Command Injection in XML Signatures and Encryption

Bradley W. Hill

Information Security Partners, July 12, 2007
brad@isecpartners.com

Abstract. The XML Digital Signature[1] (XMLDSIG) and XML Encryption[2] (XMLENC) standards are complex protocols for securing XML and other content. Among its complexities, the XMLDSIG standard specifies various "`Transform`" algorithms to identify, manipulate and canonicalize signed content and key material. Unfortunately, the defined transforms have not been rigorously constrained to prevent their use as attack vectors, and denial of service or even arbitrary code execution are probable in implementations that have not specifically guarded against such risks.

Attacks against the processing application can be embedded in the `KeyInfo` portion of a signature, making them inherently unauthenticated, or in the `SignedInfo` block. Although tampering with the `SignedInfo` should be detectable, a defective implied order of operations in the specification may still allow unauthenticated attacks here.

The ability to execute arbitrary code and perform file system operations with a malicious, invalid signature has been confirmed by the researcher in at least two independent XMLDSIG implementations, and other implementations may be similarly vulnerable. This paper describes the vulnerabilities in detail and offers advice for remediation. The most damaging attack is also likely to apply in other contexts where XSLT is accepted as input, and should be considered by all implementers of complex XML processing systems.

Categories and Subject Descriptors

Primary Classification:

K.6.5	Security and Protection
Subject:	Invasive software
	Unauthorized access
	Authentication

Additional Classification:

D.2.3	Coding Tools and Techniques (REVISED)
Subject:	Standards
D.2.1	Requirements/Specifications (D.3.1)
Subject:	Languages
	Tools

General Terms: Security, Reliability, Verification, and Design.

Keywords: XML Signatures, XML Encryption, XML Signature Transforms, XSLT, Security, Vulnerability, Attack, Attack Surface, and Countermeasure

1 Introduction

The XML Digital Signature (XMLDSIG) specification describes a method and syntax for creating digital signatures[3] in an XML[4] format. The signatures may be over arbitrary content, but the specification provides a variety of features and procedures tailored specifically for signing XML Information Sets[5] and creating signatures that are resilient through various transformations and manipulations. This paper assumes some familiarity with the XMLDSIG specification and its terms, and an exhaustive introduction is beyond the scope of this paper. Non-normative, introductory references to the technologies include [[6]] and [[7]].

Signed content in an XML Signature is identified by `Reference` elements inside the `SignedInfo` element of the `Signature`. Each `Reference` may have an associated set of `Transforms`. A `Transform` may be an arbitrary algorithm, indicated by a URI. The core specification defines a set of mandatory `Transform` algorithms that must be supported by a compliant implementation and one optional algorithm. Other standards that depend on XMLDSIG may define their own required and optional `Transform` algorithms. The XML Digital Signature syntax also defines a `KeyInfo` element, which allows for key material to be described by reference (via the `RetrievalMethod` element) or directly included in a signature. `Transforms` may also be applied to `KeyInfo` elements identified by `RetrievalMethod`.

Both in the abstract and as specified, transforms may be arbitrarily complex operations, and there can be security risks associated with executing 'active' transforms, where the operational semantics can be specified or influenced by the creator of the signature. The consequences of processing a maliciously injected `Transform` may range from denial of service to arbitrary code execution.

The XML Encryption standard uses a great deal of syntax and behavior inherited from XMLDSIG, especially for identifying key material and the same risks apply.

This paper will discuss these risks in detail and offer guidance to implementers of the XMLDSIG and XMLENC specifications for avoiding and/or mitigating these risks.

1.1 Prior Work and Motivation

After independently identifying these issues, the author has found that the defect in the implied order of operations was noted by Laurence Bull and David M. Squire in 2004[8]. Bull & Squire only discuss the potential defect in the context of their primary proposal for including arbitrary executable content as part of a signature in a custom `Transform`, which, while interesting, this author believes would only make the bad security situation around XML Signatures drastically worse.

