Usages offensifs de XSLT

Nicolas Grégoire

Agarri
nicolas.gregoire(@)agarri.fr
http://www.agarri.fr/

Résumé Le but de cet article est de montrer, par une analyse ascendante, les risques engendrés par l'utilisation irréfléchie de moteurs XSLT. Après avoir introduit XSLT, nous rappellerons les différents standards et listerons les fonctionnalités dangereuses qu'ils proposent. Puis, nous détaillerons les fonctionnalités spécifiques aux différents moteurs étudiés. Nous essaierons alors d'identifier des applications exposant involontairement des fonctionnalités XSLT dangereuses, que celles-ci proviennent de normes ou d'extensions propriétaires. Pour chacune des primitives d'exploitation, nous balayerons les possibilités à un attaquant. Nous nous intéresserons enfin à la sécurisation d'applications utilisant les moteurs XSLT.

1 Introduction

1.1 Contexte

La plupart des développements logiciels modernes utilisent des bibliothèques ou *frameworks* sous-jacents. Typiquement, le support du format XML est souvent réalisé à partir d'une bibliothèque tierce. Le choix de l'une ou l'autre de ces bibliothèques repose la plupart du temps sur des critères très éloignés de la sécurité, comme les performances, le type de licence ou l'intégration avec un langage de programmation donné.

Cette tendance se confirme lorsque les fonctionnalités proposées aux utilisateurs se complexifient. Par exemple, une application de vérification de signature électronique au format XML-DSig doit supporter le protocole XML, pouvoir réaliser des accès réseau (récupération des CRL en HTTP ou LDAP), traiter des données encodées en ASN.1 (LDAP, certificats X.509),... et peut faire appel à une dizaine de bibliothèques différentes.

Ainsi, certaines fonctionnalités avancées de moteurs de transformation XML (XSLT) sont totalement ignorées des concepteurs et développeurs d'applications les utilisant. Ces fonctionnalités pourraient être détournées et permettre à un attaquant de réaliser diverses actions malveillantes.

1.2 Motivation

Les moteurs XSLT semblent être un sujet de recherche intéressant pour plusieurs raisons :

– Ces moteurs sont usuellement distribués sous forme de bibliothèques, avec un wrapper simple permettant d'exercer l'API proposée. Ces bibliothèques sont utilisées soit directement par une application soit indirectement par une bibliothèque de plus haut niveau elle-même employée par d'autres bibliothèques ... Ceci implique qu'une « fonctionnalité dangereuse » d'un moteur XSLT très utilisé peut impacter de nombreuses applications, éventuellement très différentes (serveur d'application, logiciel bureautique, navigateur, ...) ;
– Les fonctionnalités « dangereuses » sont très disparates selon les moteurs. De plus, la plupart des usages classiques de la transformation XML ne nécessite pas leur utilisation. Elles sont donc assez peu connues et régulièrement oubliées lors des phases de revue de la surface d'attaque exposée ;
– Les quelques publications relatives à l'utilisation offensive de XSLT (cf. section « Autres travaux sur le sujet ») ont eu relativement peu d'écho, bien qu'elles démontrent que des composants très divers sont impactés par ce type de vulnérabilité ;
– A notre connaissance, aucune étude méthodique à approche ascendante (dite *bottom-up*, c'est-à-dire partant d'un ensemble de moteurs XSLT et remontant aux applications « métier ») n'a été publiée.

1.3 Problématique

Les applications de haut niveau utilisent de nombreux composants de base riches en fonctionnalités, comme les moteurs XSLT. L'absence de maîtrise de chacun de ces composants, et donc l'absence de maîtrise globale du code utilisé dans l'application, peut engendrer des problématiques de sécurité.

1.4 Hypothèse générale

On s'attend à ce qu'une étude méthodique à approche ascendante permette de mettre en évidence des vulnérabilités à impact élevé dans différentes applications mettant en œuvre des moteurs XSLT.

