XML injection (in particular CDATA sections), XPath injection, XQuery injection, SQL injection, LDAP injection, code injection, or command injection. If input data is properly sanitized in the DMZ, such attacks are made considerably more difficult. The use of white lists is recommended over the use of black lists since black lists tend to become obsolete over time.*

[Req 7.5] Migration of Legacy Applications

When a legacy application is extended with a new web service frontend, this may have several security implications. If the legacy application retains alternative access mechanisms, this may lead to security critical interactions between these two mechanisms that MUST be investigated. If the API of the legacy application is automatically transformed to a web services API, the unintended exposure of non-public APIs MUST be prevented. The security model of the legacy application MUST be reviewed in order to prevent unauthorized access to published APIs. Finally, since legacy applications are mostly implemented with older technology, buffer overflows need to be considered and prevented (e.g., through stringent validation of requests).

Motivation: *The security model of legacy applications is often completely different than that of web services. In particular, oftentimes these legacy applications were not intended to be used from outside the company by external parties.*

[Req 7.6] Control Exposure of Web Service Methods

The WSDL MUST only expose methods that are intended and safe for usage by external parties (see also [Req 7.5]). In particular if the WSDL is translated from another interface description language or taken from a web service that was developed for internal usage only, then the exposure of web service methods in the WSDL MUST explicitly be controlled. If the exposure is controlled manually, this MUST be performed for each newly created version of the WSDL.

Motivation: *Oftentimes web services offered to external partners are based on existing internal services. These internal services may even be implemented with other technologies that are automatically translated by gateways. Such a construction may lead to unintentional exposure of more sensitive methods that must not be accessible to external parties.*

[Req 7.7] Publishing WSDL

Oftentimes the users of a web service are known in advance and have to be registered with the operator of the web service (see [Req 4.1]). Therefore, the WSDL for a web service can be provided to the users without the need to publish the WSDL.

Motivation: *"Security by obscurity", is in and of itself not a bad security measure; it is only bad if all security relies only upon obscurity.*

[Req 7.8] Debug Code in Production Systems

Any debug code or debug APIs MUST be removed from the code base of the production system. It is recommended to automatize the management of debug code with the help of macro pre-processors.

Motivation: *Debug code may provide functionality that was never intended for external usage. Simply removing APIs is not sufficient, since it may still be possible for an attacker to take advantage of code that is unintentionally present in the production system.*

8 Statement of Compliance

SOC	Statement Of Compliance
SOPC	Statement Of Partly Compliance (reason must be given)
SONC	Statement Of Non Compliance (reason must be given)
N/A	Not Applicable

No.	Requirement	SOC	SOPC	SONC	N/A	Comment / Reason
2	**Architecture of Web Services**					
[Req 2.1]	Secure Architecture					
[Req 2.2]	Security Appliance					
[Req 2.3]	Trust Relationships					
3	**General Guidelines**					
[Req 3.1]	Fail Safe					
[Req 3.2]	Confidentiality Protection					
[Req 3.3]	Confidentiality Protection of Highly Sensitive Data					
[Req 3.4]	Integrity Protection					
[Req 3.5]	Secure Message Buffer					
[Req 3.6]	Prevention of Replay At-tacks					
[Req 3.7]	Logging of Security Re-lated Events					
[Req 3.8]	Monitor and Analyze Secu-rity Related Events					
[Req 3.9]	No Disclosure Of Unneces-sary Information					
[Req 3.10]	Estimate and minimize the risks of Denial of Service					
4	**Authentication and Au-thorization**					
[Req 4.1]	Strong Sender Authentica-tion					
[Req 4.2]	Recipient Authentication					
[Req 4.3]	Proper Certificate Handling					
[Req 4.4]	Protect Security Relevant Data					
[Req 4.5]	Authentication Data in the MZ					
[Req 4.6]	Communication of Authen-ticated Identity					
[Req 4.7]	Forwarding of Signature Data					
[Req 4.8]	Authorization Data in the MZ					
[Req 4.9]	Policy based Authoriza-tionPolicy based Authoriza-tion					