This paper identifies actual attacks possible against the base specification as defined and implemented. Though the defect in order of operations makes the vulnerabilities more accessible, they are quite serious even with a more conservative ordering of operations, especially when `KeyInfo` material is processed.

2 Risks and Vulnerabilities

2.1 Security Risks of the XSLT Transform

Serious security implications exist with support for the Extensible Stylesheet Language Transform (XSLT) Transform[9], an OPTIONAL algorithm in the XMLDSIG core specification, identified by the URI "`http://www.w3.org/TR/1999/REC-xslt-19991116`". A non-normative, introductory reference to XSLT is [[10]]. XSLT is a programming language designed for content-generation by select and merge operations over data sets and templates. XSLT is commonly thought of as a limited tool language for document processing but is, in fact, Turing-complete.[11]

In other words, the submitter of a signature may, by design, include an arbitrary program which must be executed to determine the validity of the signature. This, of course, should raise immediate security concerns. Assuring the security properties of mobile code is a distinct and more difficult class of problem than mere integrity. Denial of service attacks are an implicit possibility – a very few messages could trivially disable systems willing to accept such commands.

The guiding assumption in allowing XSLT must have been that it is a functional, declarative language; programs formally lack side-effects, facilities for I/O and access to the operating system. Yet, this formal assumption is false in most cases. The base specification allows for network operations that may not be security neutral, such as the `xsl:include`, `xsl:import` and `document()` functions. Additionally XSLT exposes idiosyncratic and dangerous attack surfaces via extension mechanisms.

Nearly all XSLT processor implementations allow for the definition of extensions, and commonly include and support a set of default extensions. It is typical for these extensions to enable file system operations, inclusion of functions written in a variety of scripting languages, mapping of XML namespaces to class libraries or functions in the processor's implementation language, and even connecting to a database and executing SQL. These operations are clearly security-relevant and almost certainly not expected or intended behavior for signature validation. Implementers of XMLDSIG processors who rely on their default platform XSLT services may be inadvertently exposing these services through the XMLDSIG interface.

2.1.1 Exploiting XSLT Extensions

While XSLT is standardized, extension mechanisms are usually platform specific. Space does not permit an exhaustive discussion here, but extension mechanisms with security-critical side effects exist in the majority of XSLT processing engines available, including Xalan-J[12],, Xalan-XSLTC[13], Saxon[14], MSXML[15], jd.xslt[16], Oracle XDK 10g[17], Sablotron[18], XT[19] and Unicorn[20]. Additional information on utilizing XSLT extensions in common platforms and their security implications can be found at [[21]], [[22]], [[23]] and [[24]].

The following sample XML Signature syntax illustrates exploitation of an extension mechanism, in this case the Java class mapping feature available in several flavors of Xalan, the most widely used XSLT processor for the Java environment. This signature, though invalid, will execute the program "`c:\Windows\system32\cmd.exe`" when processed by a vulnerable implemenation. The syntax is a bit unusual, as the object-oriented, imperative style of Java must be mapped into the static, functional style of XSLT, but the code is relatively straightforward. Lines 12-13 declare mappings between XML namespaces and classes in the standard Java library. Line 16 instantiates an instance of the java.lang.Runtime class, and line 17 calls the exec() method to execute a system command. Lines 18 and 19 convert and select our objects as String representations, to work around optimizations performed by the XSLT processor.