1.5 Hypothèses opérationnelles

– Les applications de haut niveau utilisent des composants de base sans en maîtriser totalement les fonctionnalités ;
– Les fonctionnalités de ces composants de base (comme les moteurs XSLT) ne sont pas limitées aux seules fonctionnalités exprimées pour l'application de haut niveau ;

– Ces fonctionnalités XSLT peuvent être très puissantes (accès en lecture ou écriture à des fichiers éventuellement distants, exécution de code) ;
– Les mécanismes de désactivation de fonctionnalités non nécessaires (et donc de réduction de la surface d'attaque) disponibles dans certains moteurs XSLT ne sont pas utilisés ;
– Certaines applications de haut niveau exposent sans le savoir des fonctionnalités XSLT dangereuses, créant ainsi des vulnérabilités pouvant avoir un impact de sécurité important.

1.6 Méthodologie

La méthodologie de l'étude est la suivante :

Standards
– Lister les fonctionnalités XSLT standardisées, en fonction du support de telle version de la norme (XSLT 1.0, 1.1 ou 2.0) ou d'une initiative communautaire (EXSLT) ;
– Etudier le risque potentiel causé par chacune de ces fonctionnalités ;
– Lister les fonctionnalités considérées comme dangereuses ;

Moteurs
- Collecter une liste relativement complète de moteurs XSLT, qu'ils soient génériques (libxslt du projet Gnome) ou spécifiques (Transformiix, utilisé par Firefox) ;
– Identifier les éventuelles fonctionnalités propriétaires de chacun de ces moteurs XSLT ;
– Comme pour les fonctionnalités standardisées, étudier le risque potentiel causé par chacune de ces fonctionnalités et lister celles considérées comme dangereuses ;

Preuves de faisabilité
– Pour chacune des fonctionnalités dangereuses (qu'elle soit standardisée ou propriétaire) supportée par chacun des moteurs, développer un *PoC* ;

Applications
– Pour chacun des moteurs, identifier des bibliothèques ou applications l'utilisant ;
– Vérifier si les fonctionnalités dangereuses du moteur sont accessibles via les applications (ce qui constituerait une vulnérabilité), en utilisant le *PoC* précédemment développé ;
– Tenter d'exploiter effectivement les applications identifiées comme vulnérables.

1.7 Cibles

Les moteurs XSLT les plus connus ont été incorporés au périmètre d'étude. Ceci inclut entre autres *libxslt*, *Xalan-C*, *Xalan-J*, *Saxon* et *MSXML*. D'autres moteurs, spécifiques à des composants communément déployés comme Firefox et Opera, ont eux aussi été incorporés.

Toutefois, de nombreux autres moteurs ont été identifiés mais non testés : *XL TXE-J* [1], *Sablotron* [2], *XT* [3], *AltovaXML* [4], *jsxml* [5], *ajaxslt* [6], ...

Parmi les applications étudiées, une attention particulière est portée aux environnements Web (navigateurs, CMS, serveurs d'applications), aux outils d'utilisateurs finaux (lecteur d'images ou de fichiers bureautiques) et à différentes fonctions de sécurité (signature électronique, SSO, ...)

1.8 Autres travaux sur le sujet

Au cours de cette étude, les publications suivantes ont été consultées. Elles montrent que le problème est connu depuis une dizaine d'années, mais que l'axe d'étude part généralement de l'application vers le moteur (approche descendante, dite *top-down*), ce que nous souhaitons justement inverser.

- Exécution de code dans Oracle XSQL servlet [7] (2001)
- Exécution de code dans l'appliance Google Search [8] (2005)
- Attaques sur XML Security [9,10,11] (2007)
- Exécution de code sous Java et .NET [12] (2009)
- Exécution de code sous PHP [13] (2009)

1.9 Etat d'avancement

La phase initiale de cette étude est terminée : l'identification des fonctionnalités standardisées potentiellement dangereuses et l'analyse fine des principaux moteurs ont été réalisées. A des fins d'exemple, quelques bibliothèques et applications intégrant ces moteurs XSLT ont été testées. Plusieurs vulnérabilités ont alors été mises à jour et remontées aux éditeurs, et certaines seront détaillées dans le présent document.