[Req 4.10]	Key Data in the MZ					
5	Session Management					
[Req 5.1]	Protect Session Identifiers					
[Req 5.2]	Cryptographically Strong Session Identifiers					
[Req 5.3]	Session State Management Outside of Access and Validation Tier					
[Req 5.4]	Session Identifier Verification					
[Req 5.5]	Unique Session Identifier					
[Req 5.6]	Fresh Session Identifier After Authentication					
[Req 5.7]	Invalidate Session State					
[Req 5.8]	Session Timeout					
6	SOAP and XML					
[Req 6.1]	Conformance to Basic Profile and Basic Security Profile					
[Req 6.2]	Secure Use of XML and Web Technologies					
[Req 6.3]	Parser and Resource Restrictions					
[Req 6.4]	Canonicalization Methods (XMLC14N and EXCC14N respectively)					
[Req 6.5]	Avoid Reencryption of Encrypted XML Data					
[Req 6.7]	Prefix the Encryption Key to the Encrypted Data					
7	WSDL and UDDI					
[Req 7.1]	Web Service Metadata in the MZ					
[Req 7.2]	Secure UDDI					
[Req 7.3]	Detailed WSDL Specification					
[Req 7.4]	WSDL and Schema Validation					
[Req 7.5]	Migration of Legacy Applications					
[Req 7.6]	Control Exposure of Web Service Methods					
[Req 7.7]	Publishing WSDL					
[Req 7.8]	Debug Code in Production Systems					

9 Checklist

Result:

P	Passed
F	Failed
A	Acceptable if reason is given: requirement is not fulfilled as defined, but implementation provide acceptable security level (target/sense of requirement is (nearly) reached)
NC	Not Checked
NA	Not Available / Not Applicable
X	Requirement can't be checked in a quick check

No.	Requirement	Result	Comment / Reason
2	Architecture of Web Services		
[Req 2.1]	Secure Architecture		
[Req 2.2]	Security Appliance		
[Req 2.3]	Trust Relationships		
3	General Guidelines		
[Req 3.1]	Fail Safe		
[Req 3.2]	Confidentiality Protection		
[Req 3.3]	Confidentiality Protection of Highly Sensitive Data		
[Req 3.4]	Integrity Protection		
[Req 3.5]	Secure Message Buffer		
[Req 3.6]	Prevention of Replay Attacks		
[Req 3.7]	Logging of Security Related Events		
[Req 3.8]	Monitor and Analyze Security Related Events		
[Req 3.9]	No Disclosure Of Unnecessary Information		
[Req 3.10]	Estimate and minimize the risks of Denial of Service		
4	Authentication and Authorization		
[Req 4.1]	Strong Sender Authentication		
[Req 4.2]	Recipient Authentication		
[Req 4.3]	Proper Certificate Handling		
[Req 4.4]	Protect Security Relevant Data		
[Req 4.5]	Authentication Data in the MZ		
[Req 4.6]	Communication of Authenticated Identity		
[Req 4.7]	Forwarding of Signature Data		

[Req 4.8]	Authorization Data in the MZ		
[Req 4.9]	Policy based Authoriza-tionPolicy based Authorization		
[Req 4.10]	Key Data in the MZ		
5	Session Management		
[Req 5.1]	Protect Session Identifiers		
[Req 5.2]	Cryptographically Strong Ses-sion Identifiers		
[Req 5.3]	Session State Management Outside of Access and Valida-tion Tier		
[Req 5.4]	Session Identifier Verification		
[Req 5.5]	Unique Session Identifier		
[Req 5.6]	Fresh Session Identifier After Authentication		
[Req 5.7]	Invalidate Session State		
[Req 5.8]	Session Timeout		
6	SOAP and XML		
[Req 6.1]	Conformance to Basic Profile and Basic Security Profile		
[Req 6.2]	Secure Use of XML and Web Technologies		
[Req 6.3]	Parser and Resource Restric-tions		
[Req 6.4]	Canonicalization Methods (XMLC14N and EXCC14N respectively)		
[Req 6.5]	Avoid Reencryption of En-crypted XML Data		
[Req 6.7]	Prefix the Encryption Key to the Encrypted Data		
7	WSDL and UDDI		
[Req 7.1]	Web Service Metadata in the MZ		
[Req 7.2]	Secure UDDI		
[Req 7.3]	Detailed WSDL Specification		
[Req 7.4]	WSDL and Schema Validation		
[Req 7.5]	Migration of Legacy Applica-tions		
[Req 7.6]	Control Exposure of Web Ser-vice Methods		
[Req 7.7]	Publishing WSDL		
[Req 7.8]	Debug Code in Production Systems		