This particular choice of program is harmless, but any command or any Java code may be substituted here. On a properly configured host, e.g. a UNC path could be specified on the command line to download and execute arbitrary code from a remote host, or a remote JAR file downloaded and directly executed in the JVM.

```
01: <?xml version="1.0" encoding="UTF-8"?>
02: <Envelope xmlns="urn:envelope">
03: <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
04: <SignedInfo>
05: <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComnts"/>
06: <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
07: <Reference URI="">
08: <Transforms>
09: <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
10: <Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
11: <xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object"
exclude-result-prefixes= "rt,ob">
<xsl:template match="/">
<xsl:variable name="runtimeObject" select="rt:getRuntime()"/>
<xsl:variable name="command" select="rt:exec($runtimeObject,&apos;c:\Windows\system32\cmd.exe&apos;)"/>
<xsl:variable name="commandAsString"   select="ob:toString($command)"/>
<xsl:value-of select="$commandAsString"/>
</xsl:template>
</xsl:stylesheet>
</Transform>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>uooqbWYa5VCqcJCbuymBKqm17vY=</DigestValue></Reference></SignedInfo>

<SignatureValue>hYlWlHBy+nwft0pcr64IdS3Hobd+RhAF6kZa1ZwA6EW3gavRXGnxIkBJo2Bish951xd0woMrMbr4EtvUY+KaDr2qvylPjVbFhh7Mr4By+DU7x/AFODhjE7DrAcszscmLDUP
X24+0mdshbbzsUbbapMLDexGm+1F6Id0mpjqdHxQ=</SignatureValue>
<KeyInfo>
<X509Data>

<X509Certificate>MIICMzCCAZygAwIBAgIEB1vNFTANBgkqhkiG9w0BAQUFADBdMR0wGwYDVQQKExREb2N0b3IgRXZpbCBOZXR3b3JrczEvMC0GA1UECxMmTWFuSW5UaGVNaW
RkbGUgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkxCzAJBgNVBAYTAlVTMB4XDTA1MDYyNjAwNDMxMloXDTA3MDYxNjAwNDMwOVowDjEMMAoGA1UEAxMDZm9vMIGfMA0GCSqGSI
b3DQEBAQUAA4GNADCBiQKBgQCQtSkEzUfVcVS0pQ/9EGVp4VzAKAXEh/LnhziJMflbQ+l2ZP9f43AhtF8F7crEDiO8roDM5hHl+pRsIKts8/JFGFVFhoEnqmJ1YgmWCXzojbl02MwtpoU4
Qt3jDQu5A7CAcwjZHBFpHkKpfW6EDNRiPkLwDZehU3kUGg5TuN0BqwIDAQABo08wTTAdBgNVHQ4EFgQUT1kDd5c4i9PV8gjHcPjq9C+Z6EowHwYDVR0jBBgwFoAUcaZ8Le2eELaUj
56dgeeGfu1pnoowCwYDVR0PBAQDAgSQMA0GCSqGSIb3DQEBBQUAA4GBAJJLlUiXACfCfqF6uAEr2GjZOx07PWOgmRiX9yA+cVSpqlKu8rCz1x0+jd5F72tj3seVuUT0uXgSTZLItwbB
WNPIscnHcv+wh95JzEOLkhT4wEEdu0p6zdG9DMj7I4s/j69zOzX95B+FLwAGfjyL5MoK+BHKOMr/tZ8TJEXUsmz5</X509Certificate>
</X509Data>
</KeyInfo>
</Signature>
</Envelope>
```

Example 1

Although there likely does not exist a sufficient density of vulnerable, publicly available XML Signature processing hosts today, this exploit could be used to create a cross-platform worm, especially when combined with services like the UDDI[25] that provide tailor-made targeting and replication information for malicious software using Web Services as a substrate. UDDI Version 3 itself supports XML Digital Signatures and thus may also be a carrier of such attacks[26] to vulnerable clients. A single signature could contain multiple malicious transforms targeted to a variety of platforms and configurations, providing fairly reliable execution in a single message, and both clients and servers are likely to share signature implementations and vulnerable to identical attacks.

2.2 Defect in Implied Order of Operations for Signature Validation

The XMLDSIG core specification offers the following guidance on signature validation:

> **3.2 Core Validation**
>
> The REQUIRED steps of core validation include (1) reference validation, the verification of the digest contained in each `Reference` in `SignedInfo`, and (2) the cryptographic signature validation of the signature calculated over `SignedInfo`.