Pour autant, il reste forcément des dizaines d'applications vulnérables, que ce soit dans des produits diffusés à grande échelle ou au contraire très spécifiques à un métier ou une organisation. A vous, chers lecteurs, d'intégrer ce type de vulnérabilités à votre méthodologie existante ;-)

2 Introduction à XSLT

2.1 Définition

« *XSLT (eXtensible Stylesheet Language Transformations), défini au sein de la recommandation XSL du W3C, est un langage de transformation XML de type fonctionnel [...] L'objectif principal est la transformation d'un document XML vers un autre, ou un dialecte XML (XHTML, XSL-FO, HTML, etc.). Cependant, le langage XSLT permet aussi les transformations vers tout autre type de document, au format texte ou dans un format binaire [...]* » [14]

XSLT 1.0, associé à XPath, est un langage de programmation « Turing-complet ». Des preuves de cette caractéristique sont disponibles sur Internet [15].

2.2 Références normatives

XSLT est principalement défini par les normes officielles définissant les versions 1.0 [16] et 2.0 [18]. Il existe également une version 1.1 [17], qui n'est jamais entrée en vigueur (« Working Draft »).

2.3 Autres références

Un effort communautaire nommé EXSLT [19] cherche à homogénéiser des extensions aux normes existantes, afin de permettre l'utilisation de fonctions additionnelles de manière similaire sur tous les moteurs implémentant EXSLT. Actuellement, c'est plus de 70 fonctions qui sont définies, regroupées en modules comme *math*, *date*, *regexp* ...
De plus, chaque moteur peut implémenter ses propres extensions, permettant ainsi des utilisations très spécifiques (envoi par mail du résultat, écriture dans une base de données).

3 Fonctionnalités dangereuses

Les fonctionnalités dangereuses déduites des normes XSLT 1.0, 1.1 et 2.0 sont listées ci-dessous, ainsi que celles relatives à EXSLT. Les fonctionnalités propriétaires seront détaillées dans la section relative à chacun des moteurs.
Les paramètres `href`, `src`, `schema-location`, ... désignent des URI. Selon l'implémentation, une URI peut nativement décrire de nombreux types de ressources (préfixes *http*, *mailto*, *ldap*, *smb*, ...)

3.1 XSLT 1.0

Les fonctionnalités potentiellement dangereuses présentes dans XSLT 1.0 sont les suivantes :

- `document()`, qui permet de lire des fichiers XML
- `xsl:import` et `xsl:include` (paramètre `href`), qui permettent de lire des fichiers XSL
- `system-property`, qui peut être utilisé pour fuiter des informations :
  - `xsl:version` : la version de XSLT supportée
  - `xsl:vendor` : l'éditeur du moteur XSLT
  - `xsl:vendor-url` : l'URL identifiant l'éditeur
- `xsl:message`, qui sert à générer des messages (popup, logs)

3.2 XSLT 1.1

Pour mémoire, la version 1.1 de la norme XSLT est à l'état de « Working Draft » depuis 2001. Les travaux de normalisation ultérieurs ont été réalisés sur la version 2.0, publiée en tant que « Recommendation » en Janvier 2007. Il n'existe à notre connaissance aucun logiciel implémentant explicitement cette norme. Toutefois, elle a clairement influencé la norme XSLT 2.0 et les extensions propriétaires les plus courantes.

Les fonctionnalités potentiellement dangereuses présentes dans XSLT 1.1 sont celles déjà décrites pour XSLT 1.0, plus les suivantes :

- `xsl:document` (paramètre `href`), qui permet d'écrire des fichiers
- `xsl:script` (paramètres `language` et `src`), qui permet d'exécuter du script (Javascript, Java)

3.3 XSLT 2.0

Les fonctionnalités potentiellement dangereuses présentes dans XSLT 2.0 sont celles déjà décrites pour XSLT 1.0, plus les suivantes :

- `unparsed-text()`, qui permet de lire des fichiers non XML
- `xsl:import-schema` (paramètre `schema-location`), qui permet de lire des fichiers XSL
- `xsl:result-document` (paramètre `href`), qui permet d'écrire des fichiers

De plus, des champs supplémentaires ont été ajoutés à `system-property` (dont `xsl:product-name` et `xsl:product-version`)

3.4 EXSLT

EXSLT propose les extensions potentiellement dangereuses suivantes :
- `exsl:document` (paramètre `href`), qui permet d'écrire des fichiers
- `func:script` (paramètres `language` et `src`), qui permet d'exécuter du script (Javascript, Java)

3.5 Tableau récapitulatif

Les références XSLT 1.1 et XSLT 2.0 incluent l'ensemble des fonctionnalités potentiellement dangereuses identifiés dans XSLT 1.0. Par commodité, seules les différences avec XSLT 1.0 sont notées pour ces deux références.