Annex

A Other applicable documents

No.	Title of internal references	Doc. number	Version	Notes
[WebApps]	Security Requirements on Web Applications	S.7.1.1	V1.0	
[WebServer]	Security Requirements on Web Servers	S.7.1.3	V1.0	
[SecGuide]	Security Guide T-Mobile, T-Mobile Security and TSI Security Management			
[SecProxy]	System Architecture Security Proxy, Internet Integration Framework		V0.4	Uta Pollmann

No.	Title of external references	Version	Author

B Abbreviations, terms and definitions

B.1 List of abbreviations

Abbreviation	Importance
GGS	Global Group Security
GGSL	Global Group Security Library
XML	eXtensible Markup Language
SOAP	Originally: Simple Object Access Protocol; since version 1.2 it is not considered an acronym any more
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery and Integration
SOA	Service Oriented Architecture
DMZ	Demilitarized Zone
MZ	Militarized Zone
PKI	Public Key Infrastructure
IAM	Identity and Access Management

B.2 Terms and definitions

This section defines the status given to requirements identified in this document. These definitions are interpreted as in the IETF standard described in RFC2119:

Term	Description
MUST	This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
MUST NOT	This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
SHOULD	This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behaviour is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.
MAY	This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

C Example for Detailed WSDL Specification

EchoMailData.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="EchoMailData"
    targetNamespace="http://www.t-mobile.com/EchoMailData.wsdl"
    xmlns:ead="http://www.t-mobile.com/EmailAddressDataDefinition.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.t-mobile.com/EchoMailData.wsdl"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
                  Sample WSDL and XML schema file</wsdl:documentation>

    <wsdl:import
      location="http://www.t-mobile.com/EmailAddressDataDefinition.xsd"
      namespace="http://www.t-mobile.com/EmailAddressDataDefinition.xsd"/>

    <wsdl:message name="EchoMailDataRequest">
        <wsdl:part name="requestMail" type="ead:EmailAddress"/>
    </wsdl:message>

    <wsdl:message name="EchoMailDataResponse">
        <wsdl:part name="responseMailData" type="ead:EmailAddressData"/>
    </wsdl:message>

    <wsdl:portType name="EchoMailDataPortType">
        <wsdl:operation name="EchoMailData">
            <wsdl:input message="tns:EchoMailDataRequest"/>
            <wsdl:output message="tns:EchoMailDataResponse"/>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name="EchoMailDataBinding"
                                      type="tns:EchoMailDataPortType">
  <soap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="EchoMailData">
            <soap:operation soapAction="EchoMailData"/>
            <wsdl:input>
                <soap:body
                    encodingStyle=
                               "http://schemas.xmlsoap.org/soap/encoding/"
                    parts="requestMail"
                    use="encoded"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body
                    encodingStyle=
                               "http://schemas.xmlsoap.org/soap/encoding/"
                    parts="responseMailData"
                    use="encoded"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="EchoMailData">
```

```
        <wsdl:port binding="tns:EchoMailDataBinding"
                                                  name="EchoMailDataPort">
            <soap:address location=
                     "http://www.t-mobile.com:9999/samples/EchoMailData"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

EmailAddressDataDefinition.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace=
                  "http://www.t-mobile.com/EmailAddressDataDefinition.xsd"
            xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:etd=
                 "http://www.t-mobile.com/EmailAddressDataDefinition.xsd">

  <xsd:simpleType name="EmailAddress">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="6"/>
      <xsd:maxLength value="100"/>
      <xsd:pattern value="([A-Za-z0-9]+[\-._+\#%/=])*[A-Za-z0-9]+@
                                        ([A-Za-z0-9\-]+[.])+[A-Za-z]{2,6}"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="EmailAddressData">
    <xsd:sequence>
      <xsd:element name="Address" type="EmailAddress" minOccurs="1"/>
      <xsd:element name="FirstName" type="xsd:string" maxLength="50"/>
      <xsd:element name="LastName" type="xsd:string" maxLength="50"/>
    </xsd:sequence>
    <xsd:attribute name="EmployeeType" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="(internal | external)"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>

</xsd:schema>
```