Although this guidance is informally "non-normative" according to email discussions in the standards & implementers group[27], it clearly seems to imply a defective order of operations. While the order of operations is unimportant vis-à-vis the cryptographic integrity properties of the signature, it can have a great deal of impact on the overall security properties of the validation operation itself, as reference validation inherently has side effects on the validating system. At a minimum, reference validation has non-deterministic characteristics with regard to performance and resource-consumption. In the case of 'active' transformations, such as XSLT, we have seen that reference validation is an attack surface for command injection. An implementation following the implied order of operations can be attacked by an adversary who is simply able to tamper with a signature and insert processing instructions into the `SignedInfo`; attacking the cryptographic properties of the signature becomes unnecessary.

2.3 Risks of Key Resolution

Signature verification may use a key explicitly supplied by the caller, but the XMLDSIG specification also provides methods for keys to be directly included, identified or described by reference in the `Signature` via the `KeyInfo` element.

Keys may commonly be identified by a URI, e.g. a security token identified by a same-document reference. This is specified by a `RetrievalMethod` element. For any given signature, the `KeyInfo` element is optional, but the XMLDSIG core specification indicates that compliant versions MUST implement support for `KeyInfo` and SHOULD implement `RetrievalMethod`. From §4.4.3:

> `RetrievalMethod` uses the same syntax and dereferencing behavior as `Reference`'s URI (section 4.3.3.1) and The Reference Processing Model (section 4.3.3.2) except that there is no `DigestMethod` or `DigestValue` child elements and presence of the URI is mandatory.

`Transforms` may be specified for a `RetrievalMethod` as they are for `References`, and the `KeyInfo` element cannot be authenticated. (The `KeyInfo` element may be part of the content protected by a signature, but to actually authenticate it would require validating the signature, which requires a key...) Implementations that support `RetrievalMethod`, and particularly `Transforms`, in `KeyInfo` inherently admit the possibility that an attacker can inject processing instructions and must mitigate such risks appropriately. It is likely not the case that an identical set of algorithms will be appropriate for both the (anonymous) `KeyInfo` and the (authenticated) `SignedInfo`. Implementations that process `KeyInfo` may be vulnerable to attacks by injected transforms even if following a conservative order of operations for signature validation.

2.4 Denial of Service attacks with transform and remote reference injection

The XSLT transform algorithm is merely the worst security risk among the set of supported transforms. If processing a defective order of operations exposes the `SignedInfo` or `KeyInfo` elements to tampering, complexity-based denial of service attacks are almost always a possibility. The XPath, XPath Filter 2.0 and Canonicalization transforms, either with complex inputs or in large multiples, can be used to consume large amounts of system resources. Further denial of service attacks may be mounted by injection of remote references, either to key material or signed content, even on systems that are highly constrained in the number and type of transform algorithms supported.

2.5 XML Encryption

XML Encryption builds on the methods and syntax established by XML Digital Signature. Encrypted content may be identified using the same syntax and dereferencing behavior as a `Reference`, and `KeyInfo` may be identified by `RetrievalMethod`. To the extent that this XMLENC behavior is identical to (and likely re-used from) an XMLDSIG implementation, the same risks apply.

3. Vulnerable Implementations

Based on the author's investigations and communication with vendors, the following products were vound to be vulnerable to anonymous, arbitrary code execution via XSLT during XML signature processing:

- Sun Microsystems
    - JSR 105 Reference Implementation
    - Java Web Services Developer Pack (JWSDP) version 1.5
    - Java Web Services Developer Pack (JWSDP) version 2.0
    - Java Platform, Standard Edition 6.0
    - Sun Java System Web Server version 7.0
    - Sun Java System Application Server Platform Edition version 8.2
    - Sun Java System Application Server Enterprise Edition version 8.2
    - Sun Java System Application Server Platform Edition version 9.0
- Institute for Applied Information Processing and Communication (IAIK)
    - XML Security Toolkit (XSECT) versions < 1.10
    - XML Signature Library (IXSIL) all versions

Vendors were notified on January 15th, 2007, and updates to protect against these attacks are included in the Java Platform, Standard Edition 6.0 update 2, released on July 4, 2007, various patches available at http://sunsolve.sun.com/search/document.do?assetkey=1-26-102992-1, released July 10, 2007, and in XSECT 1.10 and a maintenance update of IXSIL, both released March 28, 2007 and available at http://jce.iaik.tugraz.at/sic/support. In all cases, the vendors were utilizing variants of the Xalan XSLT processor and were unaware that it enables nonstandard extensions in the default configuration.