	XSLT 1.0	XSLT 1.1	XSLT 2.0	EXSLT
Fuite d'information	`xsl:message` `system-property`	x	`system-property`	x
Accès en lecture	`document()` `xsl:include` `xsl:import`	x	`unparsed-text()` `xsl:import-schema`	x
Accès en écriture	x	`xsl:document`	`xsl:result-document`	`exsl:document`
Exécution de code	x	`xsl:script`	x	`func:script`

Table 1. Fonctionnalités dangereuses issues des différentes normes

3.6 Attaques de type « XEE » (XML External Entity)

Bien que non strictement liées à XSLT, ces attaques permettant l'accès en lecture à un fichier voire à un répertoire (cf. [20]) seront elles aussi incluses dans le périmètre de nos tests.

3.7 Note sur le terme « document »

Il ne faut pas confondre :
- la fonction XPath `document()`, disponible à partir de XSLT 1.0 et permettant d'accéder en lecture à des fichiers au format XML, et
- les instructions XSLT `xsl:document` (XSLT 1.1), `exsl:document` (EXSLT) et `xsl:document-result` (XSLT 2.0) permettant d'écrire dans un fichier le résultat d'une transformation

4 Moteurs XSLT

Les moteurs suivants ont été intégrés au périmètre de l'étude :

Nom	Version	URL
Xalan-C	1.10	http://xml.apache.org/xalan-c/
Xalan-J	2.7.1	http://xml.apache.org/xalan-j/
libxslt	1.1.26	http://xmlsoft.org/XSLT/
MSXML	6.0	http://msdn.microsoft.com/en-us/library/ms763742.aspx
Transformiix	1.9.2	http://www.mozilla.org/projects/xslt/
Presto	2.7.62	http://www.opera.com/docs/specs/presto27/
Saxon-B pour Java	9.0.0.4	http://saxon.sourceforge.net/

Table 2. Moteurs XSLT intégrés à l'étude

4.1 Xalan-C

Xalan-C est une bibilothèque écrite en C++ et maintenue par le projet Apache. Elle utilise Xerces-C++ pour le parcours des structures XML.

Version testée : 1.10
Ligne de commande : xalan -in test.xml -xsl test.xsl
Identification via *system-property* :
- Version : 1
- Vendor : Apache Software Foundation
- Vendor URL : http://xml.apache.org/xalan-c

Aucune extension dangereuse n'a été identifiée dans Xalan-C.

4.2 Xalan-J

Xalan-J est une bibilothèque écrite en Java et maintenue par le projet Apache. Elle implémente l'interface *javax.xml.transform* de l'API JAXP 1.3, qui permet de proposer aux programmes une interface indépendante du moteur XSLT sous-jacent.

Version testée : 2.7.1
Ligne de commande : `java org.apache.xalan.xslt.Process -in test.xml -xsl test.xsl`
Identification via `system-property` :
- Version : 1.0

– Vendor : Apache Software Foundation
– Vendor URL : http://xml.apache.org/xalan-j

Nom	Fonctionnalité	Espace de nom	Paramètres
write	Création de fichier	http://xml.apache.org/xalan/redirect	file
Méthodes Java	Exécution de code	[anything]/[objet Java]	N/A
Méthodes Java	Exécution de code	http://xml.apache.org/xalan/java	N/A
Méthodes Java	Exécution de code	http://xml.apache.org/xslt/java	N/A
checkEnvironment	Fuite d'information	http://xml.apache.org/xalan	N/A
new, query, ...	SQL	http://xml.apache.org/xalan/sql	N/A

Table 3. Extensions dangereuses de Xalan-J

4.3 libxslt

libxslt est une bibilothèque écrite en C et développée dans le cadre du projet Gnome. Elle s'appuie sur libxml2, autre bibilothèque développée par le même auteur, Daniel Veillard.