The denial of service risks which arise from the defective order of operations have been taken less seriously and the author is aware of major implementations vulnerable to these attacks (but not the XSLT command injection) that have declined, thus far, to patch. In addition, the author has identified over 18 distinct, commercial implementations of XML Digital Signature technology, and while efforts have been made to contact the maintainers, it has only been possible to directly evaluate a small handful, and certainly more private or embedded implementations have gone undetected. The author considers it not unlikely that other implementations will be found to be vulnerable to these design defects as the attack methods become more widely known.

4. Security Guidance for Implementers

4.1 `Transform`, `Reference` and `KeyInfo` Processing

Implementers SHOULD disable the XSLT transform by default. This transform is OPTIONAL and cannot be relied upon for interoperability purposes. In the case that the XSLT transform is required:

- The caller SHOULD be required to explicitly enable the algorithm.
- The XSLT processor SHOULD have security-sensitive extensions disabled.
    - Implementers must be wary of using shared, platform-default services, such as JAXP[28], as the properties of these processors may change asynchronously. An instantiation and configuration private to the signature validation system is recommended.
    - Java XSLT processors supporting JAXP 1.3 or above may offer support for a "Secure processing" feature, identified by the URI "`http://javax.xml.XMLConstants/feature/secure-processing`". The semantics of this flag are implementation specific, but for Xalan and variants this will disable extension functions and elements.[29]
- The XSLT transform SHOULD NOT be a supported algorithm for `RetrievalMethod` without explicit user consent. Enabling the XSLT transform for `Reference`s SHOULD NOT enable it for `RetrievalMethod`s as a side effect.

Implementers SHOULD carefully consider the security implications of all transform algorithms, and whether it is appropriate to execute these processing instructions from anonymous and authenticated originators as part of signature validation. Callers SHOULD have the ability to explicitly enumerate all supported transform algorithms, and enable or disable them selectively and independently for both `Reference` and `RetrievalMethod` processing.

Callers SHOULD have the ability to set hard timeout values and limit the total amount of system resources consumed when validating signatures or dereferencing key material to mitigate a variety of complexity and network related denial of service risks inherent in XML Signatures, of which the XSLT transform is only one.

Callers SHOULD have the ability to use URI resolvers with different properties for processing the anonymous `KeyInfo` and the authenticated `SignedInfo`. For example, a caller may be willing to dereference remote URIs in `SignedInfo` after authenticating the originator, but only allow same-document references in `KeyInfo` as an attack surface reduction measure.

In general, implementers should very carefully consider whether all exposed dependencies have been properly hardened against malicious input. For implementations in a virtual machine environment with a security manager, the author recommends the use of Permissions[30] or Code Access Security[31] to constrain the signature validator in a sandbox and provide defense in depth against unexpected or unconstrained by-design behavior and implementation flaws.

4.2 Order of Operations

From the cryptographic perspective, signature validation is a pure function, but following a proper order of operations when validating a signature can substantially reduce the attack surface of a concrete implementation. The signature should be able to be divided into two classes of attack surface to which differing levels of restriction may apply: anonymous and authenticated. `KeyInfo` is always anonymous, but the processing instructions in `SignedInfo` can be authenticated.

The following order of operations SHOULD be supported by an XML Signature API:

1. Selection of a trusted key.
   a. If `KeyInfo` is to be used, the user must have the option to extract the key first and make a trust decision, before continuing with core validation.
   b. APIs of the form: "`KeyInfo validate()`", which only return a key after performing all of core validation, unacceptably expose the instructions in `SignedInfo` on the anonymous attack surface because the returned key may not be trusted by the caller and all operations are completed before a trust decision can be made.

2. Cryptographic signature validation of the signature calculated over `SignedInfo`. This assures that the `SignedInfo` has not been tampered with.