Version testée : 1.1.26
Ligne de commande : `xsltproc test.xsl test.xml`
Identification via `system-property` :
– Version : 1.0
– Vendor : libxslt
– Vendor URL : http://xmlsoft.org/XSLT/

Nom	Fonctionnalité	Espace de nom	Paramètres
output	Création de fichier	http://icl.com/saxon	href ou file
write	Création de fichier	org.apache.xalan.xslt.extensions.Redirect	href ou file
document	Création de fichier	http://www.jclark.com/xt	href
document	Création de fichier	http://www.w3.org/1999/XSL/Transform	href
document	Création de fichier	http://exslt.org/common	href

Table 4. Extensions dangereuses de libxslt

4.4 MSXML

MSXML est le moteur XML et XSLT développé par Microsoft. La version courante (v6, sortie en 2005) intègre plusieurs principes de « sécurisation par défaut », ce qui a posé problème lors de la migration depuis la version 3 [21].

Version testée : 6
Ligne de commande : `wscript.exe msxml.js in.xml in.xsl out.txt`
Identification via *system-property* :

- Version : 1
- Vendor : Microsoft
- Vendor URL : `http://www.microsoft.com`

Nom	Fonctionnalité	Espace de nom	Paramètres
system-property	Fuite d'information	`urn:schemas-microsoft-com:xslt`	`msxsl:version`
script	Exécution de code	`urn:schemas-microsoft-com:xslt`	`language`

Table 5. Extensions dangereuses de MSXML

L'exécution de scripts, l'accès à des documents locaux, la résolution des références externes, ... sont désactivés par défaut dans MSXML 6. Un document du MSDN récapitule les différentes options de sécurité et leurs valeurs par défaut selon la version du moteur [22].

4.5 Transformiix

Transformiix est le moteur XSLT intégré aux produits Mozilla, comme par exemple le navigateur Firefox.

Version testée : 1.9.2 (Firefox v3.6.13)
Ligne de commande : N/A
Identification via *system-property* :

- Version : 1
- Vendor : Transformiix
- Vendor URL : `http://www.mozilla.org/projects/xslt/`

Aucune extension dangereuse n'a été identifiée dans Transformiix.

4.6 Presto

Presto est le moteur de rendu (incluant HTML, XML et XSLT) intégré aux navigateurs Opera et Opera Mini.

Version testée : 2.7.62 (Opera 11.01)
Ligne de commande : N/A
Identification via `system-property` :

- Version : *vide*
- Vendor : Opera
- Vendor URL : http://www.opera.com/

Aucune extension dangereuse n'a été identifiée dans Presto.

4.7 Saxon-B

Saxon est un moteur XSLT supportant la norme XSLT 2.0, disponible en Java et .NET. Il existe sous forme Open-Source et commerciale. Avant la version 9.2, la version Open-Source était désignée par *B* (pour *Basic*) et la commerciale par *SA* (pour *Schema-Aware*). Depuis la version 9.2, la version Open-Source est *HE* (pour *Home Edition*), les versions commerciales étant *PE* et *EE* (respectivement *Professional Edition* et *Enterprise Edition*). Seule la version Java a été testée.

Version testée : 9.0.0.4 Basic for Java
Ligne de commande : `saxonb-xslt -xsl:in.xsl in.xml`
Identification via *system-property* :

- Version : 2.0
- Vendor : SAXON 9.0.0.4 from Saxonica
- Vendor URL : http://www.saxonica.com/

L'option `ALLOW_EXTERNAL_FUNCTIONS` (par défaut à 0) permet d'activer les fonctionnalités suivantes :

- l'écriture de fichiers via `xsl:result-document`
- l'utilisation de propriétés Java comme `os.version` ou `user.home` en tant que paramètre de `system-property`
- l'exécution de code Java, que ce soit via les extensions Saxon ou des appels directs

Nom	Fonctionnalité	Espace de nom	Paramètres
`xsl:result-document`	Création de fichiers	`http://www.w3.org/1999/XSL/Transform`	href
`system-property`	Fuite d'information	`http://www.w3.org/1999/XSL/Transform`	Propriétés Java
Méthodes Java	Exécution de code	`java:[objet Java]`	N/A
Méthodes Java	Exécution de code	`[anything]/[objet Java]`	N/A
Méthodes Java	Exécution de code	`[objet Java]`	N/A
Extensions Saxon	Utilitaires divers	`http://saxon.sf.net/`	N/A
Extensions Saxon	Utilitaires divers	`[anything]/net.sf.saxon.functions.Extensions`	N/A

Table 6. Extensions dangereuses de Saxon-B

5 Exemples de vulnérabilités

5.1 Webkit

Wekit est un moteur de rendu Web utilisé aussi bien dans des navigateurs classiques (Epiphany sous Linux, Safari sous Windows et Mac, Maxthon3 sous Windows) que sur des *smartphones* et autres gadgets comme l'iPhone, l'iPad et les nouveaux Blackberry ou dans des applications variées comme des clients de messagerie instantanée ou des lecteurs de flux RSS. Le moteur XSLT utilisé est libxslt.

Il était possible, en utilisant l'extension *output* ou un de ses alias, de créer des fichiers dont le nom et le contenu (au respect de l'UTF-8 près) est contrôlé par l'attaquant. Un correctif a été publié le 20 Février 2011 [23].

Chrome est le seul navigateur utilisant Webkit et n'étant pas impacté par cette vulnérabilité, en raison de sa *sandbox* bloquant l'accès au système de fichiers.

5.2 Liferay

Liferay est un CMS codé en Java et utilisant Xalan-J comme moteur XSLT. La version 6.0.6 datée de Mars 2011 corrige plusieurs vulnérabilités (LPS-13762, LPS-14726 et LPS-14927) liées à XSLT. Ces vulnérabilités, propres au composant *XSLT Portlet*, permettent d'accéder en lecture au système de fichiers (via une entité externe ou des URLs de type `file://`) ou d'exécuter du code Java arbitraire.

5.3 xmlsec

La bibliothèque *XMLSec* [24], écrite en C, implémente les principaux standards de sécurisation de documents XML, comme *XML Signature* ou *Exclusive Canonical XML*. Cette biblothèque intègre des fonctions de transformation XSLT, malgré la recommandation « *Best Practice 3 : Consider avoiding XSLT Transforms* » du document « *XML Signature Best Practices* » [25] du W3C. Le moteur XSLT employé est libxslt. Comme pour Webkit, il est possible de créer des fichiers arbitraires, ici lors d'une simple vérification de signature. Ceci est particulièrement impactant dans le cas de services de vérification librement accessibles sur Internet.

La version 1.2.17 publiée en Mars 2011 corrige le problème. L'identifiant CVE-2011-1425 a été attribué à cette vulnérabilité.

Tous les logiciels utilisant xmlsec (dont plusieurs solutions de sécurité de type SSO/-SAML ou signature XML) sont donc vulnérables s'ils n'utilisent pas une version durcie de xmlsec (c'est-à-dire compilée avec –without-libxslt **ou** restreignant explicitement les accès au système de fichiers via `xsltSetSecurityPrefs()` **ou** de version supérieure à 1.2.16).

5.4 PHP

La version 4 de PHP utilise le moteur XSLT Sablotron, qui n'a pas été intégré à l'étude. PHP 5 utilise libxslt, et il est possible de créer des fichiers arbitraires sur le serveur Web si le contenu XSL est fourni par l'attaquant.

De plus, si la méthode `XSLTProcessor::registerPHPFunctions()` [26] est utilisée côté serveur, il devient possible d'appeler du code PHP directement depuis un document XSLT.

5.5 *Framework* .NET

Sharepoint et DotNetNuke sont deux exemples d'applications utilisant le *framework* .NET. Une vulnérabilité dans le moteur XSLT intégré à .NET a été identifiée et signalée à Microsoft. Un correctif devrait être disponible d'ici Juillet.

6 Exploitation

6.1 Primitive « lecture de fichier »

L'accès en lecture à des fichiers éventuellement distants peut permettre de :

- voler des crédences NTLM (sous Windows)
- accéder des ports visibles depuis le moteur XSLT mais pas depuis l'attaquant
- réaliser des attaques de type CSRF
- fuiter le contenu des fichiers accédés, en partie ou totalité
- exploiter des vulnérabilités dans le code client (HTTP, FTP, SMTP, LDAP, SMB, ...) du moteur XSLT ou XML (exemple : `nanohttp.c` et `nanoftp.c` dans libxml2)

6.2 Primitive « écriture de fichier »

Utilisateur simple sous Unix La présentation faite lors de ShmooCon2011 par IBM X-Force et intitulée « *USB Autorun attacks against Linux* » [27] aborde de manière assez complète ces problématiques (dépôt dans `~/.config/autostart/`, exploitation de vulnérabilités dans les générateurs ou afficheurs d'imagettes, ...).