3. At this point, the processing instructions have been authenticated, and the caller may choose to proceed to reference validation, the verification of the digest contained in each `Reference` in `SignedInfo`.

5. Conclusion

XML Signatures and Encryption contain complex sets of processing instructions that resemble a network protocol or active file format. Examining the specification in this context reveals important security properties that are orthogonal to its cryptographic properties. An order of operations that is unimportant from the perspective of symbolic logic is seen to be very important from the perspective of application attack surface reduction. Like all complex inputs received from unauthenticated sources, care must be taken by applications processing XML Signatures to constrain allowed behavior and handle anonymous and unauthorized inputs with appropriate caution. Implementers should thoroughly investigate and understand the risks and properties of all dependencies included in the processing path and mitigate threats to these subsystems as necessary.

XMLDSIG and XMLENC implementing applications can be made more secure with a few constraints and clarifications to the processing model. Although the most aggressive changes recommended here may have some negative impact on interoperability, the author hopes that ongoing work on security profiles[32] and dynamic exchange of policy[33] will also be of help in specifying and creating systems using these technologies that are both secure and interoperable.

General lessons can also be taken for all XML processing applications. In particular, the use of XSLT should be carefully and skeptically evaluated in any security-critical context. Stylesheets are programs and should not be accepted from untrusted sources. This advice applies not just to XSLT, but other XML programming languages such as XQuery. The widespread adoption of XML has generally been extremely positive from an application security perspective, but as XML-consuming applications increase in complexity, they also increase in risk. Even non-Turing-complete XML sub-dialects and related tools like WSDL, Schema and XPointer can contain fearsome complexity that may be implemented with or backed by code-generation facilities. Where XML is or can be transformed into code, great care must be taken to prevent serious security vulnerabilities.

Acknowledgements: The author would like to thank the following people, whose patient attention and helpful comments assisted in developing this paper and the techniques described herein: Laurence Bull, Jesse Burns, Blair Dillaway, Christian Geuer-Pollmann, Brian LaMacchia and Sean Mullan.

[1] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-Signature Syntax and Processing. In D. Eastlake, J. Reagle, and D. Solo, editors, W3C Recommendation. World Wide Web Consortium, 12 February 2002.
`http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`
[2] T. Imamura, B. Dillaway and E. Simon. XML Encryption Syntax and Processing. In D. Eastlake, J. Reagle, editors, W3C Recommendation. World Wide Web Consortium, 10 December 2002.
`http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/`
[3] T. Beth, M. Frisch, and G.J. Simmons, editors. Public-Key Cryptography: State of the Art and Future Directions, volume 578 of Lecture Notes in Computer Science. Springer, 3–6 July 1992. E.I.S.S.Workshop Oberwolfach Final Report.
[4] Extensible Markup Language (XML) 1.0 (Fourth Edition). T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, editors. W3C Recommendation. World Wide Web Consortium, 16 August 2006, edited in place 29 September 2006.
[5] J. Cowan and R. Tobin, editors, *XML Information Set (Second Edition)*, W3C (MIT, ERCIM, Keio), 4 February 2004
[6] D. Eastlake and K. Niles, *Secure XML: The New Syntax for Signatures and Encryption*, Pearson Education, July 19, 2002
[7] J. Rosenberg and D. Remy, *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature and XML Encryption*, Sams, 12 May 2004
[8] L. Bull and D. Squire, XML Signature Extensibility Using Custom Transforms, in *Web Information Systems – WISE 2004*, pp 102-112. Springer Berlin / Heidelberg, November 2004
`http://springerlink.com/content/qp0eyrbgdcn47jh1`
[9] XSL Transformations (XSLT) Version 1.0. J. Clark, editor, W3C Recommendation, World Wide Web Consortium, 16 November 1999.
`http://www.w3c.org/TR/1999/REC-xslt-19991116`
[10] D. Tidwell, *XSLT*, O'Reilly Media, 15 August 2001
[11] Brainerd, W.S., Landweber, L.H. (1974), *Theory of Computation*, Wiley
[12] O. Predescu, et al., Xalan-Java, The Apache Software Foundation, Hewlett Packard Corporation, IBM Corporation, Sun Microsystems and Lotus Development Corporation 1999-2007.
`http://xml.apache.org/xalan-j/`
[13] Ibid.
[14] M. Kay, SAXON, M. Kay 2007
`http://saxon.sourceforge.net/`
[15] MSXML, Microsoft Corporation. 2000-2007
`http://msdn.microsoft.com/xml/default.aspx`
[16] J. Döbler, jd.xslt, Aztecrider, 2001