Utilisateur simple sous Windows Il est possible d'énumérer et d'infecter les périphériques amovibles connectés à l'ordinateur sans avoir à générer de contenu binaire (via `autorun.inf` et un script DOS ou VBScript). De la même façon, les répertoires de démarrage sont une cible intéressante.

Si les limitations liées au format du fichier de sortie (UTF-8 dans le cas de libxslt) sont levées, de nombreuses autres possibilités permettant une exécution de code quasi-instantanée sont offertes :

- dépôt de DLL dans le chemin de recherche (*Binary planting* [28])
- exploitation de failles locales impactant le mode « Prévisualisation » comme CVE-2009-0658 (Adobe Reader) ou CVE-2010-3970 (Windows Shell)

Utilisateur privilégié sous Unix Même dans le cas d'un format de sortie restreint, le placement dans le $PATH d'un script de même nom qu'un binaire sensible est aisé.

Utilisateur privilégié sous Windows La possibilité de créer des fichiers en tant qu'utilisateur privilégié permet d'exécuter automatiquement du code par l'utilisation de fichiers MOF, comme démontré par Stuxnet. Des exemples de code sont librement disponibles, en particulier `wbemexec.rb` dans Metasploit.

Serveur Web La création de fichiers arbitraires (que ce soit via PHP/libxslt, Java/Xalan-J, SAML/xmlsec/libxslt, ...) permet de revenir au scénario bien plus classique d'upload de fichier. Des scripts d'exploitation adaptés à ce second scénario sont publiquement disponibles pour PHP, JSP, ASP, CFM, ...

6.3 Primitive « exécution de code »

Les passerelles entre XSLT et les langages sous-jacents sont parfois contre-intuitives, en raison du passage d'un langage impératif (Java, PHP, ...) vers un langage fonctionnel (XSLT). Cela est principalement ressenti lors de l'utilisation de boucles ou de types « complexes » comme les tableaux.

Par ailleurs, les trois contextes d'exécution de code décrits dans ce document (PHP, Java, .NET) permettent facilement l'exécution de commandes arbitraires :
- Java : Runtime.getRuntime().exec()
- PHP : passthru() ou équivalent
- .NET : System.Diagnostics.Process.Start()

Il est à noter que la présence d'une version récente de *bash* permet l'obtention d'un *reverse-shell* sans outil additionnel [29].

7 Sécurisation

7.1 Réduction de la surface d'attaque

En terme de sécurisation d'une application, les premières choses à réaliser sont l'analyse des besoins fonctionnels et des menaces propres au contexte de déploiement. Ensuite, une minimalisation de la surface d'attaque doit être entreprise. Typiquement, une application de vérification de signature XML ne devrait pas permettre la réalisation de transformation XSLT.

7.2 Choix du moteur XSLT

Si la réalisation de transformations XSLT est nécessaire, le choix du moteur et de sa configuration est primordial. Presto, Transformiix et Xalan-C ne proposent pas de fonctionnalité dangereuse, MSXML et Saxon en proposent mais les désactivent par défaut, alors que libxslt et Xalan-J sont par défaut dans un mode peu sécurisé.

7.3 Utilisation des API de sécurisation

Les moteurs XSLT implémentant JAXP 1.3 (dont Xalan-J et XL TXE-J) supportent l'option `FEATURE_SECURE_PROCESSING` [30] qui permet entre autres de désactiver les transformations XSLT. Le moteur libxslt propose une fonctionnalité similaire [31] qui permet de configurer si le programme peut lire et/ou écrire des fichiers locaux et/ou distants.

8 Conclusion

Pour mémoire, l'hypothèse générale était la suivante : « *On s'attend à ce qu'une étude méthodique à approche ascendante permette de mettre en évidence des vulnérabilités à impact élevé dans différentes applications mettant en œuvre des moteurs XSLT.* »

Il semble que cette hypothèse a bien été validée par notre étude. Toutefois, il reste de nombreux axes de recherche qui mériteraient notre attention. On peut citer par exemple :

- les recomandations du W3C en ce qui concerne la présence de XSLT dans les signatures XML [25]
- la fusion au sein d'un même fichier SVG de contenus XML et XSLT [32], avec les problèmes de *Same Origin Policy* que cela peut poser
- la fourniture aux moteurs XSLT d'entrées volontairement mal-formées (*fuzzing*), par simple mutation de jeux de tests publiquement disponibles

Références

1. IBM : XL TXE-J reference information
   http://publib.boulder.ibm.com/infocenter/realtime/v2r0/topic/com.ibm.rt.doc.20/user/xml/xltxej_reference.html
2. Sablotron XSLT, DOM and XPath processor
   http://www.gingerall.org/sablotron.html
3. XT, a fast, free implementation of XSLT in java
   http://www.blnz.com/xt/index.html
4. AltovaXML - XSLT 1.0/2.0 Engine, XQuery Engine, XML Validator
   http://www.altova.com/altovaxml.html
5. jsxml : JavaScript XML/XSLT library
   http://jsxml.net/
6. ajaxslt : Pure javascript implementation of XSLT and XPath
   http://code.google.com/p/ajaxslt/
7. Georgi Guninski security advisory #34
   http://www.guninski.com/oraxsql.html
8. Google Search Appliance proxystylesheet XSLT Java Code Execution
   http://osvdb.org/20981
9. iSEC Partners : Command injection in XML Signatures & Encryption
   http://www.isecpartners.com/files/XMLDSIG_Command_Injection.pdf
10. iSEC Partners : Attacking XML Security
    http://www.isecpartners.com/files/iSEC_HILL_AttackingXMLSecurity_bh07.pdf

11. iSEC Partners : A taxonomy of attacks against XML Signature & Encryption
http://www.isecpartners.com/files/iSEC_HILL_AttackingXMLSecurity_Handout.pdf
12. XSLT Command Execution Exploit
http://labs.securitycompass.com/index.php/2009/09/18/
13. The hidden dangers of XSLTProcessor
http://www.acunetix.com/blog/web-security-zone/articles/the-hidden-dangers-of-xsltprocessor-remote-xsl-
14. Extensible Stylesheet Language Transformations
http://fr.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations
15. Universal Turing Machine in XSLT
http://www.unidex.com/turing/utm.htm
16. Référence XSLT 1.0
http://www.w3.org/TR/xslt
17. Référence XSLT 1.1
http://www.w3.org/TR/xslt11/
18. Référence XSLT 2.0
http://www.w3.org/TR/xslt20/
19. Référence EXSLT
http://www.exslt.org/
20. Kingcope : Attacking Server Side XML Parsers
http://www.exploit-db.com/download_pdf/16093/
21. Upgrading to MSXML 6.0
http://blogs.msdn.com/b/xmlteam/archive/2007/03/12/upgrading-to-msxml-6-0.aspx
22. DOM Properties that Have Security Implications
http://msdn.microsoft.com/en-us/library/ms761392.aspx
23. Webkit Trac : Tighten up access permissions by using libxslt API
http://trac.webkit.org/changeset/79159
24. XML Security Library
http://www.aleksey.com/xmlsec/
25. W3C : XML Signature Best Practices
http://www.w3.org/TR/xmldsig-bestpractices/
26. PHP5 `XSLTProcessor::registerPHPFunctions()`
http://php.net/manual/fr/xsltprocessor.registerphpfunctions.php
27. ShmooCon 2011 presentation : USB autorun attacks against Linux
http://blogs.iss.net/archive/Shmoocon2011.html
28. OWASP - Binary planting
http://www.owasp.org/index.php/Binary_planting
29. Connect-Back Shell (Literally)
http://labs.neohapsis.com/2008/04/17/connect-back-shell-literally/
30. JAXP : XML Constants
http://jaxp.java.net/docs/api/javax/xml/XMLConstants.html#FEATURE_SECURE_PROCESSING
31. The libxslt security framework
http://xmlsoft.org/XSLT/html/libxslt-security.htm
32. A harmless SVG + XSLT curiousity
http://scarybeastsecurity.blogspot.com/2011/01/harmless-svg-xslt-curiousity.html