[17] Oracle XML Developers Kit, XDK 10g Production, Oracle Corporation, 2004-2006
`http://www.oracle.com/technology/tech/xml/xdk/software/production10g/index.html`
[18] Sablotron, Ginger Alliance 2006
`http://www.gingerall.org/sablotron.html`
[19] J. Clark and B. Lindsey, XT 2006
`http://www.blnz.com/xt/index.html`
[20] Unicorn XSLT Processor, Unicorn Enterprises 2000-2003
`http://www.unicorn-enterprises.com/products_uxt.html`
[21] A. Skonnard, Extending XSLT with JScript, C#, and Visual Basic .NET, MSDN Magazine, Microsoft, Corporation, March 2002.
`http://msdn.microsoft.com/msdnmag/issues/02/03/xml/`
[22] E. Harold, Simple Xalan Extension Functions: Mixing Java with XSLT, IBM developerWorks, 07 November 2006
`http://www-128.ibm.com/developerworks/library/x-xalanextensions.html`
[23] Xalan-Java Extensions, The Apache Software Foundation, 2005
`http://xml.apache.org/xalan-j/extensions.html`
[24] XSLT Security, MSDN Library, Microsoft Corporation, 2007
`http://msdn2.microsoft.com/en-us/library/ms763800.aspx`
[25] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. Dovey, D. Feygin, A. Hately, R. Kochman, P. Macias, M. Novotny, M. Paolucci, C. Riegen, T. Rogers, K. Sycara, P. Wenzel, and Zhe Wu, UDDI Version 3.0.2. UDDI Spec Technical Committee Draft, Dated 20041019, L. Clement, A. Hately, C. Reigen and T. Rogers, editors., Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Microsoft Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., and VeriSign, Inc. 2001-2002, OASIS Open 2002-2004
`http://uddi.org/pubs/uddi-v3.0.2-20041019.htm`
[26] Ibid.
`http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908441`
[27] `http://lists.w3.org/Archives/Public/w3c-ietf-xmldsig/2001OctDec/0064`
[28] Java API for XML Processing (JAXP), Sun Developer Network, Sun Microsystems, Inc. 2007
`http://java.sun.com/webservices/jaxp/`
[29] Transform Features, Apache Software Foundation, 2005
`http://xml.apache.org/xalan-j/features.html#secureprocessing`
[30] L. Gong, Java™ 2 Platform Security Architecture, Sun Microsystems, Inc. 2002-2007
`http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc3.html#19802`
[31] Code Access Security, .NET Framework Developer's Guide, Microsoft Corporation, 2007
`http://msdn2.microsoft.com/en-us/library/930b76w0(VS.80).aspx`
[32] Basic Security Profile Version 1.1, Working Group Draft, M. McIntosh, M. Gudgin, K. S. Morrison, A. Barbir, editors. Web Services Interoperability Organization, 2006-10-19
`http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html`
[33] G. Della-Libera, M. Gudgin, P. Hallam-Baker, M. Hondo, H. Granqvist, C. Kaler, H. Maruyama, M. McIntosh, A. Nadalin, N. Nagaratnam, R. Philpott, H. Prafullchandra, J. Shewchuk, D. Walter, and R. Zolfonoon, Web Services Security Policy Language, C. Kaler and A. Nadalin, editors. International Business Machines Corporation, Microsoft Corporation, RSA Security, Inc. and VeriSign, Inc., July 2005
`http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf`