



MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

L 19018 - 70 - F: 8,90 € - RD



70 NOVEMBRE/DÉCEMBRE 2013

France METRO : 8,90 € - CH : 15 CHF - BE/PORT CONT : 9,90 € - DOM TOM : 9,50 € - CAN : 16 \$ cad - Maroc : 110 MAD - Tunisie 19 TND

RÉSEAU **BGP**

BGP : quelle sécurité pour le protocole de routage au cœur d'Internet ?

p. 60



SYSTÈME **SIEM PRELUDE**

SIEM : retour d'expérience sur la mise en place de Prelude

p. 68



CODE **SHELLCODES**

Shellcodes polymorphiques : comment les détecter ?

p. 78



EXPLOIT CORNER

Epathobj : découvrez la faille exploitable de Windows NT à 8

p. 04



PENTEST CORNER

DominatorPro : l'outil indispensable pour traquer les « DOM based XSS »

p. 10



FORENSIC CORNER

Automatisez vos analyses forensics avec DFF et Python !

p. 18



DOSSIER

BIG DATA quand la taille compte !

p. 26

- 1 - Dimensionnez vos infrastructures pour le Big Data
- 2 - Détectez les menaces par analyse comportementale
- 3 - Big data et vie privée : entre expérimentations et confusions



6^{ème}
Forum International
de la Cybersécurité



21 et 22 janvier 2014
Lille Grand Palais - France

IDENTITÉ
NUMÉRIQUE ET
CONFIANCE

www.forum-fic.com/2014

contact@forum-fic.com



ÉDITO

YES WE SCAN !

Comme certainement beaucoup de lecteurs de MISC, j'ai suivi le feuilleton des péripéties estivales d'Edward Snowden avec beaucoup de gourmandise. Depuis la fin de la saga Harry Potter, jamais je n'avais eu plus hâte de découvrir les nouveaux épisodes d'une série.

Par un heureux hasard du calendrier, c'est pendant la conférence SSTIC, réunissant la fine fleur des experts français en sécurité, que la première bombe de Snowden, le programme PRISM, a été dévoilée. À cette occasion, ceux-ci, se réunissant comme chacun le sait non pas à Monaco mais à Rennes, se sont livrés aux analyses les plus éclairées sur la sécurité du Cloud. Parfois même à des heures très avancées de la nuit, ils sont allés jusqu'à sacrifier leur sommeil dans des rues mal éclairées de la capitale bretonne, et ce, afin d'échanger jusqu'à plus soif sur les conséquences de ces révélations.

Heureusement, les fournisseurs de services égratignés ont immédiatement réagi aux divagations du félon Snowden. Ils ont ainsi nié avec aplomb tout accès privilégié des services secrets américains aux si précieuses données de leurs utilisateurs. Tous les DSI venant d'externaliser leur messagerie ont donc pu retrouver le sommeil. Les projets de réponses aux appels d'offres internationaux reposaient en toute sécurité sur les serveurs de messagerie outre Atlantique, loin des oreilles indiscretes de leurs concurrents américains. Et si, dans le pire des cas, la NSA gardait une copie de toute leur messagerie, cela ferait un PRA à moindre coût. De toute façon, la NSA ne recherche rien d'autre que des terroristes, car, le renseignement étant un métier de seigneurs, il est bien éloigné des considérations économiques.

Mais, malheureusement pour le sommeil de nos DSI, le doute a fini par s'immiscer dans leur esprit lorsque Google et Microsoft ont réclamé au FISA (Foreign Intelligence Surveillance Act) de pouvoir cesser de mentir à leurs clients quant aux indiscretions de la NSA.

Selon le think-tank Itif (Information Technology & Innovation Fondation), les révélations de Snowden pourraient faire perdre 35 milliards de dollars à l'industrie du cloud US.

Cela constitue certainement une aubaine pour le cloud français, même si des esprits taquins font remarquer sur la liste de diffusion FRnOG [1] que le Cloud français c'est « du VMWare sur des serveurs Dell avec des processeurs Intel routés par Cisco et Huawei ».

Après un bref sursaut d'intérêt, lorsqu'il a été question d'espionnage de l'union européenne, le soufflet est finalement retombé, les dernières révélations n'agitant plus que le microcosme des informaticiens.

Pour le grand public, la vie privée semble décidément, pour paraphraser Jean-Marc Manach, un problème de vieux cons...

Pourtant les révélations suivantes ne manquent pas de sel, comme l'intégration de portes dérobées dans les primitives cryptographiques par la NSA. À ce propos, la lecture de l'article publié en 2007 par Bruce Schneier sur l'étrange histoire de PRNG Double-EC est un plaisir d'esthète qui se déguste comme un vin dont les années ont sublimé les arômes. Schneier s'y étonne de l'insistance de la NSA pour que ce générateur pseudo-aléatoire soit intégré aux standards du NIST :

« I don't understand why the NSA was so insistent about including Dual_EC_DRBG in the standard. It makes no sense as a trap door : It's public, and rather obvious. It makes no sense from an engineering perspective : It's too slow for anyone to willingly use it. »

Pour finir, je vous recommande l'excellent blog « Terrorismes, guérillas, stratégie et autres activités humaines » qui apporte un éclairage décalé sur le monde du renseignement et tout particulièrement ce billet sur l'affaire Snowden parfaitement à propos: « À force de jamais rien comprendre, un jour il va vous arriver des bricoles » [2].

Cedric Foll / @follc

[1] <http://www.frnog.org/>

[2] <http://aboutjaffar.blog.lemonde.fr/2013/07/01/imbeciles/>

Rendez-vous au 3 janvier 2014 pour le n°71 !

SOMMAIRE

EXPLOIT CORNER

[04-08] EPATHOBJ de NT4 à Windows 8 (CVE-2013-3660)

PENTEST CORNER

[10-16] Découverte de « DOM-based XSS » avec DOMinatorPro

FORENSIC CORNER

[18-25] Scriptez vos analyses forensiques avec Python et DFF

DOSSIER



BIG DATA QUAND LA TAILLE COMPTE !

[26] Préambule

[27-46] HPC, « Big Data » : de la théorie à la pratique

[47-53] Détection de menaces par analyse comportementale de l'activité réseau et des utilisateurs

[54-59] Big data : entre expérimentations et confusions

RÉSEAU

[60-66] Quelques pistes sur le renforcement de la sécurité autour du protocole de routage BGP

SYSTÈME

[68-76] Mise en place du SIEM Prelude en entreprise - Retour d'expérience

CODE

[78-82] Détection des shellcodes polymorphes

ABONNEMENT

[35/36/77] Bons d'abonnement

www.miscmag.com

MISC est édité par Les Éditions Diamond
B.P. 20142 / 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 - Fax : 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.miscmag.com
boutique.ed-diamond.com
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8,90 Euros

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Secrétaire de rédaction : Véronique Sittler
Conception graphique : Jérémie Gall
Responsable publicité : Black Mouse Communication - Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Illustrations : www.fotolia.com
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01



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

Charte de MISC

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

Nouveau !

Les abonnements numériques et les anciens numéros sont désormais disponibles sur :



en version PDF : numerique.ed-diamond.com



en version papier : boutique.ed-diamond.com

EPATHOBJ DE NT4 À WINDOWS 8 (CVE-2013-3660)

Florian LEDOUX - @myst3rie - florianledouxnord@gmail.com

mots-clés : NOYAU / EXPLOITATION / SMEP / WIN32K

En mars, Tavis Ormandi découvre un bug dans le noyau Windows. Sa particularité est qu'il impacte toutes les versions du système (de Windows 8 à NT4). Un mois plus tard, une élévation de privilèges basée sur ce bug est rendue publique. La vulnérabilité sera finalement corrigée en juillet.

1 Découverte du bug

De par sa localisation et sa portée, l'impact du bug est immense. En effet, le bug est localisé dans le driver win32k.sys, responsable du rendu graphique (Graphics Device Interface ou GDI) et plus précisément dans un mécanisme ancien, datant du noyau NT. Cette faille est restée cachée depuis 1990. Sa découverte est due au fuzzing de la GDI dans des conditions particulières, où la mémoire est saturée. **[TAVISO]**

Le mécanisme vulnérable est situé dans la gestion des objets graphiques nommés PATH. Le rôle d'un

PATH est de décrire une forme graphique, grâce à un ensemble de points, courbes et arcs. En mémoire, notre PATH est représenté par une liste doublement chaînée de tous ses composants (points, courbe, arcs) :

```
typedef struct _PATHRECORD {
    struct _PATHRECORD *pprnext;
    struct _PATHRECORD *pprprev;
    FLONG                flags;
    ULONG                count;
    POINTFIX              points[0];
} PATHRECORD, *PPATHRECORD;
```

Cette structure **PATHRECORD** peut contenir différents types de courbes mais elles seront toujours stockées en tant que courbes de Bézier. Ce type de courbe est décrit par une suite de triplets, un point de terminaison et deux points de contrôle (Fig. 1).

Le problème réside dans les fonctions qui gèrent l'aplanissement des courbes et leurs transformations en un ensemble de lignes droites (Fig. 2).

Pour effectuer l'aplanissement, la fonction **gdi32!FlattenPath()** est appelée depuis le mode utilisateur. Celle-ci fait indirectement appel en noyau à **win32k!EPATHOBJ::bFlatten()** qui parcourt la liste doublement chaînée de **PATHRECORD** à la recherche de points décrivant une courbe de Bézier. Si elle en rencontre, la fonction **win32k!EPATHOBJ::pprFlattenRec()** est alors appelée :

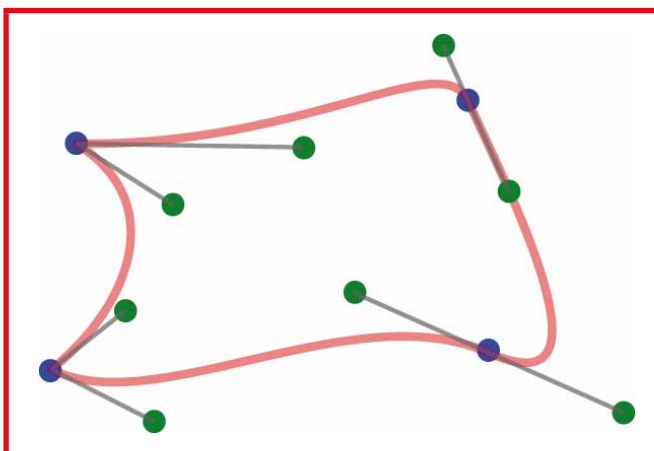


Fig. 1 : Courbe de Bézier

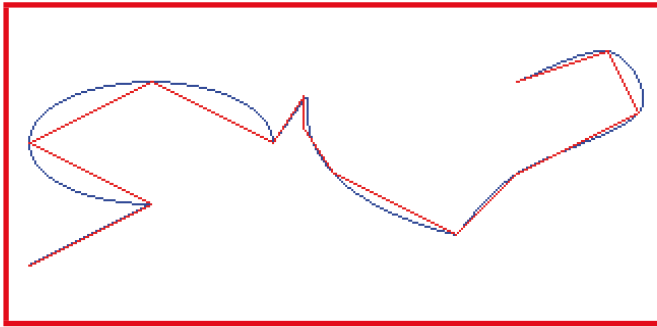


Fig. 2 : Aplatissement d'une courbe

```

BOOL __thiscall EPATHOBJ::bFlatten( EPATHOBJ *this )
{
    PATHOBJ *ppath;
    PATHRECORD *ppr;

    ppath = this->ppath;
    for( *ppr = ppath->pprfirst; ppr != NULL; ppr = ppr->pprnext )
    {
        if ( ppr->flags & PD_BEZIER )
        {
            ppr = EPATHOBJ::pprFlattenRec(this, ppr);
            if( !ppr )
                return FALSE;
        }
    }
    return TRUE;
}

```

Cette dernière fonction va vérifier le **PATHRECORD** donné en paramètre, et plus précisément son champ «points» qui est un tableau de structure **POINTFIX** :

```

typedef struct _POINTFIX
{
    ULONG x;
    ULONG y;
} POINTFIX, *PPOINTFIX;

```

Si l'aplanissement de la courbe décrite par notre **PATHRECORD** nécessite plus de 8 triplets de points supplémentaires, la fonction **win32k!EPATHOBJ::newpathrec()** aura la charge d'allouer une nouvelle structure par l'intermédiaire de la fonction **win32k!newpathalloc()**.

1.1 Le problème de PATHALLOC

Cette fonction **win32k!newpathalloc()** fait appel à l'allocateur **win32k!PATHALLOC**. Celui-ci gère une liste chaînée **win32k!PATHALLOC::freelist** ayant quatre entrées. Dans le cas où notre liste est vide,

il fait appel à **PALLOCMEM()** qui alloue et initialise à zéro les espaces mémoire alloués. Mais lorsqu'une entrée est libérée ou ré-allouée, sa mémoire n'est pas mise à zéro. Si par malchance le noyau récupère un de ces quatre espaces mémoire provenant de **win32k!PATHALLOC::freelist**, il aura la tâche d'initialiser cette mémoire avant utilisation.

1.2 Le problème de pprFlattenRec()

Hélas, dans certains cas notre fonction **win32k!EPATHHOBJ::pprFlattenRec()** n'initialise pas comme elle le devrait le **PATHRECORD** alloué par **win32k!PATHALLOC** :

```

PPATHREC EPATHOBJ::pprFlattenRec(PATHRECORD *ppr)
{
    PATHRECORD *pprNew;

    newpathrec(&pprNew);

    pprNew->pprprev = ppr->pprprev;
    pprNew->count = 0;
    pprNew->flags = (ppr->flags & ~PD_BEZIER);
    ...
}

```

Le champ **pprNew.next** n'est pas initialisé. Il peut donc contenir une valeur provenant d'un ancien **PATHRECORD** libéré. Il est à noter que, dans le cas où la mémoire est saturée, on peut bloquer l'allocation de nouveaux **PATHRECORD** et donc forcer le champ **next** du **PATHRECORD** en fin de chaîne à rester non initialisé.

2 Exploitation

2.1 Contrôle du PATHRECORD

Tout d'abord, il est nécessaire de saturer la mémoire d'objets graphiques, la fonction **CreateRoundRectRgn()** sera utilisée, celle-ci crée des rectangles de différentes formes. Ces formes seront dessinées sur un bureau alternatif afin d'être furtif et ce, grâce à **CreateDesktop()**.

Nous allons ensuite utiliser la fonction **BeginPath()** sur ce nouveau bureau. Cette dernière va libérer des **PATHRECORD**, nous laissant la possibilité d'insérer de nouveaux **PATHRECORD** dans **win32k!PATHALLOC::freelist** et ce, par l'intermédiaire de la fonction **PolyDraw()**, responsable de la création des courbes de Bézier :



```

BOOL PolyDraw(
    _In_ HDC hdc,
    _In_ const POINTFIX *points,
    _In_ const BYTE *pointTypes,
    _In_ int cCount
);

```

Les coordonnées des points ainsi dessinées correspondront à l'adresse d'une structure **PATHRECORD** créée en espace utilisateur. **PolyDraw()** va être appelé avec, à chaque fois, de moins en moins de points à dessiner. Cela aura pour effet de remplir la mémoire allouée dans notre **freelist** par des valeurs contrôlées. A terme, l'allocation de **PATHRECORD** échouera laissant le champ **next** du dernier **PATHRECORD** créé non initialisé.

Il ne s'agit pas ici d'une race condition comme il l'a très bien été expliqué sur le blog de VUPEN [VUPEN]. C'est uniquement, lorsque la fonction **win32k!EPATHOBJ::bFlatten()** est appelée que notre liste est parcourue. Nous pouvons donc libérer tous les objets graphiques qui saturent la mémoire, pour ensuite faire appel à la fonction **FlattenPath()** qui parcourra notre liste jusqu'à un **PATHRECORD** que nous contrôlons totalement.

2.2 Exécution de code

Les **PATHRECORD** vont être parcourus par **win32k!EPATHOBJ::bFlatten()** dont le code est disponible plus haut. La mémoire n'étant plus saturée, **win32k!EPATHOBJ::pprFlattenRec()** va manipuler le **PATHRECORD** que nous contrôlons. Les premières instructions de la fonction sont les suivantes :

```

PPATHREC EPATHOBJ::pprFlattenRec(PATHRECORD *ppr)
{
    ...
    pprNew->pprprev = ppr->pprprev;
    pprNew->count = 0;
    pprNew->flags = (ppr->flags & ~PD_BEZIER);

    pprNew->pprprev->pprnext = pprNew;
    ...
}

```

où **pprNew** est un **PATHRECORD** fraîchement alloué par **win32k!newpathalloc()** et **pprNew->pprprev** pointe sur notre structure **PATHRECORD** :

```

ExploitRecord.next = (PPATHRECORD)0x42424242;
ExploitRecord.prev = (PPATHRECORD)0x41414141;
ExploitRecord.flags = PD_BEZIER | PD_BEGINSUBPATH;
ExploitRecord.count = 4;

```

Nous sommes donc capables, de réécrire un pointeur en mémoire (en l'occurrence à l'adresse 0x41414141),

mais il sera remplacé par l'adresse de **pprNew** que nous ne contrôlons pas.

```

> dd 0x41414141 L1
41414141 c065a048

> dd c065a048
c065a048 00000000 42424242

```

Il est donc possible de réécrire un pointeur de fonction dans l'espace d'adressage du noyau. Lors de la redirection du flux d'exécution, notre champ **ExploitRecord.next** (ici 0x42424242) sera exécuté. Mais il faut savoir que cette valeur doit aussi pointer sur un **PATHRECORD** valide afin de terminer la chaîne. Dans le cas contraire, nous aurons un crash.

Comme nous exécutons du code en local sur la machine, l'ASLR (Address Space Layout Randomization) ne posera pas de problème. Nous allons récupérer un pointeur de fonction en noyau, plus exactement celui qui se situe juste après **nt!HalDispatchTable** (symbole exporté). Ainsi, lorsque nous appellerons **NtQueryIntervalProfile()** en mode utilisateur, le pointeur réécrit sera appelé et notre code sera exécuté. [RS]

Lors de l'exécution des 4 octets que nous contrôlons dans le noyau, le paramètre donné à **NtQueryIntervalProfile()** va se retrouver sur la pile. Ainsi si nous exécutons une instruction comme **JUMP [ESP+X]**, le code sera redirigé vers l'adresse donnée en paramètre. Nous avons donc déjà 3 octets pour former notre **JUMP [ESP+X]** qui sont **0xFFx64x24** le dernier octet représentera X (l'offset vers lequel on saute). C'est l'octet de poids fort de notre adresse, il devra donc être inférieur à 0x80 pour que l'adresse pointe en espace utilisateur vers notre structure **PATHRECORD**. La valeur de **ExploitRecord.next** sera donc **0x602464FF** :

```

> dd nt!HalDispatchTable+0x4 L1
811def28 c065a048

> dd c065a048
c065a048 00000000 602464ff

```

C'est une adresse valide en espace utilisateur et elle équivaut en assembleur à **JMP [ESP+60]**, (l'offset peut changer entre les versions de Windows). Nous allons donc être capables de sauter à l'adresse donnée en argument à **NtQueryIntervalProfile()**.

Il est à noter que nos 4 octets sont stockés dans un espace mémoire en lecture/écriture et ne sont pas exécutables. Malgré cela, les objets de la GDI en mode noyau sont alloués dans une mémoire non paginée et non protégée par le DEP (Data Execution Prevention) donc, l'exécution de code n'est pas bloquée.

À DÉCOUVRIR ACTUELLEMENT !

Toutes les bonnes pratiques pour faire connaître et exploiter au mieux...

VOTRE PROJET OPEN SOURCE !



Document est la propriété exclusive de Johann Locatelli (johann.locatelli@businessdecision.com) - 06 janvier 2016 à 09:44

 **GNU/LINUX
MAGAZINE
HORS-SÉRIE N°69**

DISPONIBLE CHEZ VOTRE MARCHAND
DE JOURNAUX ET EN LIGNE SUR :
boutique.ed-diamond.com



3 Contournement du SMEP

Le SMEP (Supervisor Mode Execution Protection) est une sécurité intégrée dans les nouveaux processeurs Intel. Son rôle est de bloquer l'exécution de code localisé dans des pages mémoires utilisateur (marquées par le bit User/Supervisor) si on est en mode privilégié (CPL<3). Pour se faire le CPU vérifiera à qui appartient le PTE (Page Table Entries) de la page exécutée et lèvera une exception #PF (Page Fault) en cas de problème. [INTEL]

Le SMEP est actif si le 20ème bit du registre CR4 est à 1. Dans notre cas, après que nos 4 octets en mode noyau ont été exécutés un saut vers notre code en espace utilisateur sera effectué. A partir de ce moment une exception sera levée par le SMEP bloquant la suite des hostilités.

Il existe plusieurs possibilités de contourner cette protection en fonction des versions de Windows. Le but est d'insérer des données contrôlées dans l'espace mémoire du noyau, puis de retrouver leurs adresses afin de rediriger le flux d'exécution vers celles-ci. Il suffira enfin de désactiver le SMEP grâce au code injecté, pour finalement rediriger le flux d'exécution en espace utilisateur.

3.1 Insertion de donnée dans le noyau

Grâce à l'API Windows il est possible de créer des objets (Events, Timers, Sections...) stockés par le noyau. Alors que le corps de l'objet est dans le noyau, certains champs sont modifiables depuis l'espace utilisateur via des appels systèmes. Un attaquant pourra donc influencer sur la mémoire du noyau de cette façon.

Dans le cas de Windows 7 un nouveau type d'objet voit le jour, appelé **UserApcReserve** ils sont alloués en mémoire non paginée. La fonction **NtQueueApcThreadEx()** permet leur création et 16 octets consécutifs peuvent y être contrôlés. [J00RU]

Pour Windows 8 la plupart des objets du noyau sont protégés. De nouveaux types de POOL font leur apparition, **NonPagedPoolNx** et **NonPagedPoolSessionNx** donnant la possibilité à une page mémoire d'être non paginée et non exécutable. Pour autant la version x86 garde une Session Pool paginée et marquée comme exécutable. Un objet de la GDI tel qu'une palette peut alors être utilisé pour contrôler une portion de la mémoire noyau. [PTSECURITY]

Il restera enfin à injecter le code de désactivation du SMEP :

```
MOV EAX, cr4
BTR EAX, 20
MOV CR4, EAX
JMP 0x00031337
```

3.2 Retrouver nos objets

L'architecture Windows permet de retrouver de nombreuses informations depuis l'espace utilisateur sans aucun droit spécifique. L'adresse base de module du noyau peut, par exemple être retrouvée avec **NtQuerySystemInformation()** ou **EnumDeviceDrivers()**. Dans le cas d'objets cela sera possible grâce à **NtQuerySystemInformation(SystemHandleInformation)**. Pour les objets graphiques appartenant à la GDI il est possible de passer par la **GdiSharedHandleTable** ou par les fuites de win32k ;). [RECON]

Conclusion

Le noyau Windows n'est pas exempt de bugs, bien qu'il soit de plus en plus sécurisé et audité, il est encore possible de trouver des failles restées cachées durant des dizaines d'années. Leur exploitation reste encore possible aujourd'hui. Malgré cela ce savoir-faire se raréfie. [TAVISOFD]

L'exploitation d'EPATHOBJ sur plate-forme x64 n'a pas été traitée ici, pour autant, elle est parfaitement possible. Microsoft a mis plusieurs mois pour corriger la faille exposant tous ses utilisateurs à une escalade de privilèges sur la plupart des versions de son système d'exploitation. ■

■ RÉFÉRENCES

[TAVISO] <http://blog.cmpxchg8b.com/2013/05/introduction-to-windows-kernel-security.html>

[VUPEN] http://www.vupen.com/blog/20130723.Advanced_Exploitation_Windows_Kernel_Win32k_EoP_MS13-053.php

[RS] http://www.reversemode.com/index.php?option=com_remository&Itemid=2&func=fileinfo&id=51

[INTEL] Intel Manual 3A, 4.6 Access Rights

[J00RU] <http://j00ru.vexillium.org/?p=783>

[PTSECURITY] <http://blog.ptsecurity.com/2012/09/intel-smep-overview-and-partial-bypass.html>

[RECON] <http://recon.cx/2013/schedule/events/26.html>

[TAVISOFD] <http://seclists.org/fulldisclosure/2013/May/111>



**POUR RENFORCER
LA SÉCURITÉ
DE VOTRE ENTREPRISE,
GLISSEZ-VOUS DANS
LA PEAU D'UN HACKER !**

FORMATIONS INTRUSIONS

**Cours SANS Institute
Certifications GIAC**



SEC 542

Tests d'intrusion applicatifs
et hacking éthique

SEC 560

Network Penetration Testing and
Ethical Hacking

SEC 660

Tests d'intrusion avancés, exploits,
hacking éthique

Dates et plan disponibles

Renseignements et inscriptions

par téléphone +33 (0) 141 409 700

ou par courriel à : formations@hsc.fr

www.hsc-formation.fr





DÉCOUVERTE DE « DOM-BASED XSS » AVEC DOMINATORPRO

Laurent Butti – laurent.butti@gmail.com

mots-clés : DOM-BASED XSS / DOMINATOR / SÉCURITÉ WEB

Dans le cadre des tests de sécurité en boîte noire sur les applications Web, les failles « DOM-based XSS » sont plus difficiles à découvrir que les classiques « Reflected XSS » et « Persistent XSS ». Cet article rappelle les principes de ce type de failles, et détaille le mode de fonctionnement d'un outil performant dans la découverte des « DOM-based XSS », DOMinatorPro développé par Stefano Di Paola.

1 Introduction

Les vulnérabilités Cross-Site Scripting (XSS) sont une classe de faille qui consiste à injecter du code JavaScript malveillant dans le contexte de navigation d'un navigateur Internet. Habituellement, nous distinguons deux principales types de failles XSS : les « Reflected » dont l'attaque est réalisée via une charge dans la requête HTTP qui est alors restituée immédiatement dans la réponse HTTP au navigateur et les « Stored » dont le stockage d'une charge par une première requête HTTP sera alors restituée dans un rendu ultérieur suite à la

navigation de l'Internaute (le cas classique est l'insertion de données dans une base de stockage, typiquement pour des forums).

À ce jour, de nombreuses publications affirment que les vulnérabilités de type XSS sont de loin les plus présentes dans les applications Web [STATSI, STATS2] (Figure 1).

Ceci est certainement dû à plusieurs facteurs :

- absence de mécanisme automatique d'encodage contextuel des données non de confiance lors de la restitution HTML (seuls quelques frameworks s'y risquent comme [NETTE] ;

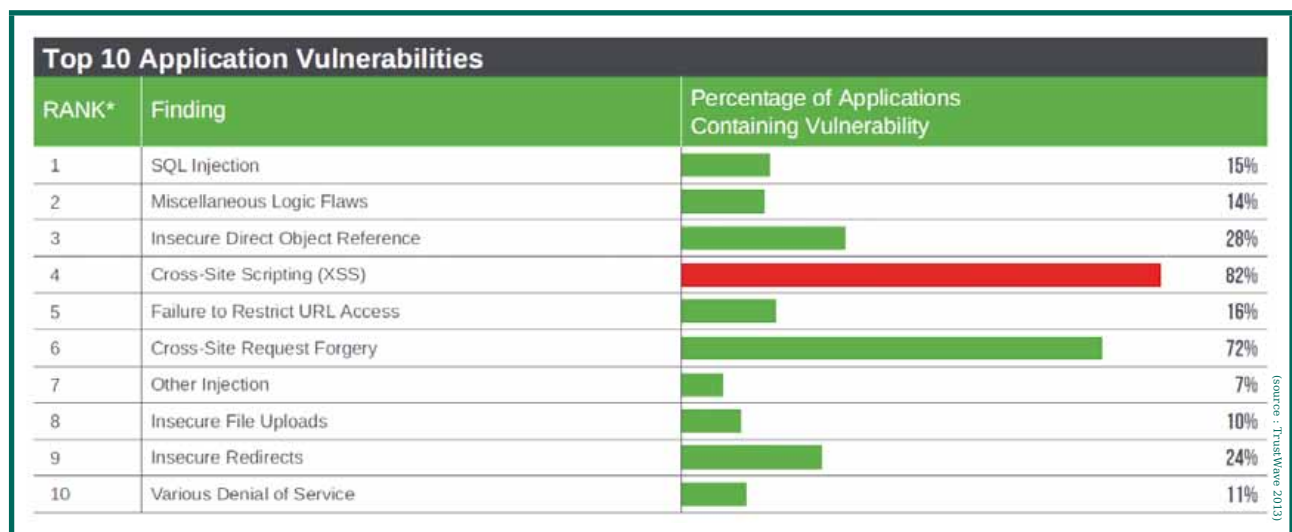


Figure 1 : Top 10 des vulnérabilités dans les applications Web



(Traditional) Stored XSS	DOM-Based Stored XSS (data from server)	'Pure' DOM-Based Stored XSS (data from HTML5 Local storage)
(Traditional) Reflected XSS	DOM-Based Reflected XSS (data from server)	'Pure' DOM-Based Reflected XSS (data from DOM)

(source : DaveWichersAspect Security)

Figure 2 : Résumé des types de failles XSS

- gravité des failles XSS souvent sous-estimées (et donc moins souvent corrigées) du fait que leurs impact ne soit pas aussi direct que ceux d'une faille d'injection SQL dont les effets potentiellement dévastateurs sont mieux compris.

Une publication d'Amit Klein, en 2005, a mis en évidence les failles de type « DOM-based XSS » **[KLEIN]**. Traditionnellement, les failles XSS ont lieu côté serveur où la réponse HTML rendue à l'utilisateur est réalisée par le code exécuté sur le serveur. Dans le cadre de « Reflected XSS », les données non de confiance sont restituées par le serveur directement ; et dans le cadre de « Persistent/Stored XSS », les données non de confiance sont restituées par le serveur indirectement (e.g. depuis un stockage dans une base de données).

Les traitements côté client dans le navigateur qui peuvent modifier le Document Object Model (DOM), en JavaScript, peuvent induire les mêmes problèmes lors de manipulation de données non de confiance : ce sont les « DOM-based XSS ». Là encore, nous aurons les « DOM-based XSS » des types Reflected et Persistent/Stored. La figure 2 résume l'ensemble de ces classes de failles XSS.

Avec l'avènement des nouvelles technologies du Web, la logique applicative est de plus en plus déportée vers le poste client, grâce au langage phare du Web qu'est le JavaScript. De part son ubiquité dans les navigateurs, il a fait l'objet de profondes avancées que ce soit au niveau des performances **[PERF]** qu'au niveau de son utilisation (nombreux frameworks de développement). Aujourd'hui, JavaScript est un point de passage obligé dans les technologies Web.

Par conséquent, les auditeurs sont de plus en plus souvent confrontés à des applications JavaScript exécutées par le navigateur, qui interrogent des Web Services (API), le plus souvent en REST, mais aussi les API du navigateur (par exemple celles exposées par les nouveaux standards HTML5 comme le stockage local, ou celles utilisables lors de développement d'extensions de navigateurs).

Rajoutons-y les frameworks JavaScript qui sont de plus en plus populaires (qui ont pour but de rendre

le développement Web aisé et rapide), le plus connu étant jQuery (représentant 25 % de part de marché des frameworks JavaScript sur les 10000 sites Web les plus importants selon **[TRENDS]**).

L'auditeur manuel devant réaliser un audit Web doit alors s'outiller pour éviter de sombrer dans les centaines de lignes de code JavaScript développées avec des frameworks JavaScript dont la logique est parfois difficile à appréhender.

Dans cet article, nous proposons de présenter les problématiques dans la recherche des failles « DOM-based XSS », les outils existants et les tests réalisés sur l'outil « DOMinatorPro » développé par Stefano Di Paola.

2 Problématiques

Lors d'audit d'applications Web en boîte noire, la partie « DOM-based XSS » est particulièrement difficile puisque nous n'avons pas forcément accès à du code JavaScript lisible (car minifié ou obfusqué par diverses techniques **[YUI, UGLIFYJS]**).

Lorsque cela n'est pas le cas, nous devons nous plonger à la recherche des points d'entrée (appelés « sources ») et des points de sortie (appelés « sinks »), mais aussi sur les éléments parcourus entre ces points d'entrée et ces points de sortie afin d'identifier si des traitements rendent inexploitable la potentielle faille de sécurité.

L'auditeur manuel doit donc arriver à :

- identifier l'ensemble des sources de données ;
- identifier l'ensemble des sinks de données ;
- tracer le chemin des sources vers les sinks.

Nous conseillons au lecteur de consulter un document très intéressant qui décrit l'ensemble des sources et sinks **[SINKS-SOURCES]**. Ce document est maintenu par l'auteur d'un outil Open Source d'analyse de code source JavaScript **[JSPRIME]**.

3 Les outils de découverte de DOM-based XSS

Dans ce contexte de difficulté d'audit manuel, l'auditeur peut s'outiller afin d'accélérer ses recherches.

Nous pouvons distinguer les techniques d'analyse suivantes :

- « boîte blanche » qui ont pour rôle de comprendre le code source (non minifié, non obfusqué), de modéliser les chemins possibles et d'en déduire grâce à une base de connaissance des « sources »

et des « sinks », la présence de vulnérabilités potentielles ;

- « boîte noire » qui ont pour rôle d'injecter des cas de tests dans les points d'entrée d'une application, d'observer le comportement puis alors d'en déduire la présence de vulnérabilités potentielles.

Dans la première catégorie, peu de candidats dans le monde Open Source, puisqu'à notre connaissance seul JsPrime semble pouvoir, à terme (car projet jeune), répondre partiellement (car limites intrinsèques à l'analyse de code source) à cette problématique. Dans le monde commercial, seules les sociétés HP et IBM avec respectivement les produits Fortify Source Code Analyzer et Rational AppScan Source Edition semblent supporter ce langage.

Dans la deuxième catégorie, la recherche de « DOM-based XSS » est intrinsèquement difficile pour des outils en boîte noire puisque l'exécution de JavaScript malveillant ne nécessite pas obligatoirement d'interaction avec le serveur.

Le web scanner Open Source w3af propose une brique de découverte de « DOM-based XSS » basée uniquement sur des expressions régulières qui recherchent les sinks et les sources, ce qui est nettement insuffisant pour arriver à des résultats convenables.

Dans la catégorie des proxies intrusifs, nous pouvons citer l'extension [BURPEXT], qui se base essentiellement sur des expressions régulières pour repérer des motifs suspects. L'outil, que nous n'avons pas testé, aura alors des difficultés intrinsèques pour éviter les faux positifs.

Dans la catégorie des outils dans les navigateurs, ce sont plus des outils d'assistance pour investigation que des outils dédiés à la recherche de failles DOM-based XSS. En effet, l'extension XSS-Rays (Chrome) de Gareth Heyes est par exemple utile dans les investigations manuelles mais ne se focalise pas sur les DOM-based XSS.

En conséquence, peu d'outils référents sur ce sujet !

4 DOMinatorPro

L'outil, développé par Stefano Di Paola, essaye de combler ce manque. Il repose sur des modifications apportées au moteur JavaScript de Mozilla Firefox (SpiderMonkey) afin d'ajouter un mécanisme de marquage (« taint »), de propagation de ces marquages et de traçage de ces marquages [DOMINATOR-SOURCES]. Grâce à la connaissance des sources et des sinks, l'outil est alors capable de tracer le parcours des données utilisateurs marquées et de remonter les failles potentielles lorsque des données marquées en entrée (sources) arrivent dans des

traitements en sortie (sinks). Cette technique est ainsi appelée par l'auteur : le « Realtime Dynamic Data Tainting ».

Sur la partie marquage et propagation, cela semble équivalent aux techniques utilisées par les outils d'analyse statique de code source. Cependant, l'aspect dynamique revêt un intérêt majeur, celui de ne pas pouvoir se tromper ! En effet, les outils d'analyse statique de code source doivent parser le code source, le comprendre et le modéliser, ce qui souvent présente des difficultés importantes comme nous pourrions le souligner dans la partie sur les exemples.

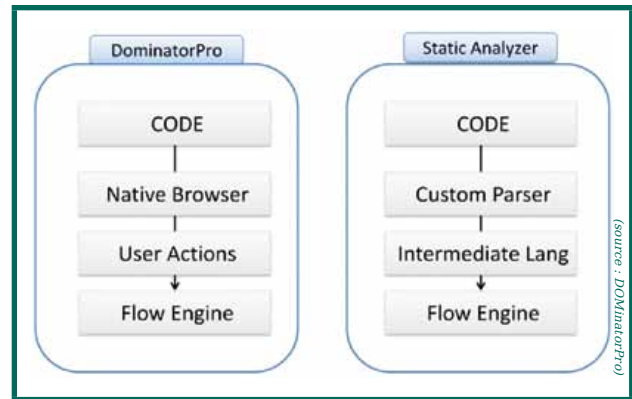


Figure 3 : Comparaison entre DOMinatorPro et un outil d'analyse de code source

L'outil est alors en théorie capable de découvrir tous les types de failles liées au contexte d'exécution du JavaScript dans le navigateur. Cela comprend l'exécution de JavaScript, l'injection HTML, la pollution de paramètres HTTP (HTTP Parameter Pollution)... et ce via de nombreux vecteurs (paramètres d'URL, en-têtes HTTP, cookies...).

La notification des failles est réalisée grâce à une extension intégrée avec Firebug qui notifie alors l'auditeur de failles potentielles. L'outil est donc composé de deux parties principales comme cela est présenté dans la figure 4.

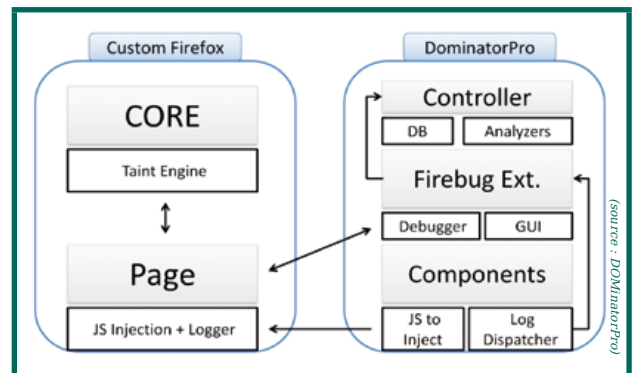


Figure 4 : Architecture DOMinatorPro

Les faits d'arme de DOMinatorPro font régulièrement l'objet de billets par son auteur [BLOG]. De nombreuses



vulnérabilités « DOM-based XSS » ont été découvertes grâce à cet outil, dont certaines sur les services de Google **[PLUSONE]**. Par exemple, cette dernière exploitait une faiblesse dans la validation d'une entrée utilisateur qui était réalisée via une liste noire de caractères non autorisés qui est malheureusement souvent contournable.

5 Exemples

Concentrons-nous maintenant sur quelques exemples qui auront pour but d'évaluer la pertinence de l'outil dans la découverte des « DOM-based XSS ». À noter que la plupart des exemples présentés dans cet article sont largement inspirés de la suite de cas de tests de l'outil JsPrime. Par ailleurs, ces exemples ne sont pas forcément représentatifs de cas réels, ce qui est souvent le cas dans des suites de tests désirant de tester les limites des outils.

La version testée est DOMinatorPro 0.9.2.1 sous Linux x64, dernière version à la date de rédaction de l'article.

```
<script>
function extract_location(obj) {
return obj['location'];
}
function extract_hash(obj) {
return obj['hash'];
}
document.write((function () {
return extract_hash(extract_location(document));
})()) // UNSAFE
</script>
```

Dans l'exemple ci-dessus, nous définissons une fonction anonyme qui va appeler de manière imbriquée deux autres fonctions, qui en résultat donnera alors **location.hash** qui est une source non de confiance utilisée dans un contexte de sink **document.write**. Cette vulnérabilité de type DOM-based XSS est découvrable avec DOMinatorPro.

```
<script>
var s_rev = 'hsah.noitacol.tnemucod(etirw.tnemucod';
var s_script = s_rev.split('').reverse().join('');
eval(s_script); // UNSAFE
</script>
```

L'exemple ci-dessus est très significatif d'un cas a priori non compréhensible en analyse statique de code source (puisque pas de notion d'exécution), mais qui devrait être réalisable en analyse dynamique puisque DOMinatorPro y arrive sans difficultés.

```
<div id="results"></div>
<script>
div = document.getElementById('results');
```

```
quora = {
  zebra: "text",
  yahoo: function () {
    this.benz = this.zebra;
  },
  benz: div.innerHTML
};

quora.zebra = location.hash.split('#')[1];
quora.yahoo();
div.innerHTML = quora.benz; // UNSAFE
</script>
```

Dans l'exemple ci-dessus, il est découvert correctement par DOMinatorPro puisque l'attribution du **location.hash.split('#')[1]** est faite dans la variable locale **benz** suite à l'appel de **quora.yahoo()**.

```
<div id="results"></div>
<script>
div = document.getElementById('results');
quora = {
  zebra: "text",
  yahoo: function () {
    this.benz = this.zebra;
  },
  benz: div.innerHTML
};

quora.zebra = location.hash.split('#')[1];
//quora.yahoo();
div.innerHTML = quora.benz; // SAFE
</script>
```

Dans l'exemple ci-dessus, DOMinatorPro déjoue facilement le piège puisqu'il n'y a pas d'appel à **quora.yahoo()**, par conséquent **quora.benz** ne présente pas de vulnérabilité. Un outil d'analyse statique pourrait quant à lui tomber dans le piège.

```
<script>
function function_1() {
  var x = 5;
  document.write(x); // SAFE
}
function function_2() {
  document.write(x);
}
function function_3() {
  var x = document.location.hash.split('#')[1];
}

function function_4() {
  x = document.location.hash.split('#')[1];
}

function_1(); // SAFE
function_4(); // SAFE
function_2(); // UNSAFE
document.write(x); // UNSAFE
```

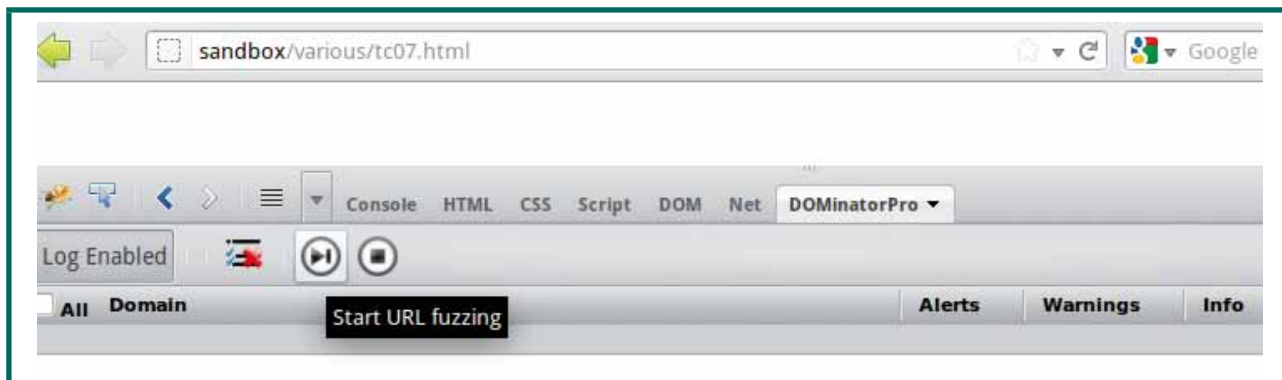



Figure 5 : Connexion à la page Web vulnérable

```
function_1(); // SAFE
function_3(); // SAFE
document.write(x); // UNSAFE
</script>
```

Dans l'exemple ci-dessus, nous avons souhaité illustrer les problématiques de compréhension des variables globales et locales, problématique récurrente pour les analyseurs de code source. Pour DOMinatorPro, il n'en est rien, grâce à l'analyse dynamique qui en est faite.

```
<script>
var escaped = encodeURIComponent(document.location.hash.split('#')
[1]);
document.write(escaped); // SAFE
</script>
```

Dans l'exemple ci-dessus, nous avons souhaité vérifier si l'outil était capable de comprendre la neutralisation des données non de confiance (la source) dans le contexte d'utilisation (le sink). DOMinatorPro ne lève pas de faux positif sur ce cas de test.

```
<script>
var escaped = encodeURIComponent(document.location.hash.split('#')
[1]);
document.write(decodeURIComponent(escaped)); // UNSAFE
</script>
```

Dans l'exemple ci-dessus, nous avons souhaité vérifier si l'outil était capable de comprendre que des données neutralisées (ici via `encodeURIComponent()`) ne l'étaient plus si un appel à la fonction inverse était réalisée (ici via `decodeURIComponent()`). DOMinatorPro lève correctement le vrai positif sur ce cas de test. Les outils d'analyse statique peuvent tomber dans le piège en considérant la donnée comme de confiance une fois passée dans `encodeURIComponent()`.

```
<script src="jquery.js">
</script>
<script>
```

```
var hash = document.location.hash.substr(1);
$(hash); // $() is a sink
jQuery(hash); // jQuery() is a sink
</script>
```

Dans l'exemple ci-dessus, un test trivial avec le framework jQuery et ses célèbres sinks que sont `$()` et `jQuery()`, DOMinatorPro les découvre aussi.

```
var outerValue = 'ninja';
var later;

function outerFunction() {
    var innerValue = 'samurai';

    function innerFunction() {
        var a = location.hash.split('#')[1];
        eval(a);
    }
    later = innerFunction;
}

outerFunction();

later(); // UNSAFE
```

L'exemple ci-dessus est périlleux pour un auditeur manuel qui doit pouvoir distinguer le fait que la variable globale `later` est redéfinie globalement via l'appel à la fonction `outerFunction()`, variable `later` qui alors référence la fonction `innerFunction()`, qui sera appelée via `later()`. Pénible pour un auditeur manuel, imaginons alors pour un outil devant modéliser le code source... Là encore, DOMinatorPro se tire de ce piège.

6 Détails sur la GUI DominatorPro

Dans le chapitre précédent, nous avons présenté les résultats sur quelques cas de tests JavaScript sans aborder l'aspect pratique et lisibilité des alertes remontées par DOMinatorPro.

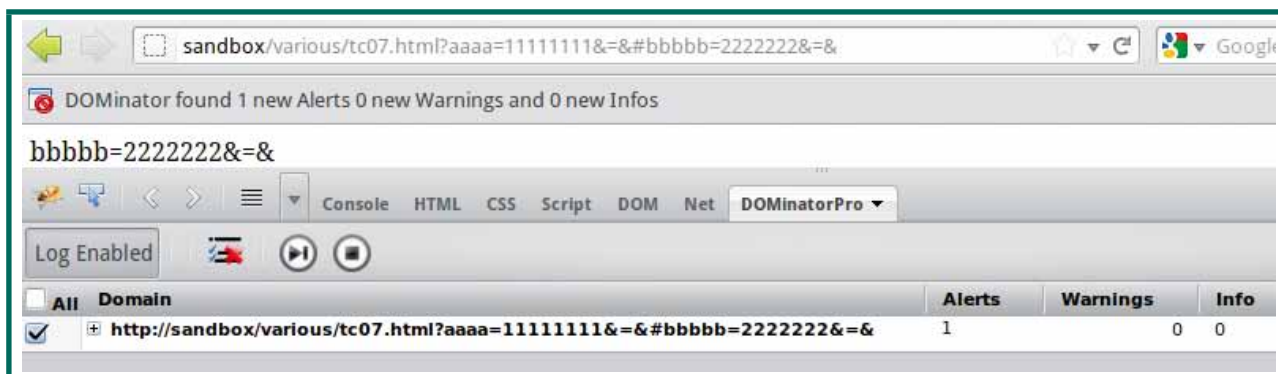


Figure 6 : Exécution de la fonctionnalité « fuzz » de l'outil

Présentons quelques captures d'écran relatives à la découverte d'une « DOM-based XSS » sur le cas de test suivant :

```
<div id="results"></div>
<script>
var param = document.location.hash.split("#")[1];
if (param) {
  var d = document.createElement('div');
  d.innerHTML = param; // UNSAFE
  if (document.body != null) {
    document.body.appendChild(d);
  }
}
</script>
```

Voir figure 5.

Lors de la première connexion à la page Web vulnérable, le module DOMinatorPro dans Firebug n'a pas averti d'un quelconque problème. En effet, il n'a pu voir que le vecteur de l'attaque était dans l'URL. À ce moment-là, soit il est possible de tester « à la main » quelques URLs en fonction des points d'entrée, soit nous pouvons utiliser le bouton « fuzz » de DOMinatorPro qui se chargera d'essayer quelques heuristiques de points d'entrée (Figure 6).

Nous constatons alors que l'outil a identifié avec le niveau « Alert » la faille de sécurité, niveau le plus élevé qui témoigne d'une forte confiance dans l'analyse (Figure 7).

L'onglet « Source History » apporte des informations importantes qui seront utilisées lors de l'analyse de la propagation de marquage. En effet, une cause courante des faux positifs est l'utilisation d'expressions régulières qui valident l'entrée non de confiance, or l'outil ne peut que difficilement présupposer de la pertinence de l'expression régulière, tâche dévolue à l'auditeur manuel. Et oui, il reste encore un peu de boulot...

7 Quelques limites et évolutions futures

Les limites de l'outil sont essentiellement atteintes dans un contexte d'audits récurrents dans un cycle de vie logiciel. En effet, dans le cadre d'automatisation de tests de sécurité, il est nécessaire que l'outil soit entièrement autonome et que les faux positifs,

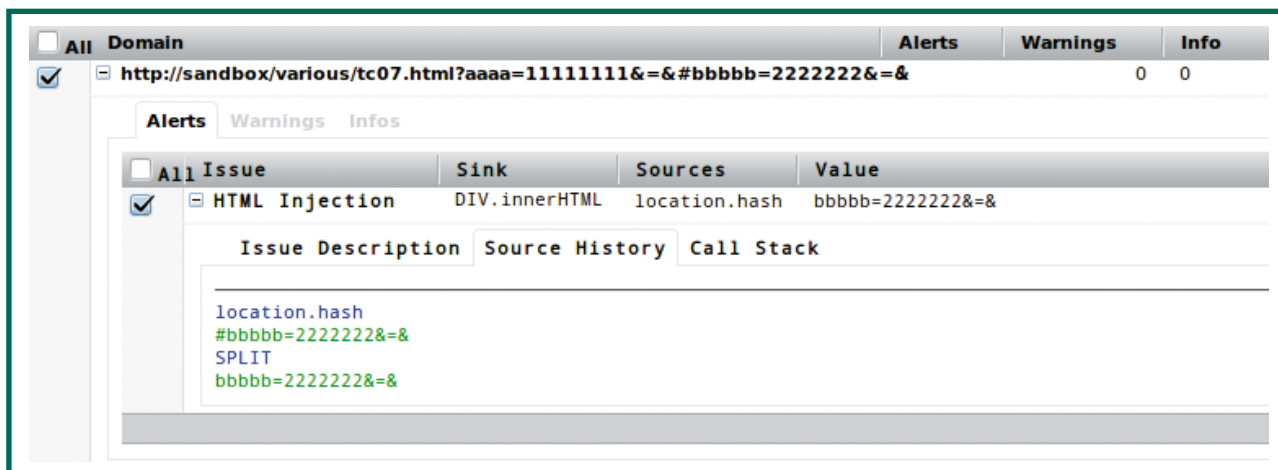


Figure 7 : Activation de l'onglet « Source History »

une fois identifiés de manière unique, ne soient plus remontés dans les analyses ultérieures. Ceci est particulièrement vrai lorsque de nombreux hôtes partagent du code JavaScript (éléments communs) ce qui est souvent le cas pour des problématiques d'optimisation de la performance de chargement des pages ou de rationalisation des développements.

Parmi quelques autres limites, nous pouvons citer :

- pas de mécanisme de crawling et d'automatisation ;
- pas de possibilité de ne rapporter que certaines catégories de failles ;
- pas d'outil de génération de preuve de concept à la volée ;
- plutôt lent à l'exécution, ainsi que parfois quelques plantages ;
- obligation d'exécuter une version modifiée de Mozilla Firefox.

Ce dernier point va peut-être évoluer à terme, puisque chez Mozilla, les équipes de développement de SpiderMonkey se sont basés sur les principes de DOMinatorPro afin d'intégrer les mêmes préceptes directement dans le moteur JavaScript **[SPIDERMONKEY]**. L'objectif est de disposer de cette technique ensuite en standard, ce qui la rendra alors accessible via les outils développeurs classiques **[THREAD, DEVTOOLS]**.

La version Entreprise de DOMinatorPro semble prendre en compte certaines des limites présentées dans cet article, cependant, nous ne l'avons pas testée.

Conclusion

Cet article ressemble à une apologie de l'outil DOMinatorPro pour la recherche de failles « DOM-based XSS ». Bien qu'imparfait, il est à ce jour un outil indispensable dans l'arsenal de l'auditeur humain sur les technologies Web. Les cas des tests présentés dans cet article en témoignent. DOMinator est le meilleur outil connu dans ce domaine de recherche manuelle de DOM-based XSS, et réalise parfaitement la tâche à laquelle il est dévolu : assister les auditeurs manuels dans la recherche de ce type de failles.

Cependant, il n'est aujourd'hui pas du tout adapté dans un contexte d'entreprise où l'automatisation des tests de sécurité est un prérequis pour couvrir un maximum de vulnérabilités sur un maximum d'applications exposées sur Internet. La version Entreprise de DOMinatorPro semble prendre en compte certaines des limites présentées dans cet article, cependant, nous ne l'avons pas testée. Les évolutions prévues dans le moteur JavaScript de Mozilla Firefox

(SpiderMonkey) pourront éventuellement changer la donne grâce à de nouveaux outils qui pourront se reposer dessus, toujours est-il que le principe d'analyse dynamique développée dans DOMinatorPro semble être la meilleure approche à ce jour afin d'optimiser l'efficacité d'un audit de sécurité sur la partie « DOM-based XSS ». ■

■ REMERCIEMENTS

Les plus vifs remerciements à Thomas Chopitea et Benjamin Caillat pour leurs relectures.

■ RÉFÉRENCES

[BLOG] <http://blog.mindedsecurity.com/>

[BURPEXT] <https://www.codemagi.com/downloads/dom-xss-scanner-checks>

[DEVTOOLS] <https://developer.mozilla.org/en-US/docs/Tools>

[DOMINATOR-SOURCES] <https://github.com/wisec/DOMinator>

[KLEIN] <http://www.webappsec.org/projects/articles/071105.shtml>

[JSPRIME] <https://github.com/dpnishant/jsprime/>

[NETTE] <http://doc.nette.org/en/templating>

[PERF] http://en.wikipedia.org/wiki/JavaScript_engine#Performance_evolution

[SINKS-SOURCES] <http://goo.gl/olzYM4>

[OWASP] https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet

[SPIDERMONKEY] https://github.com/alagenchev/spider_monkey

[STATS1] https://www.whitehatsec.com/assets/WPstats_summer12_12th.pdf

[STATS2] <https://www2.trustwave.com/2013GSR.html>

[THREAD] <https://groups.google.com/forum/#!msg/mozilla.dev.tech.js-engine.internals/dQ9-cEltaCk/mXTkLAPXIMoJ>

[TRENDS] <http://trends.builtwith.com/JavaScript>

[UGLIFYJS] <https://github.com/mishoo/UglifyJS2>

[YUI] <http://yui.github.io/yuicompressor/>

création agence Comelink - cr@lephotoconcept.com - 06 janvier 2016 à 09:44



**AJOUTEZ
LES NOUVELLES MÉTHODES
DE DURCISSEMENT
SYSTÈME À VOTRE
ARSENAL.**

FORMATIONS SÉCURISATION

**Cours SANS Institute
Certifications GIAC**



SEC 505
Sécuriser Windows

SEC 506
Sécuriser Unix & Linux

DEV 522
Durcissement des applications Web

Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr

www.hsc-formation.fr



SCRIPTEZ VOS ANALYSES FORENSIQUES AVEC PYTHON ET DFF

Frédéric Baguelin – frederic.baguelin@arxsys.fr - @udgover

mots-clés : PYTHON / FORENSICS / SCRIPTING / AUTOMATISATION / MÉTADONNÉES / ANALYSE / RECHERCHES

DFF (*Digital Forensics Framework*) est un outil permettant d'effectuer des analyses forensiques à partir de périphériques, images de disques ou fichiers. En plus des reconstructions des différentes couches logiques (volumes, système de fichiers) il est possible d'ajouter des fonctionnalités et d'automatiser des tâches en Python.

1 Présentation

L'outil DFF [1] est issu d'un projet de fin d'études qui fait suite au constat que dans le domaine de l'inforensique il existe soit des outils propriétaires (et souvent onéreux) soit une pléthore d'outils Open Source très spécialisés. L'objectif était donc de fournir un environnement réunissant différentes fonctionnalités au sein d'un outil unique et accessible à tous. Après leurs études, les créateurs du projet ont souhaité continuer l'aventure en créant leur société : ArxSys [2], en 2009, également l'année de la première version officielle. DFF fonctionne aussi bien sur les systèmes Unix(-like) que Windows en 32 et 64 bits. Parmi les fonctionnalités proposées, DFF supporte en entrée différents formats, de la classique image acquise avec **dd** aux conteneurs forensiques mais aussi directement sur des périphériques, c'est à dire sans acquisition préalable. A partir de ces entrées, il devient possible de remonter les partitions puis les systèmes de fichiers tout en prenant en compte les fichiers supprimés et les différentes zones non allouées. Enfin, l'application de différents filtres ou modules de visualisation permet de réduire la surface d'analyse pour faire ressortir les éléments probants. Le code source du projet, sous licence GPLv2, est disponible sur le dépôt **git** du projet [3] et des paquets et installateurs sont également fournis pour Windows et les distributions basées sur Debian. A savoir également que **cert.org** fournit des packages pour Fedora et CentOS/RHEL [4]. Les exemples qui

suivent sont basés sur la branche *develop* du dépôt Git. Cet article a pour but de vous présenter l'utilisation de l'API de DFF au travers de scripts exécutés au lancement ou depuis l'interpréteur Python.

1.1 Architecture générale

Depuis le début, L'architecture de DFF a été pensée de façon modulaire et orientée objet, laissant la possibilité de charger au démarrage ou à chaud de nouveaux modules développés en Python ou en C++. Pour ce faire, DFF repose sur une API fournissant différentes fonctionnalités allant de la gestion du *mapping* des fichiers reconstruits au moteur de recherche intégré, le tout de manière multithreadée. Les interfaces utilisateurs (console et graphique) utilisent cette API pour interagir avec le *framework* et fournir les informations aux utilisateurs. D'un point de vue général, l'architecture est présentée dans le schéma suivant :

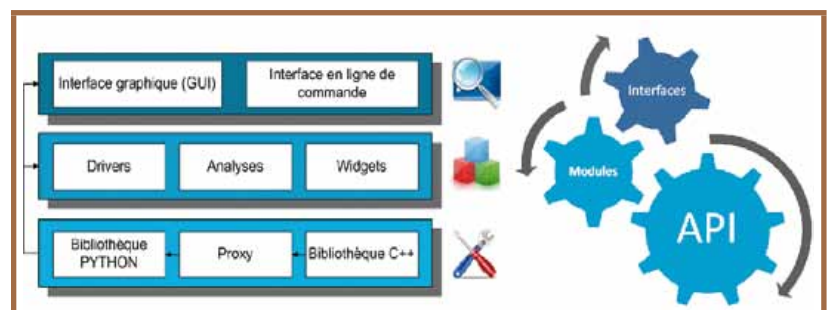


Fig. 1 : Architecture générale de DFF

1.2 API

L'API de DFF s'est enrichie depuis ses premières versions. Elle est massivement utilisée par les modules ainsi que les interfaces utilisateurs. Elle fournit les fonctionnalités nécessaires au chargement des modules ainsi qu'à leur ordonnancement. Une partie de l'API fournit également une gestion générique des types couramment utilisés (entier, chaîne de caractères, booléen). Ceci permet de travailler avec un objet générique, appelé **Variant**, encapsulant les données en entrée et en sortie pour les modules. La brique fondamentale de l'API est le système de fichiers virtuels, abrégé VFS (*Virtual File System*). Accessible globalement, il est en charge du maintien de l'arborescence des différentes couches de reconstruction. Les modules de reconstruction (appelés *drivers*) vont créer des nœuds (**Nodes**) et enregistrer leur point de montage au sein du VFS qui est empilable signifiant qu'il est possible à partir d'un **Node** donné de créer une nouvelle arborescence.

1.3 Modules

Il existe différents types de modules dans DFF : les modules de type FSO (*File System Object*) qui vont générer une arborescence au sein du VFS. Il y a également les modules d'analyse prenant en entrée un ou plusieurs nœuds du VFS afin d'extraire et de rendre accessibles par le biais d'attributs des métadonnées comme, par exemple, le module EXIF. Enfin, il y a les modules fournissant une vue graphique tels que les modules de visualisation (image, texte, PDF, et autres).

1.4 Interfaces utilisateurs

Il est possible d'interagir par le biais de 3 interfaces avec DFF :

- l'interpréteur Python ;
- l'interface console ;
- l'interface graphique.

Il est possible de travailler avec DFF uniquement à travers l'utilisation de l'interpréteur Python. Cette interface est utile lorsque l'on souhaite tester le développement de modules mais également pour l'automatisation.

L'interface en mode console peut être utile lorsqu'il est nécessaire de conduire une analyse avec un environnement dénué d'interface graphique. Cette interface est l'équivalent d'un *shell* classique avec gestion de la complétion sur les chemins de fichiers mais également sur les paramètres des modules.

L'interface graphique quant à elle facilite la visualisation des différents fichiers lors d'une analyse mais aussi

l'accès à certaines fonctionnalités du framework, telles que les requêtes prédéfinies du moteur de recherche ou la création de *timeline*. L'interface graphique embarque également l'interpréteur Python et la console.

2 Premiers pas avec DFF

Il est possible de démarrer DFF dans les trois modes d'interface précédemment cités. Depuis l'interpréteur Python les clauses **from ... import ...** seront utilisées comme par exemple **from dff.ui.console.console import Console** pour accéder à la classe gérant l'interface console. Pour lancer l'interface en mode console ou graphique, on utilisera respectivement **dff** et **dff-gui**.

Note

Il est possible de basculer de l'interface console vers l'interpréteur Python et vice-versa si Python est exécuté en mode « Inspection ». Pour activer cette fonctionnalité, soit vous invoquez le binaire python avec l'option **-i** suivie du script que vous souhaitez exécuter, soit vous déclarez la variable d'environnement **PYTHONINSPECT**. Ensuite, pour basculer de la console vers Python, il faut exécuter la commande **exit**. Pour revenir à l'interface console de l'interpréteur Python, il suffit d'exécuter la fonction **fg()**. Grâce à cette manipulation vous pouvez aussi lancer l'interface graphique après avoir démarré en console en exécutant **GUI().launch()** depuis l'interpréteur Python.

2.1 Interface console

DFF fournit une interface console permettant de naviguer dans l'arborescence ainsi que d'appliquer des actions, représentées par des modules, sur les éléments du système de fichiers virtuel. Comme toute console, la complétion est utilisable par l'utilisation de la touche tabulation. Si vous utilisez la complétion

```
#####
# Welcome on Digital Forensics Framework #
#####
dff / >
Module      : cut                merge                spare                split
Search      : carvergui
Statistics  : filesshort      timeline
Builtins    : batch            carver              carverui             cd
              fg                fileinfo            find                 history
              info            jobs                link                 load
              ls                 nan                 open                 show_cwd
Hash        : hash
Volatile memory : evclatality
Mailbox     : exchange
Connectors  : dff                devices             euf                  local
File systems : extfs            fatfs               ntfs
Volumes     : partition       vmware
Databases   : sqllitedb        winreg
Export      : extract        fuse
Viewers     : PDF viewer        Registry viewer     diff                 disassemble
              hexadecimal       pictures            player              sqlliteanalyzer
              textviewer        thumbnailvideo     sds
Metadata    : compound        link                actsexif             prefetch
```

Fig 2 : Complétion sur les modules

à partir d'une invite de commande vide, les modules chargés seront présentés par catégories (fig.2). La complétion fonctionne également pour les arguments d'un module donné en précisant s'ils sont obligatoires ou optionnels (fig. 3). Si vous devez spécifier un argument contenant des espaces, il est nécessaire de les échapper avec un antislash. Enfin, si un argument nécessite plusieurs paramètres, il suffit de les séparer par une virgule.

```
dff / > mvolatility zeus.vmem
optional: hdd_base profile
dff / > mvolatility zeus.vmem --profile
predefined: VistaSP0x64 VistaSP0x86 VistaSP1x64
             VistaSP1x86 VistaSP2x64 VistaSP2x86
             Min2003SP0x86 Min2003SP1x64 Min2003SP1x86
             Min2003SP2x64 Min2003SP2x86 Min2008R2SP0x64
             Min2008R2SP1x64 Min2008SP1x64 Min2008SP1x86
             Min2008SP2x64 Min2008SP2x86 Min7SP0x64
             Min7SP0x86 Min7SP1x64 Min7SP1x86
             MinXPSP1x64 MinXPSP2x64 MinXPSP2x86
             MinXPSP3x86
dff / > mvolatility zeus.vmem --profile Min7
predefined: Min7SP0x64 Min7SP0x86 Min7SP1x64 Min7SP1x86
dff / > mvolatility zeus.vmem --profile Min7
```

Fig 3 : Complétion des arguments d'un module

2.2 Interface graphique

Détailler toutes les fonctionnalités fournies par l'interface graphique dépasserait le cadre de cet article mais la figure 4 et l'énumération qui suit expliquent les différentes vues :

- 1) La barre d'outils et d'actions permet d'accéder aux paramètres de langues, aux modules ainsi qu'à la configuration de la vue centrale pour activer ou désactiver certaines fonctionnalités.
- 2) Cette zone permet de naviguer dans l'arborescence à l'instar de n'importe quel navigateur de fichiers. La flèche située sur la gauche des répertoires permet d'activer la vue récursive à partir du niveau sélectionné. Ainsi, si vous activez la vue récursive depuis le nœud racine, la vue centrale présentera l'intégralité des nœuds présents dans le système de fichiers virtuels.
- 3) La vue centrale permet de visualiser les fichiers et répertoires ainsi que d'appliquer des actions sur ceux-ci en utilisant le clic droit qui donne accès au menu contextuel. Il est possible de configurer cette vue pour basculer en mode vignettes, de zoomer mais également d'ajouter ou d'enlever des colonnes. Les éléments supprimés sont représentés en rouge. Il est également possible de filtrer la vue en utilisant le moteur de recherche intégré.
- 4) Lorsqu'un élément de la vue centrale est sélectionné, toutes les informations le concernant sont présentées dans la vue des attributs. Les attributs sont ordonnés par module. Sur la figure 4, les métadonnées NTFS et les types *magic* et MIME sont présentés. Dans la section 3, nous verrons qu'il est possible de rechercher un élément à partir de chacun de ces attributs.

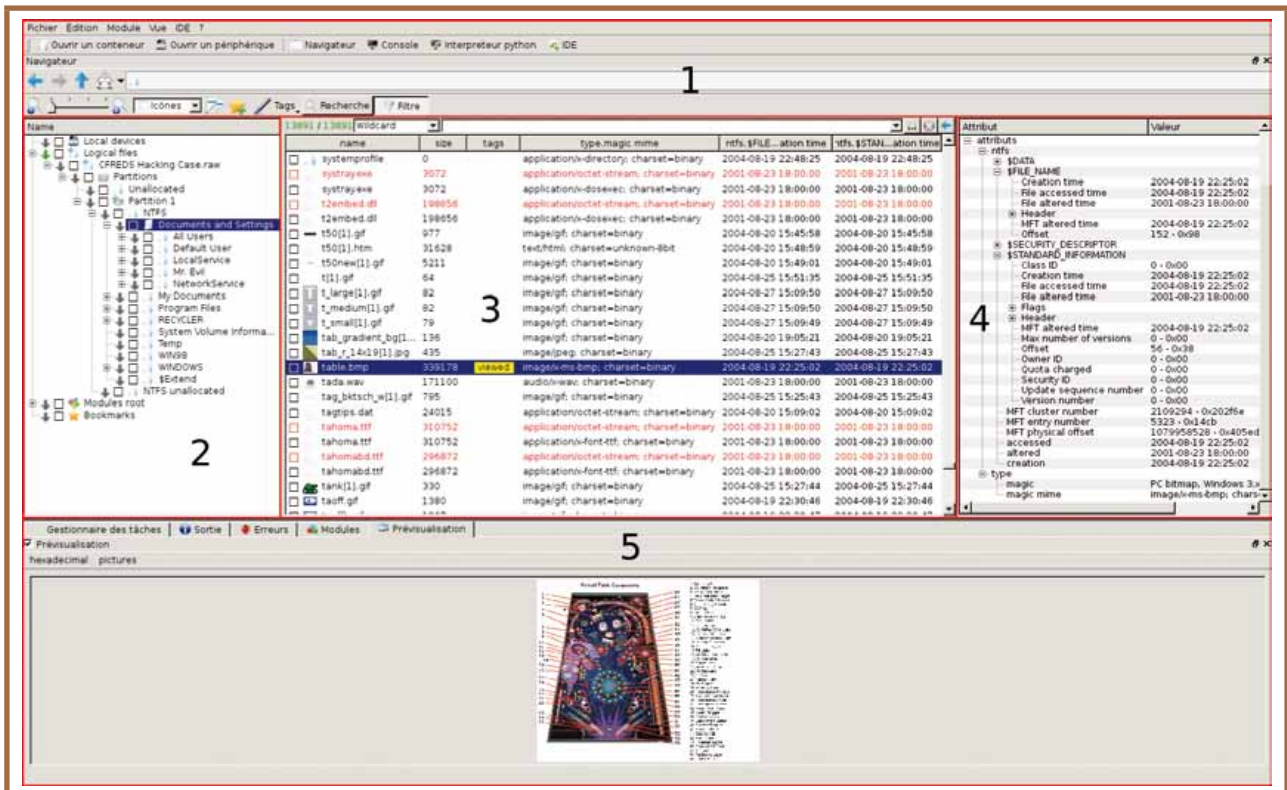


Fig 4 : Les différentes vues de l'interface graphique

5) La dernière zone contient différents onglets fournissant les informations telles que les modules appliqués dans le gestionnaire de tâches, les sorties et erreurs des modules ainsi que le mode de visualisation le plus adéquat. Ce dernier est automatiquement sélectionné en fonction du type du fichier et des modules de visualisation chargés.

2.3 Importer un jeu de données à analyser

La première étape pour travailler sur des données consiste à les charger dans DFF afin qu'elles soient accessibles à travers le système de fichiers virtuel (VFS). Ce chargement est rendu possible grâce à des modules faisant l'interface entre les périphériques ou fichiers présents localement et le VFS :

- **local** prend en entrée un ou plusieurs répertoires ou fichiers bruts comme, par exemple, une acquisition réalisée avec **dd** ou un répertoire contenant des documents bureautiques ou des images dont on voudrait accéder aux métadonnées ;
- **ewf** pour la prise en charge des conteneurs de type *Expert Witness Format*, un conteneur forensique compressé et éventuellement chiffré obtenu à l'aide d'un outil comme **ewfacquire** et pouvant également contenir des métadonnées telles que l'heure et l'empreinte de l'acquisition ou le nom de l'analyste ;
- **aff** (*Advanced Forensics Format*) qui est un format de fichier équivalent à **ewf** mais entièrement ouvert et documenté ;
- **devices** quant à lui permet d'accéder directement à un périphérique depuis DFF.

Bien entendu, tous les modules de DFF sont développés de sorte à garantir un fonctionnement en lecture seule garantissant ainsi l'intégrité des données chargées.

3 Utilisation de l'API à l'exécution

3.1 Présentation des données utilisées

Pour avoir un jeu de données constant au travers de nos exemples, nous allons travailler sur une acquisition librement disponible depuis le site digitalcorpora.org. Elle est composée de deux fichiers **ewf** que vous trouverez sur [5]. Brièvement, ce site propose différentes images de disques, de téléphones, de RAM ainsi que des captures réseau et scénario pour tester ses outils. Il y a également une section nommée « Real Data Corpus » [6] provenant

de disques et clés d'occasion achetés sur des sites de vente en ligne dont les propriétaires n'avaient pas supprimé les données qu'ils contenaient...

3.2 Utilisation du mode batch

Avant de lancer directement l'une des interfaces de DFF, nous allons dans un premier temps utiliser un module permettant de charger des fichiers de type *batch* directement au démarrage. Ces fichiers peuvent contenir à la fois du code Python et des commandes telles qu'on les fournirait en mode console. La syntaxe n'est pas compliquée : les lignes commençant avec un point d'exclamation sont vues comme une commande (application d'un module), celles commençant avec un dièse pour l'ajout de commentaires et toutes les autres sont interprétées par Python. Dans un premier temps, nous allons charger notre jeu de données en utilisant la commande **ewf** et lister la racine du système de fichiers virtuel avec **ls**. Pour ce faire, nous allons éditer un fichier et ajouter les lignes suivantes. Ce fichier sera ensuite fourni au démarrage de DFF en ligne de commande. Dans cet exemple, le fichier est nommé **misc.dfb**, avec **dfb** pour *digital forensics batch*, mais il n'y a pas de convention précise. La seule obligation étant que le fichier suive la syntaxe précédemment présentée.

```
#chargement de l'acquisition en fournissant le chemin vers les
fichiers ewf à monter
!ewf /data/dumps/nps-2008-jean.E??
print "Listing du repertoire racine"
!ls /
!fileinfo /Jean's\ hard\ drive\ from\ the\ first\ M57\ project
```

Pour lancer ce batch au démarrage de DFF, il suffit d'utiliser l'option **-b** suivie du chemin vers le fichier. Ici la commande sera **dff.py -b misc.dfb**. Sachez qu'il est également possible de lancer un batch à posteriori via l'utilisation du module **batch** depuis la console ou l'interface graphique. Dans le script précédent, la dernière ligne exécute la commande **fileinfo** sur le fichier virtuel nommé **Jean's hard drive from the first M57 project** résultant de l'application du module **ewf**. La commande **fileinfo** permet d'afficher les métadonnées d'un élément donné du VFS. Si tout s'est bien déroulé, vous devriez obtenir la sortie suivante : fig. 5 page suivante.

On peut visualiser les différentes métadonnées relatives aux fichiers en fonction du module appliqué. Ici nous accédons aux attributs du module **ewf** ainsi que les attributs relatifs aux types MIME et magic qui nous informent que le fichier correspond à un secteur de démarrage x86 avec une partition. Il est donc possible d'appliquer le module **partition** sur cet élément qui est d'ailleurs suggéré à la ligne commençant par « *relevant module(s)* ». Vous l'avez compris, il devient vite fastidieux de devoir consulter à chaque étape quel est le type de fichier nouvellement monté. Nous allons donc voir comment automatiser l'application de modules.

```
[OK] loading carverui v1.0.0
executing batch script /cases/misc/misc.dfb
Listing du repertoire racine
Jean's hard drive from the first M57 project

result: name :      Jean's hard drive from the first M57 project
node type :
relevant module(s) :  partition
generated by:      euf
size:              10737418240
attributes:

  euf
  MDS: 78a52b5bac78f4e711607707ac0e3f93
  acquiry_date: Mon Jan 31 22:38:29 2011
  acquiry_operating_system: Darwin
  acquiry_software_version: 20101104
  description: Jean's hard drive from the first M57 project
  evidence_number: 2008-M57-Jean
  examiner_name: Donny
  system_date: Mon Jan 31 22:38:29 2011

  type
  magic: x86 boot sector, Microsoft Windows XP MBR, Serial 0x39bf39be
; partition 1: ID=0x7, active, starthead 1, startsector 63, 20948637 sectors, code
offset 0xc0
  magic mime: application/octet-stream; charset=binary

#####
# Welcome on Digital Forensics Framework #
#####
dff / > |
```

Fig 5 : Sortie console après lancement du script batch

3.3 Application automatique des modules

DFF fournit, par le biais de l'API du gestionnaire de tâches, la possibilité d'automatiser l'application de modules sur des éléments du VFS. Cette fonctionnalité, appelée *postprocessing*, repose sur un mécanisme de signaux émis par le VFS lorsqu'un nouveau point de montage est créé et sur la détection de type et de son module associé. Plutôt que d'appliquer par défaut tous les modules compatibles, il est possible de choisir uniquement lesquels le seront de manière automatique. Il est nécessaire de fournir ces informations avant l'utilisation des modules vus en section 2.3. Nous allons donc ajouter les lignes suivantes au début de notre fichier batch, le tout agrémenté de quelques commentaires.

```
# import de l'API du gestionnaire de tâches
from dff.api.taskmanager.taskmanager import TaskManager
# Obtention de l'objet en charge du postprocessing
ppModules = TaskManager().ppModules
# Ajout des modules à appliquer de manière automatique
ppModules.add('partition')
ppModules.add('ntfs')
# exemple de passage d'argument à un module. Ici, l'argument offset
au module partition
ppModules.addArgument('partition', 'offset', 0)
```

Les modules **partition** et **ntfs** seront automatiquement appliqués dans le cas présent. Il faut également savoir que le type de chaque élément de l'arborescence reconstruit sera pré-calculé de manière multithreadée afin de pouvoir appliquer par la suite les modules compatibles. Cela permet par la suite d'effectuer une recherche par type quasi instantanée.

Nous sommes maintenant en mesure de naviguer dans le système de fichiers correspondant au système

acquis. L'objectif, à présent, consiste à accéder depuis l'interpréteur Python aux éléments du VFS ainsi qu'à leurs métadonnées et leur contenu.

3.4 Accès aux éléments du VFS

Pour UNIX, tout est fichier, pour DFF, tout est **Node**. Le système de fichiers virtuels de DFF est en charge de la cohérence de l'arborescence des différents points de montage. Cette arborescence est composée de nœuds qui sont accessibles à travers différentes méthodes de l'API. Nous allons voir quelles possibilités s'offrent à nous pour travailler sur les éléments présents dans le VFS. Deux méthodes permettent d'accéder aux **Nodes** :

- soit pour un **Node** donnée avec la méthode **getnode()** qui prend en paramètre une chaîne de caractères représentant le chemin de celui-ci ;
- soit en utilisant la méthode **walk()** équivalente à celle fournie par **os.walk()** de Python.

Cette dernière méthode fournit les noms de fichiers présents dans un répertoire en traversant l'arborescence soit de la racine vers les feuilles de l'arbre (*top-down*) soit des feuilles vers la racine (*bottom-up*). Pour chaque répertoire de l'arborescence sous-jacente, la fonction fournit un tuple de trois éléments : le nom du répertoire actuel, la liste des répertoires et la liste des fichiers présents dans celui-ci. Le premier paramètre correspond à l'endroit à partir duquel traverser l'arbre. Cela peut être soit une chaîne de caractères soit directement un **Node** (obtenu précédemment avec **getnode()**, par exemple), les éléments du tuple résultant étant du même type que celui fourni en entrée. Le deuxième paramètre correspond au type de traversée (*top-down* ou *bottom-up*), optionnel, qui est par défaut en mode *top-down*. Enfin, le dernier paramètre correspond à la profondeur maximale de traversée. Il est également optionnel et égal à -1 par défaut, ceci signifiant qu'il n'y a pas de limite de profondeur.

3.4.1 Nodes

Lorsqu'on obtient un objet de type **Node**, différentes méthodes sont accessibles. Nous ne verrons ici que les méthodes qui seront utiles pour la suite de l'article :

- **path()**, **name()**, **extension()** et **absolute()** qui retournent chacune une chaîne de caractères correspondant respectivement au chemin, au nom, à l'extension et au nom complet (chemin + nom) associé au nœud ;
- **size()** qui fournit la taille du nœud en octets ;
- **attributes()** qui retourne une *map* relative aux attributs, celle-ci associant une clé, représentée par une chaîne de caractères, à une valeur de type **Variant** et pouvant elle-même être encapsulée dans un type **Variant** rendant possible une arborescence d'attributs ;

- **dataType()** fournit le type du nœud reposant sur l'utilisation de la librairie magic et retourne également une map pour les types MIME et magic ;
- **open()** ouvre un fichier et retourne un objet de type **VFile** (présenté plus en détails dans la section suivante) qui donne accès au contenu d'un élément.

Dans l'exemple suivant, nous allons utiliser ces différentes méthodes depuis l'interpréteur Python afin de pouvoir traverser l'arbre et récupérer les fichiers de type image. Si vous utilisez l'interface graphique, il vous suffit de lancer l'interpréteur en cliquant sur le bouton associé dans la barre d'outils. Si vous utilisez l'interface console, vous devez appliquer la manipulation présentée en note de la section 2.

```
>>> from dff.api.vfs.vfs import vfs
>>> v = vfs()
root = v.getnode("/Jean's
hard drive from the first M57 project/Partitions/Partition 1/NTFS")
>>> images = []
>>> for (current, dirs, files) in v.walk(root):
...     for file in files:
...         if file.dataType()['magic mime'].toString().
startswith('image'):
...             images.append(file)
>>> print len(images)
5938
>>> print images[-1].absolute(), images[-1].size()
/Jean's hard drive from the first M57 project/Partitions/Partition
1/NTFS/WINDOWS/ServicePackFiles/ServicePackCache/i386/lvback.gif 7047
```

3.4.2 Lecture et recherche dans le contenu d'un élément

Maintenant que nous sommes en mesure d'accéder à tous les éléments du système de fichiers virtuel, nous allons voir comment lire et rechercher des motifs dedans. Pour ce faire, nous allons utiliser les **VFiles** précédemment cités ainsi que l'API de recherche. Pour rappel, un **VFile** est obtenu en appelant la méthode **open()** sur un **Node**. Cet objet est l'équivalent de l'objet **file** de Python qui fournit les méthodes pour lire et se déplacer dans un fichier et qui sont détaillées ci-après :

- **seek(offset[, whence])** qui permet de se déplacer dans un fichier, avec **offset** un entier et **whence** (optionnel) qui spécifie à partir de quel endroit l'offset sera calculé (les valeurs possibles sont les mêmes que l'appel système **lseek()** sur les systèmes POSIX ;
- **read([size])** retourne un tampon d'une longueur **size** (ou moins si **size** est plus grand que la capacité de lecture) ou, sans argument, un tampon égal à la taille restante par rapport à la position actuelle, cette dernière fonctionnalité devant donc être utilisée avec précaution ;
- **tell()** retourne la position actuelle dans le fichier.

L'objet **VFile** fournit également des méthodes de recherche prenant en paramètre obligatoire un objet de type **Search** détaillé par la suite et deux paramètres optionnels : le premier correspond à l'offset à partir

duquel commencer la recherche, le second à l'offset auquel s'arrêter. Il est ainsi possible de rechercher des motifs uniquement sur une fenêtre donnée. Par défaut ces arguments se voient attribués respectivement les valeurs 0 et 2**64-1. Les fonctions de recherche supportées sont :

- **find** retourne l'offset de la première occurrence du motif recherché ou -1 s'il n'est pas trouvé ;
- **rfind** l'équivalent de **find** retournant l'offset de la dernière occurrence du motif recherché ou -1 s'il n'est pas trouvé, cette méthode n'étant pas utilisable avec les expressions régulières et les recherches par approximation ;
- **indexes** qui retourne une liste d'offsets (ou vide) pour chaque occurrence trouvée.

L'objet **Search**, pris en paramètres par les trois méthodes précédentes, permet de configurer le type de motif que l'on souhaite rechercher. Les syntaxes des motifs sont les suivantes :

- chaîne de caractères fixe avec algorithme optimisé ;
- chaîne pouvant contenir les méta-caractères (ou jokers) « ? » pour substituer un caractère et « * » pour substituer zéro caractère ou plus ;
- les expressions régulières ainsi que la recherche par approximation qui reposent sur l'utilisation de la bibliothèque TRE [7].

Pour chaque type de recherche il est également possible de préciser si celle-ci est sensible à la casse ou non.

Dans l'exemple suivant, nous allons rechercher tous les fichiers de type texte contenant le mot « password » et afficher le nom absolu de celui-ci et l'offset de la première occurrence. Enfin nous verrons comment ouvrir un fichier et se déplacer à un offset donné pour y lire son contenu.

```
>>> from dff.api.search.libsearch import Search
>>> part1 = v.getnode("/Jean's hard drive from the first M57 project/Partitions
1/NTFS")
>>> s = Search()
>>> s.setPatternSyntax(Search.Fixed)
>>> s.setPattern("password")
>>> s.setCaseSensitivity(Search.CaseInsensitive)
>>> s.compile()
>>> for (current, dirs, files) in v.walk(root):
...     for file in files:
...         if file.dataType()['magic mime'].toString().startswith("text"):
...             vfile = file.open()
...             idx = vfile.find(s)
...             ret = vfile.close()
...             if idx != -1:
...                 print 'match found in', file.absolute(), 'at', hex(idx)
match found in /Jean's hard drive from the first M57
project/Partitions/Partition 1/NTFS/Documents and
Settings/Administrator/Application
Data/Mozilla/Firefox/Profiles/towjib3x.default/compreg.dat at 0x16a21L
[...]
match found in /Jean's hard drive from the first M57
project/Partitions/Partition
1/NTFS/WINDOWS/ServicePackFiles/ServicePackCache/i386/layout.inf at 0x38bf3L
>>> vfile = v.open("/Jean's hard drive from the first M57
project/Partitions/Partition 1/NTFS/Documents and Settings/Jean/Local
Settings/Temporary Internet Files/Content.IE5/20S83G5U/omniuni[2].js")
>>> vfile.seek(0x3daa)
>>> vfile.read(16)
"PASSWORDFILE.in"
```

3.4.3 Utilisation des fonctionnalités de recherche

La dernière fonctionnalité de l'API présentée dans cet article correspond au moteur de recherche intégré. Il est possible de chercher sur tous les attributs d'un élément ainsi que dans son contenu. La syntaxe repose sur l'évaluation d'expressions booléennes. L'ordre de priorité des opérateurs, de la plus haute à la plus basse, est le suivant :

- **not** ;
- **>**, **>=**, **<**, **<=**, **==**, **!=** ;
- **and** ;
- **or**.

Afin de gérer les priorités entre opérateurs, il est recommandé d'utiliser les parenthèses. Différents types sont utilisables pour une requête.

3.4.3.1 Les entiers et les booléens

Les entiers peuvent être écrits soit en base dix soit en base seize avec le préfixe **0x**. La valeur maximale gérée est un entier de 64 bits non signé. Les expressions utilisant une comparaison booléenne (uniquement avec les opérateurs **==** ou **!=**) utiliseront les mots clés **true** ou **false**.

3.4.3.2 Les chaînes de caractères

Le moteur de recherche gère les mêmes types de chaînes présentés dans la section relative aux recherches dans le contenu. Afin de différencier les syntaxes à utiliser il est nécessaire de les décorer :

- chaîne fixe entourée de guillemet (ex. : **"password"**) ;
- motif de type *shell* entouré de **\$**, (ex. : **\$.doc\$**) ;
- expression régulière telle qu'utilisée en Perl (ex. : **/[A-Z0-9-._%+]+@[A-Z0-9-]+\.[A-Z]{2,6}/**) ;
- expression approximative avec le caractère tilde (ex. : **~password~**) qui pourra par exemple correspondre au mot « p4ssword ».

Les requêtes utilisant les chaînes de caractères doivent utiliser l'opérateur **matches** (**==** est uniquement utilisable avec les chaînes fixes). Par exemple pour rechercher tous les fichiers ayant l'extension **.doc** on utilisera l'expression **name matches \$.doc\$**.

3.4.3.3 Les listes

Au lieu d'écrire des expressions à rallonge pour comparer plusieurs valeurs pour un attribut d'un élément, il est possible d'utiliser directement des listes. Par exemple, si l'on souhaite chercher tous les éléments du VFS ayant soit l'extension **.exe** soit l'extension **.dll**, on pourra utiliser l'expression **name in [\$.exe\$, \$.dll\$]** équivalente à **name matches \$.exe\$ or name matches \$.dll\$**.

Il est également possible d'utiliser un compteur pour le nombre d'éléments minimum à faire correspondre. Cette fonctionnalité est surtout utilisée pour la recherche de mots dans le contenu d'un fichier. La syntaxe est la suivante : **data matches any|all|none|<nombre> of [liste de patterns]**. Avec **any** pour n'importe quel motif de la liste trouvé, **all** pour tous, **none** pour aucun et un nombre si l'on veut par exemple au moins deux motifs de la liste qui soient présents. Ainsi, l'expression **data matches 3 of ["quick", ~brown~, /f[ox]*/, \$?umps\$, \$lazy*\$, "dog"]** sera validée uniquement si le contenu contient au minimum trois des motifs de la liste.

3.4.3.4 Les timestamps

La syntaxe pour les *timestamps* est basée sur la RFC 3339 [8]. Il est possible de comparer uniquement sur une année en utilisant le mot clé **year** (ex : **year == 2013**). Pour des recherches plus précises, il existe deux façons de comparer des informations temporelles. Soit en comparant toutes celles relatives à un élément en utilisant le mot clé **time**, soit en utilisant précisément l'attribut temporel à comparer. Le format pour un jour précis est **AAAA-MM-JJ** et pour une heure précise **AAAA-MM-JJThh:mm:ss**. Ainsi la requête **time > 2013-01-01T00:00:00** sélectionnera les nœuds ayant au moins une information temporelle supérieure au 1er janvier 2013, minuit.

3.4.3.5 Les attributs longs

Les attributs sont ceux fournis par la commande **fileinfo** dans la console ou dans la vue attributs dans l'interface graphique. Il suffit de copier-coller le nom de l'attribut et de l'encadrer avec des « @ ». Ainsi, la syntaxe pour connaître les fichiers ayant une date de création antérieure au 1er janvier 2013 pour l'attribut **\$FILE_NAME** du système de fichier NTFS serait la suivante : **@ntfs.\$FILE_NAME.Creation time@ < 2013-01-01T00:00:00**

Il est également possible d'utiliser un motif de type *shell* pour un champ de l'attribut. Un exemple d'utilisation est présenté en section 3.4.4.

3.4.3.6 Les attributs prédéfinis

Enfin, plusieurs mots clés correspondant à des attributs sont prédéfinis. Les mots clés prédéfinis sont les suivants :

- **magic** et **mime** pour travailler respectivement sur les types magic et MIME d'un élément et qui sont utilisés avec une comparaison de chaîne de caractères (ex. : pour rechercher tous les fichiers de type PNG, on utilisera la requête **mime == "image/png; charset=binary"**) ;
- **size** pour comparer avec la taille d'un élément, éventuellement en utilisant des multiplicateurs directement dans la syntaxe (**KB**, **MB** ou **GB**) ;

- **deleted** pour obtenir ou exclure les éléments supprimés (ex. : **deleted == true** fournira tous les éléments supprimés) ;
- **folder, file** pour filtrer les éléments de type répertoire ou fichier (ex. : pour obtenir tous les fichiers supprimés nous utiliserons la requête **file == true and deleted == true**) ;
- **name, extension** et **path** pour comparer une chaîne de caractère au nom, à l'extension ou au chemin de l'élément (ex. : pour rechercher les fichiers avec une extension **.exe** en dehors des répertoires **Windows** et **Program Files**, on pourrait utiliser la requête **extension == "exe" and not (path matches /NTFS\ (Windows|Program Files)/)**).

3.4.4 Utilisation de l'API du moteur de recherche

L'API du moteur de recherche peut être intégrée dans un script afin de rechercher ou de filtrer uniquement les éléments intéressants. Chaque filtre créé se voit associer un nom et une requête. Après avoir compilé la requête, il est possible d'appliquer la recherche à partir d'un nœud obtenu avec la méthode **getNode** du VFS. Un mécanisme de signaux permet également d'obtenir des informations sur la progression de la requête, le nombre actuel de correspondances trouvées mais également de stopper la recherche si besoin. Dans l'exemple suivant l'objectif est de chercher les mails contenant deux des trois mots suivants : 'salaries', 'names' ou 'employees' dans les mails reçus ou émis par Jean. Nous partons du principe qu'il puisse y avoir plusieurs utilisateurs sur la machine et donc nous n'allons pas nous restreindre uniquement à la boîte Outlook appartenant à Jean. Afin d'appliquer automatiquement le module de reconstruction des boîtes mails, il est nécessaire d'ajouter à notre script batch la ligne suivante : **ppModules.add('exchange')** avant l'application du module **ewf**. Dans un autre fichier batch, vous pouvez ajouter le listing suivant :

```
#import de l'API de filtre et vfs
from dff.api.filters.libfilters import Filter
from dff.api.vfs.vfs import vfs
# obtention du vfs
v = vfs()
# creation d'un objet filtre avec le nom mail lookup associe
f = Filter('mail lookup')
# compilation de la requete
f.compile('@exchange.Recipients.*.Email address@ == "jean@m57.biz"
or @exchange.Message headers.Sender email address@ == "jean@m57.
biz") and data matches 2 of ["salaries", "names", "employees"]')
# obtention du noeud racine
root = v.getNode('/')
# evaluation de chaque element de l'arbre depuis le noeud racine
f.process(root)
# recuperation de la liste de nœuds validant la requete
matches = f.matchesNodes()
# affichage du nombre de correspondances
print len(matches)
# Pour chaque nœud, affichage de son chemin absolu, de ses
attributs et de son contenu.
```

```
for match in matches:
    print match.absolute()
    print match.attributes()
    vfile = match.open()
    print vfile.read()
    vfile.close()
```

Vous pouvez l'exécuter une fois le post-traitement terminé en utilisant la commande **batch** depuis l'interface console ou graphique. La signification de la requête est la suivante : entre parenthèses, la requête signifie que **jean@m57.biz** doit correspondre à l'un des destinataires OU à l'émetteur (vous remarquerez l'utilisation du joker dans la première expression afin de ne pas avoir à spécifier chaque champ destinataire (Recipient 1, Recipient 2, ...) ; si l'une ou l'autre des expressions est évaluée à vrai, alors la dernière expression sera également évaluée. Celle-ci signifie que le contenu du mail doit contenir au moins deux des mots spécifiés dans la liste.

Conclusion

Nous avons vu dans cet article comment il était possible d'automatiser l'application de modules avec DFF ainsi qu'accélérer une analyse grâce à l'utilisation de certaines fonctionnalités de l'API. En plus des possibilités d'automatisation en Python, il est également possible de développer différents types de modules mais cela nécessiterait un article à part entière pour couvrir ce sujet. Si, à la suite de la lecture de cet article, vous avez des questions ou des suggestions, n'hésitez pas à venir échanger sur le canal IRC [9] du projet ou par le biais des listes de diffusion [10]. ■

■ REMERCIEMENTS

Je tiens à remercier Emmanuel Fleury, Sébastien Larinier et David Lesperon pour leur relecture attentive et leur précieux commentaires ainsi que Benjamin Caillat et Guillaume Arcas pour leur support.

■ RÉFÉRENCES

- [1] www.digital-forensic.org
- [2] www.arxsys.fr
- [3] git://git.digital-forensic.org/dff
- [4] <http://www.cert.org/forensics/tools/>
- [5] <http://digitalcorpora.org/corpora/scenarios/m57-jean>
- [6] <http://digitalcorpora.org/corpora/disk-images/real-data-corpus>
- [7] <http://laurikari.net/tre/>
- [8] <http://www.ietf.org/rfc/rfc3339.txt>
- [9] #digital-forensic sur freenode
- [10] <http://lists.digital-forensic.org/listinfo>



BIG DATA

quand la taille compte !

Le Big Data est l'un des buzz de ces deux dernières années. Au-delà de l'aspect marketing largement relayé par les vendeurs de serveurs, d'espaces de stockage et de logiciels, le Big Data s'appuie sur plusieurs concepts techniques éprouvés ou émergents que nous allons exposer dans ce dossier.

Le Big Data est le domaine informatique qui s'attache à traiter des masses de données impossibles à exploiter avec des outils « classiques » tels que les architectures serveurs usuelles (serveurs virtualisés, espaces de stockage SAN mutualisés, disques SAS), les bases de données relationnelles ou encore les codes applicatifs ne tirant pas partie des possibilités vectorielles et parallèles des processeurs actuels.

Ces problématiques ne sont pourtant pas récentes. Certains acteurs publics ou privés ont commencé à se heurter aux limites des systèmes d'informations usuels et tout particulièrement aux bases de données relationnelles pour le stockage et l'exploitation d'une très grande quantité de données. Ce fut notamment le cas de Google qui a dû développer différentes briques logicielles pour stocker et exploiter la quantité de données nécessaires à son moteur de recherche. L'une de ces briques est BigTable, une base de données noSQL, distribuée de type « clef/valeur » développée par Google en 2004 et qui a inspiré les projets libres tels que Cassandra, HBase, Hypertable ou encore Redis.

Force est de constater, qu'au-delà du buzz, l'évolution récente tant logicielle que matérielle rend accessible aux entreprises de taille plus modestes que Google l'accès au traitement des Big Data.

En effet, le besoin de traiter de très grandes masses de données n'est pas nouveau, que ce soit pour centraliser, corrélérer et exploiter une très grande quantité de logs, réaliser du

profiling des habitudes clients pour mieux cibler les publicités, la recherche scientifique telle que le décodage du génome,...

Jusqu'à quelques années, les directions informatiques se heurtaient aux limitations matérielles et logicielles disponibles. A moins d'investir dans un super ordinateur et une armée d'ingénieurs spécialisés, certains traitements restaient inaccessibles. Les architectures matérielles sont toutefois en pleine évolution. En ce qui concerne les processeurs, on a vu apparaître sur les architectures x86 des capacités vectorielles (instructions SSE) et multi-coeurs offrant la possibilité de calculs parallèles. En matière de stockage, l'apparition des disques SSD a fait exploser le nombre d'IOPS (nombre d'opérations d'I/O par seconde) d'un facteur 1.000 par rapport aux disques durs classiques. L'offre logicielle a également beaucoup évolué depuis la fin des années 2000 avec notamment l'apparition des bases de données noSQL qui sont conçues pour les très gros traitements ainsi que l'évolution des logiciels pour tirer partie des architectures multi-coeurs et vectorielles.

Et pour finir, peut-être la meilleure raison de s'intéresser aux Big Data est que les études de cabinets de recrutement désignent cette activité comme l'une des mieux rémunérées[1] du secteur informatique !

Cedric Foll / @follc

[1] <http://www.urbanlinker.com/etude-des-salaires-developpeurs-2013-recrutement/>

HPC, « BIG DATA » : DE LA THÉORIE À LA PRATIQUE (1/2)

Adrien Guinet – adrien@guinet.me - Ingénieur R&D chez Quarkslab

Serge Guelton – serge.guelton@telecom-bretagne.eu - Ingénieur R&D
chez Quarkslab, Chercheur associé à Télécom Bretagne en Compil & HPC

« *Premature optimization is the root of all evil* »
Donald Knuth



mots-clés : MESURE DE PERFORMANCE / PARALLÉLISME / MULTI-CŒURS /
INSTRUCTIONS VECTORIELLES

Depuis quelques années, la création d'applications performantes et capables de tirer partie de la capacité de calcul de une à plusieurs (milliers de) machines, traitant et créant des quantités d'informations de plus en plus importantes, est devenue un problème auquel de plus en plus d'industriels et divers corps de métiers font face. Cela va du traditionnel calcul scientifique, au domaine de la finance ou encore de la « business intelligence ».

1 Introduction

Le traitement de grosses masses de données demande de plus en plus de stockage et/ou de capacité de calculs. Pourtant, un disque dur actuel permet de lire et écrire à environ 200Mo/s, de faire environ 200 opérations d'écriture et/ou lecture par seconde ou Input/Output Operations Per Second (IOPS). Les Solid State Drive (SSD) améliorent un peu la donne avec pour les meilleurs d'entre eux un débit d'environ 600Mo/s et 100k IOPS. Un CPU Intel Xeon récent est capable de chiffrer un flux en AES à environ 512Mo/s¹.

Pour comparaison, Google indexe environ 20 milliards de sites par jour [1], une infrastructure de serveurs classiques peut produire jusqu'à plusieurs centaines de Go de fichiers journaux par jour. Ces services ne peuvent pas être assurés par une machine conventionnelle, le parallélisme est au cœur des solutions utilisées, et cela implique d'autres modèles de conceptions et programmation.

Cette article divisé en deux parties présente un peu plus en profondeur ce qu'on appelle « scalabilité linéaire », ce qui se cache derrière les termes marketing utilisés aujourd'hui (« Big Data », « NoSQL » et

consorts). Nous ferons ensuite une présentation des performances du « commodity hardware » actuels et des outils permettant de faire ces mesures, ainsi que des différents niveaux de parallélisme existant. Nous rentrerons ensuite dans les détails afin d'arriver à construire une application réellement scalable, et des exemples applicables dans le domaine de la sécurité.

Remarques générales

- Tout au long de l'article, les notations Ko, Mo ou Go sont utilisées. Les conversions suivantes sont utilisées : 1Go = 1024Mo, 1Mo = 1024Ko et 1Ko = 1024 octets.
- Lorsqu'un gain de performance est défini, il est égal au temps après optimisation divisé par le temps original (avant optimisation).
- Sauf mention contraire, les différents benchmarks ont été réalisés sur un Xeon E3-1245v2 cadencé à 3.8Ghz, équipé de 32Go de RAM.
- Tous les exemples de codes mentionnés sont disponibles ici : <https://github.com/aguinet/misc-examples>.



2 Les mots à la mode : performances, scalabilité linéaire, goulot d'étranglement et leurs amis

2.1 Les goulots

Commençons par quelques mots de sagesse de Donald Knuth : « *Premature optimization is the root of all evil* ». En effet les performances d'une application peuvent être un critère important ou non. Généralement, beaucoup de prototypes ne sont pas fait avec cette problématique en tête, et c'est une très bonne chose ! La question reste la suivante : comment trouver ce qui limite les performances d'une application ? Quels outils utilisés et comment s'en sortir sans devoir la réécrire complètement ?

Le concept de goulot d'étranglement ou bottleneck, est primordial. On considère souvent qu'une application passe 90 % de son temps dans 10 % du code. C'est donc là qu'il va falloir frapper fort.

Illustrons ces propos avec un exemple simple, connu de tous : **grep**.

Grep est un bon exemple car c'est un outil qui travaille sur potentiellement une quantité de données importantes, et peut faire des traitements relativement « complexes » (en terme de calcul CPU), comme vérifier une expression régulière.

Faisons un premier test de performance sur un PC utilisant un disque dur rotatif classique et un Core i7. Nous allons analyser un fichier d'authentification de serveur Linux classique (**/var/log/auth.log**) pour rechercher les connexions SSH échouées :

```
# time grep -E 'invalid user (\S+) from ([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+) port ([0-9]+)' /var/log/auth.log >/dev/null
real    0m12.174s
```

Et un autre test pour estimer le débit du disque dur utilisé :

```
# dd if=/var/log/auth.log of=/dev/null bs=1M
[...]
555995144 bytes (556 MB) copied, 4.31017 s, 129 MB/s
```

Comme nous pouvons le voir, notre fichier de 530Mo (pour **dd**, 1 MB=1000*1000 octets [2] hors la convention choisie ici est 1Mo = 1024*1024 octets) a été lu en 12.174s, ce qui donne un débit de traitement moyen d'environ 42Mo/s. Cela reste en dessous du débit moyen de lecture de notre fichier, soit environ 123Mo/s.

Avec ce petit exemple, il apparaît que le facteur limitant est le traitement de l'expression régulière, et non le débit d'accès aux données. Pour améliorer cela, nous pouvons utiliser la commande **parallel**¹ afin d'utiliser les différents cœurs disponibles sur la machine :

```
# time parallel --pipe --block 16M grep -E 'invalid user (\S+) from ([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+) port ([0-9]+)' </var/log/auth.log >/dev/null
[...]
real    0m4.069s
```

La syntaxe utilisée est assez intuitive et peut être décrite ainsi :

- **--pipe** indique à **parallel** de découper ce qu'il reçoit en entrée en plusieurs blocs, et de traiter ces blocs sur différents CPU ;
- **--block** indique la taille du bloc. Une taille de bloc importante permet de tirer au maximum profit du débit du disque original, et limite le sur-coût dû à la gestion du parallélisme (car moins de blocs), mais une taille de bloc trop importante limite le parallélisme (cas limite : 1 bloc de la taille du fichier). C'est ce qu'on appelle généralement le problème de la granularité du découpage ;
- la commande à exécuter. **parallel** va créer des sous processus **grep** auxquels il va transmettre (via **stdin**) les différents blocs à traiter.

Nous obtenons ainsi un débit de traitement d'environ 118Mo/s, assez proche du débit du disque initial. Le facteur limitant est donc devenu le débit du disque dur en entrée. Rajouter des processeurs ne changera plus grand chose à la vitesse de traitement.

Ce petit exemple illustre deux choses. Premièrement, il est facile, avec quelques lignes de scripts, de paralléliser un processus et donc de gagner du temps. Deuxièmement, nous venons de voir ici l'effet de deux goulots d'étranglement différents : le premier était la puissance de calcul, il a ensuite été remplacé après « optimisation » par l'accès aux données.

Tout cela influe sur la scalabilité potentielle d'un processus, ce qui nous amène à la section suivante.

2.2 La « scalabilité linéaire »

Classiquement, la complexité temporelle d'un algorithme est décrite par le nombre d'opérations nécessaires à son aboutissement. Un algorithme passe à l'échelle linéairement si sa complexité temporelle est en $O(n)$ (n étant la taille des données en entrée de cette algorithme). Le même concept existe avec la complexité spatiale, à savoir son occupation mémoire.



Par exemple, les meilleurs algorithmes de tris ont actuellement une complexité temporelle en moyenne de $O(n \cdot \log(n))$ (et ne passent donc pas à l'échelle linéairement).

Dans la même idée, nous pouvons dire qu'une application est « scalable linéairement » s'il est possible d'accroître ses capacités de traitement linéairement en augmentant les ressources qui lui sont associées. Si on définit l'accélération d'un programme parallèle comme le ratio du temps d'exécution sur une machine sur le temps d'exécution sur n machines, cela revient à dire que l'accélération est une fonction linéaire du nombre de processeurs. C'est malheureusement rarement le cas.

Par exemple, nous pouvons nous demander :

Est-ce que faire tourner cette application sur un processeur à 4 cœurs va impliquer un gain de performance de quatre ?

Est-ce que si elle utilise un GPU, lui en allouer quatre va lui permettre d'être quatre fois plus rapide ?

Est-ce que lui fournir une ferme de quatre serveurs identiques va lui permettre d'aller quatre fois plus vite ?

Vous avez compris l'idée générale...

Ce critère de scalabilité peut être appliqué sur divers points de mesure de l'application. Par exemple, pour **grep**, cela peut être simplement le débit de traitement. Concrètement, en posant d ce débit en fonction du nombre de CPU utilisé, nous dirons par la suite que **grep** est linéairement scalable par rapport au nombre de CPU si $d(n) = a \cdot n$. Si $a = 1$, alors cette scalabilité serait dite « parfaite ». Si $a > 1$, on parle de scalabilité « sur-linéaire ».

Dans le cas général, ces fonctions ont un seuil au delà duquel aucun gain n'est possible, à cause d'un autre goulot d'étranglement qui apparaît ailleurs. C'est le phénomène que nous avons vu section 1. En effet, si nous traçons la courbe du gain dû à l'utilisation de plusieurs cœurs, nous obtenons la figure 1 :

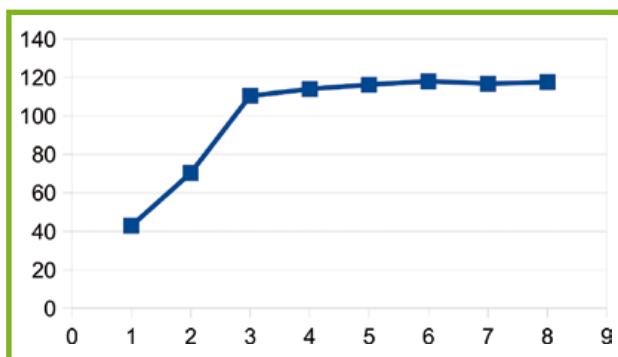


Fig. 1 : Bande passante moyenne de données traitées par **grep** (en Mo/s) par rapport au nombre de processus utilisés

Il apparaît assez clairement qu'au bout de 3 cœurs, le disque dur devient le composant limitant.

2.3 Loi d'Amdhal et chemins critiques

Pour aller plus loin, la théorie qui est généralement citée et qui capture le phénomène décrit ci-dessus se base sur la loi d'Amdhal. Cette dernière énonce que tout programme possède une partie séquentielle et que c'est cette partie qui limitera tôt ou tard l'accélération. Intuitivement, si le programme passe 90 % de son temps sur une section à scalabilité linéaire, et 10 % de son temps sur une section séquentielle, la parallélisation sur 9 cœurs de la section parallèle nous amène à une application qui ne tourne d'une part que 5 fois plus vite (avant $1s + 9s = 10s$, après $1s + 1s = 2s$), et d'autre part qu'on ne pourra jamais avoir une accélération supérieure à $10(1s + 9s / \infty = 1s)$. C'est très décevant, mais il vaut mieux en avoir conscience avant de promettre monts et merveilles...

Cette loi est très liée au concept de chemin critique. Le chemin critique d'une application, c'est le plus court chemin dans son graphe d'exécution (parallèle) depuis le lancement du programme jusqu'à la fin du programme. Quand vous faites une tarte aux pommes (facile ! : i) préparer la pâte, ii) épluchez les pommes, iii) faire cuire la tarte). On peut se mettre à plusieurs pour épluchez les pommes, mais regarder le four bosser ne changera pas le temps de cuisson. On peut préparer la pâte pendant qu'on épluche les pommes par contre. Comme il faut plus de temps pour préparer la pâte que pour éplucher une pomme, le chemin critique est i) préparer la pâte (pendant ce temps vos camarades épluchent rageusement les pommes) et ii) enfourner. Vous aurez beau ajouter des ressources, le chemin critique restera en travers de votre chemin. Belle leçon de vie !

2.4 Mesurer les performances d'une application

Comme nous avons pu le voir, il est très important de disposer des bons outils afin de mesurer les performances de son application, afin d'identifier les différents « bottleneck ».

Une première approche est d'utiliser des outils de profilage (*profiling*) comme le vénérable **gprof**, le puissant **callgrind**, **prof** qui est intégré au noyau ou **Intel Vtune** (payant) qui permettent d'identifier les portions de codes utilisant le plus de cycles CPU. Certains outils donnent de plus des métriques permettant de mieux comprendre la cause de l'utilisation de ces cycles (défauts de cache, attente d'entrée sorties, etc...).

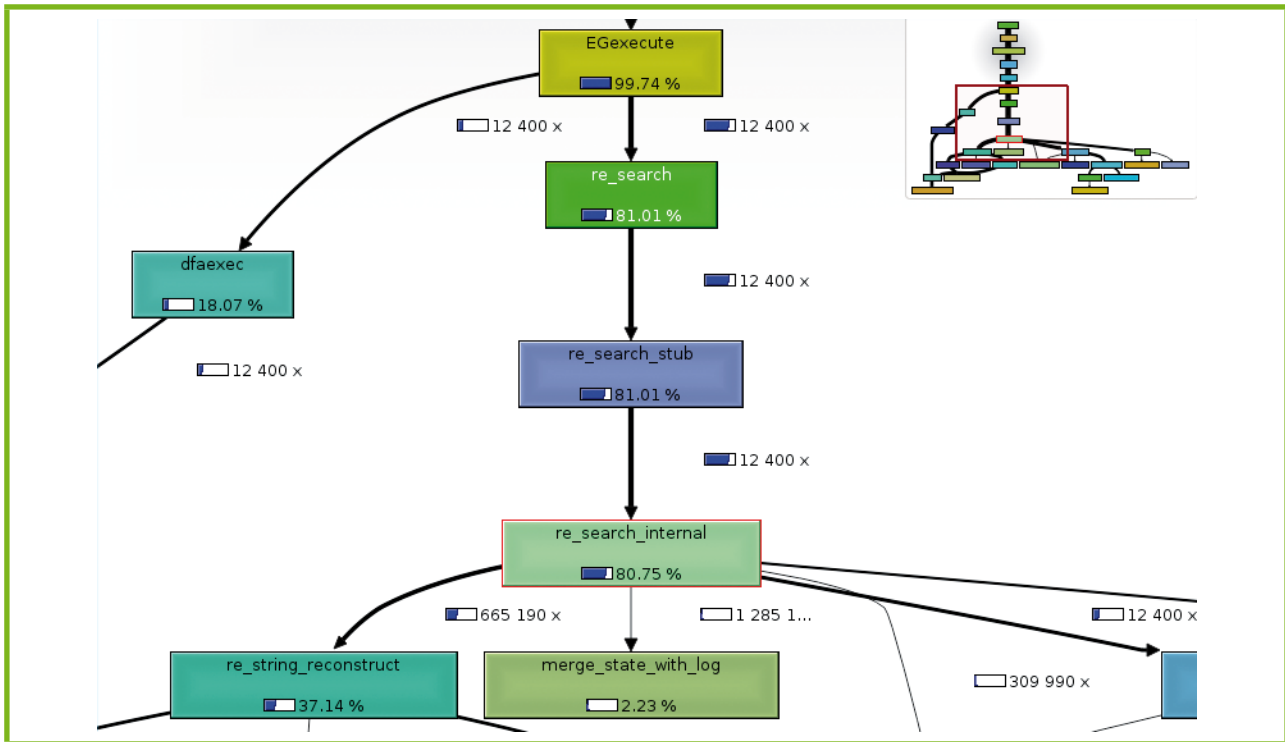


Fig. 2 : Exemple de sortie de callgrind sur grep

Callgrind [3] est un outil intéressant basé sur valgrind. Il exécute le programme en émulant le processeur et en profite pour calculer le nombre de cycles cumulés de chaque fonction. Dans la même idée, cachegrind, toujours basé sur valgrind, émule le comportement du cache. Ainsi, si une fonction **foo** appelle **bar**, alors le nombre de cycles CPU nécessaires à l'exécution de **bar** sera ajouté à celui de **foo**. Avec ce mécanisme, la fonction **main** arrive avec quasiment 100 % des cycles CPU (« quasiment » car il reste l'initialisation du linker ou encore la construction des objets C++ statiques qui est faite avant l'exécution de **main**). Ainsi, cela permet de voir assez rapidement quelles parties d'une application prennent le plus de cycles CPU, et ainsi de se préoccuper en premier principalement de celles-ci.

Callgrind se lance à travers valgrind. Pour l'installer sous Debian, faire simplement, en tant que root :

```
# aptitude install valgrind
```

Voici un exemple d'utilisation sur **grep** :

```
$ valgrind --tool=callgrind grep -E 'invalid user (\S+) from'
<auth.log >/dev/null
```

Un fichier nommé **callgrind.out.PID** va être créé. Il peut être visualisé grâce à l'outil graphique **kcachegrind** (disponible généralement dans votre distribution Linux préférée). Sous Debian, on peut simplement faire :

```
# aptitude install kcachegrind
```

La figure 2 nous montre ainsi que la fonction **re_search_internal** prend 80 % du temps CPU. Il faut bien noter que cela ne veut pas forcément dire que cette fonction est « CPU-bound », mais que c'est la fonction qui a consommé (en cumulé) le plus de cycles (qui peuvent être de l'attente d'entrées/sorties).

prof et Intel VTune peuvent être utilisés pour accéder aux compteurs de performances matériels présents sur le processeur et obtenir des informations réelles à un niveau de précision plus fin.

Ces informations sont très importantes pour déterminer quels sont les facteurs limitant d'une application. Nous allons voir maintenant les différentes solutions existantes pour tenter de gagner en performances.

3 Les différents niveaux de parallélisme d'un CPU

Nous avons déjà vu qu'il était possible d'utiliser par exemple plusieurs cœurs du CPU d'une machine pour accélérer un traitement initial effectué avec **grep**. Ce niveau de parallélisme est un niveau parmi beaucoup d'autres, et nous allons essayer ici de décrire l'existant.

Les processeurs ont déjà tout un tas de différents mécanismes permettant de paralléliser certains types d'opérations.



3.1 Instructions SIMD

Les processeurs modernes possèdent aujourd'hui souvent des jeux d'instructions dit vectoriels. Ils permettent d'effectuer des calculs sur des registres de ce type. Ce parallélisme est ainsi caractérisé couramment de SIMD (*Single Instruction Multiple Data*). Concrètement, cela est schématisé figure 3, où une opération vectorielle permet d'effectuer quatre additions en une seule opération, permettant un gain théorique de quatre.

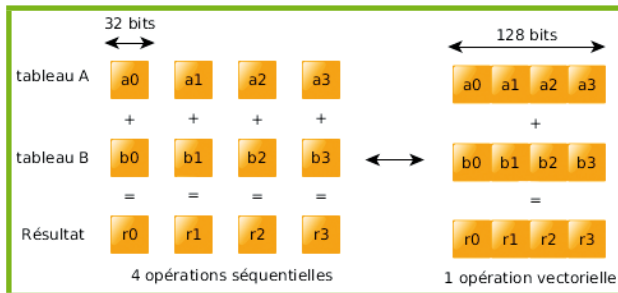


Fig. 3 : Illustration de l'utilisation d'un registre vectoriel 128 bits pour additionner quatre nombre entiers de 32 bits

Nous allons démontrer l'intérêt de telles instructions avec l'exemple ci-dessous :

```
float r[n];
float a[n], b[n], c[n];
for (size_t i = 0; i < n; i++) {
    r[i] = sqrt(a[i]*b[i]+c[i]);
}
```

Le calcul est simple, et consiste à prendre trois tableaux a,b,c de nombres flottants et de calculer **sqrt(a*b+c)**. Afin de bien faire apparaître le schéma figure 3, nous allons « dérouler » cette boucle par quatre itérations et obtenir ainsi ce code :

```
float r[n];
float a[n], b[n], c[n];
for (size_t i = 0; i < (n/4)*4; i += 4) {
    r[i+0] = sqrt(a[i+0]*b[i+0]+c[i+0]);
    r[i+1] = sqrt(a[i+1]*b[i+1]+c[i+1]);
    r[i+2] = sqrt(a[i+2]*b[i+2]+c[i+2]);
    r[i+3] = sqrt(a[i+3]*b[i+3]+c[i+3]);
}
```

Maintenant, si **(a[i], .. , a[i+3])**, **(b[i], ..., b[i+3])**, **(c[i], ..., c[i+3])** et **(r[0], ..., r[3])** sont considérés comme des vecteurs, alors ce code peut être réécrit de cette façon :

```
float a[n], b[n], c[n];
for (size_t i = 0; i < (n/4)*4; i += 4) {
    vecteur_4_floats vr, va, vb, vc;
    va = load_4_float(&a[i]); // va = (a[i], a[i+1], a[i+2], a[i+3])
    vb = load_4_float(&b[i]); // vb = ...
    vc = load_4_float(&c[i]); // vc = ...
    vr = sqrt(va*vb+vc); // Opération qui agit sur ces vecteurs
```

```
store_4_float(&r[i], vr); // vr = (sqrt(a[i]*b[i]+c[i]),
sqrt(a[i+1]*b[i+1]+c[i+1]), ...)
}
```

Le nombre d'opérations est ainsi divisé par quatre.

Plusieurs jeux d'instructions permettent d'effectuer ce genre d'optimisations, et parmi les plus connus, nous pouvons citer les plus couramment utilisés, à savoir le SSE et l'AVX (architecture x86), et le NEON (architecture ARM).

À titre d'exemple, nous allons ainsi nous attaquer à l'écriture du code précédent en SSE. Plusieurs choix s'offrent à nous :

- écrire du code assembleur directement
- utiliser les intrinsèques fournis par les compilateurs : ils permettent d'utiliser des instructions vectorielles directement en C avec des types spécifiques. Intel fourni une très bonne documentation permettant de voir l'ensemble des opérations possibles ici : <http://software.intel.com/en-us/articles/intel-intrinsics-guide>.
- utiliser des bibliothèques spécialisées (voir ci-dessous)

Voici un exemple utilisant les intrinsèques SSE :

```
float a[n], b[n], c[n];
for (size_t i = 0; i < (n/4)*4; i += 4) {
    __m128 vr, va, vb, vc; // __m128 == vecteur de 128 bits contenant
des flottants de 32 bits
    va = _mm_load_ps(&a[i]); // chargement du registre vectoriel va
    vb = _mm_load_ps(&b[i]); // chargement du registre vectoriel vb
    vc = _mm_load_ps(&c[i]); // chargement du registre vectoriel vc
    vr = _mm_sqrt_ps(_mm_add_ps(_mm_mul_ps(va, vb), vc)); // vr =
sqrt((va*vb) + vc)
    _mm_store_ps(&r[i], vr); // sauvegarde du résultat
}
```

Un rapide benchmark nous donne :

```
$ gcc -std=c99 -Wall -O2 -march=native -ftree-vectorizer-verbose=1
vec.c helpers.c -o vec
$ ./vec 10000000
serial: in 35.14910 ms. [...]
sse: in 11.93404 ms. [...]
```

Soit un gain d'environ 3.95 (très proche de 4).

Le problème d'écrire ce code est qu'il devient dépendant de la plate-forme pour laquelle il a été écrit. Par exemple, ce code sous ARM nécessiterait d'être réécrit pour le jeu d'instructions NEON. De plus, la version séquentielle d'origine reste plus maintenable que son homologue vectorisée en SSE. Nous ne parlerons pas non plus du temps nécessaire à faire ce genre de conversions sur des cas plus complexes.

Heureusement pour nous, les compilateurs modernes font, dans certains cas, ce travail pour nous.

Dans le cas précédent, si nous compilons ce code avec **gcc** avec les options **'-Ofast -march=native'**,

une version vectorisée de la boucle va être produite, utilisant les instructions disponibles sur la machine sur laquelle il est exécuté. Cela peut se confirmer en rajoutant l'option **'-ftree-vectorizer-verbose'** :

```
$ gcc -std=c99 -c -Wall -Ofast -march=native -ftree-vectorizer-verbose=1 vec.c
Analyzing loop at vec.c:6
Vectorizing loop at vec.c:6
[...]
vec.c:6: note: created 3 versioning for alias checks.

vec.c:6: note: == [...] Setting upper bound of nb iterations for epilogue loop to 2

vec.c:6: note: LOOP VECTORIZED.
vec.c:4: note: vectorized 1 loops in function.
```

Ainsi, si le benchmark précédent est relancé avec ces options de compilations, nous obtenons :

```
$ ./vec 10000000
serial: in 11.65700 ms. [...]
sse: in 11.74307 ms. [...]
```

Soit une implémentation plus ou moins équivalente (en terme de performances) que notre vectorisation à la main !

Cette vectorisation automatique règle le problème de devoir transformer notre code à la main, mais possède toujours certains désavantages :

- le binaire compilé reste dépendant de l'architecture ciblée : la contrainte du support d'un jeu d'instructions spécifique est rajouté
- les passes d'optimisations utilisées par les compilateurs sont efficaces pour des codes réguliers (comprendre des nids de boucles sans accès bizarres) mais se cassent vite les dents sur des codes réels. Le compilateur d'Intel fait référence dans le domaine.

Voici un exemple de code difficile à analyser pour un compilateur :

```
void minmax(const uint32_t n, float const* buf, [...])
{
    size_t idx_min = 0, idx_max = 0;
    float min = FLT_MAX;
    float max = FLT_MIN;
    for (uint32_t i = 0; i < n; i++) {
        const float v = buf[i];
        if (v < min) {
            min = v;
            idx_min = i;
        }
        if (v > max) {
            max = v;
            idx_max = i;
        }
    }
    [...]
}
```

Ce code comporte des parties vectorisables (comme les comparaisons), mais gcc nous informe que :

```
$ gcc -c -O3 -march=native -ftree-vectorizer-verbose=2 -Wall -std=c99 minmax.c

Analyzing loop at minmax.c:10
minmax.c:10: note: reduction used in loop.
minmax.c:10: note: Unknown def-use cycle pattern.
[...]
minmax.c:10: note: reduction: not commutative/associative: idx_max_3 = max_15 > max_39 ? idx_max_40 : idx_max_37;
[ ... ]
minmax.c:10: note: not vectorized: unsupported use in stmt.
minmax.c:10: note: unexpected pattern.
minmax.c:5: note: vectorized 0 loops in function.
```

Le stockage des indexes des valeurs minimums et maximums semble poser problème. Pour les curieux, une version vectorisée à la main et commentée de ce code est disponible ici <https://github.com/aguinet/misc-examples/blob/master/vectorisation/minmax.c>.

Du point de vue du développeur, chaque méthode a ainsi ses avantages et inconvénients :

- implémenter un code vectorisé à la main permet d'avoir (si bien fait) de très bonnes performances mais implique un coût de développement non négligeable et du code difficile à relire et maintenir. Il devient de plus dépendant d'une plateforme spécifique.
- laisser le compilateur vectoriser le code, mais cela implique que certains cas ne seront pas pris en compte. Si des performances absolues ne sont pas recherchées, cela n'est pas vraiment un problème. Le binaire reste lui par contre dépendant d'une plateforme.

Nous pouvons tout de même citer des projets qui essaient de régler les soucis ci-dessous : les développeurs de la bibliothèque C++ NT2 **[4]** développent boost::simd, qui permet d'exprimer de manière plus portable du code vectorisé **[5]**. Pour le problème de la portabilité d'un binaire, d'autres projets permettent d'émettre du code vectoriel à la volée (JIT) suivant l'architecture sur laquelle il est lancé. On peut citer par ex. Vapor SIMD **[6]**.

Une dernière solution à ce problème est de fournir, dans le binaire final, plusieurs versions du code : la version originale et les versions optimisées pour différents jeux d'instructions, et de faire le choix dynamiquement à l'exécution.

3.2 CPU multi-cœurs

La montée en fréquence des processeurs étant devenue de plus en plus problématique (c'est la fameuse fin de la loi de Moore), les constructeurs ont commencé à intégrer plusieurs cœurs au sein

d'un même CPU. Cela permet d'avoir plusieurs fils d'exécution exécutés en parallèle. Concrètement, dans les architectures actuelles, certaines parties du processeur sont partagées entre les cœurs, comme nous pouvons le voir figure 4.

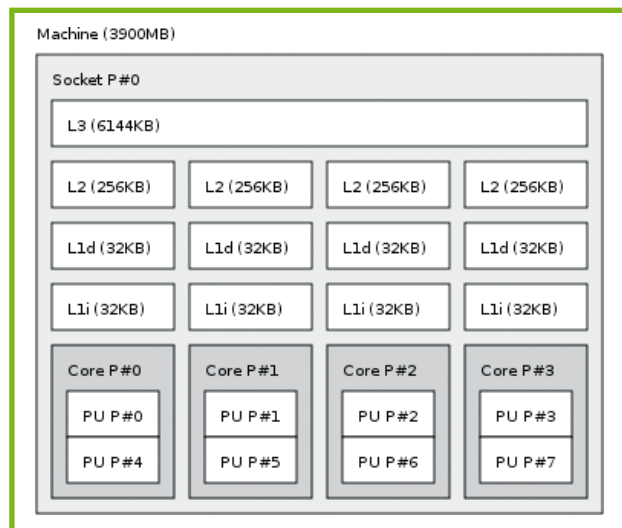


Fig. 4 : Architecture des différents cache d'un CPU. Copie d'écran du logiciel « hwloc-ls » sur un Core i7 première génération

Les caches de données, qui permettent un accès rapide aux données accédées récemment, sont spécifiques à chaque cœur pour les niveaux 1 et 2. Le cache de niveau 3 est partagé. Nous reviendrons par la suite sur les performances de chacun de ces caches, leurs intérêts et désavantages.

Pouvoir exploiter ces différentes unités de calculs n'est malheureusement pour nous développeurs pas automatique. Beaucoup d'algorithmes utilisés aujourd'hui ont été pensé pour des processeurs mono-cœur, et il faut donc repenser la façon dont sont implémentés ces processus.

Heureusement, des bibliothèques et extensions de compilateurs existent afin d'aider le programmeur dans cette tâche difficile.

Commençons par citer le standard OpenMP, utilisable en FORTRAN/C/C++², qui ajoute des directives sous forme de pragma et une bibliothèque implémentant les fonctions nécessaires à cette extension.

Un exemple simple peut être le suivant :

```
float r[n];
float a[n], b[n], c[n];
#pragma omp parallel for
for (size_t i = 0; i < n; i++) {
    r[i] = a[i]*b[i]+c[i];
}
```

Comme nous pouvons le voir, ici l'ajout d'une simple directive indique que cette boucle peut être exécutée

sur plusieurs threads. Plusieurs options sont possibles, comme la spécification du nombre de threads ou le type d'ordonnancement (schedule) utilisé :

```
#pragma omp parallel for num_threads(4) schedule(static)
for (size_t i = 0; i < n; i++) {
    r[i] = a[i]*b[i]+c[i];
}
```

Les différents types de scheduler permettent de définir la façon dont est distribué le travail à travers les différents threads. Le plus simple à comprendre est le « statique » : chaque thread va avoir une part égale du nombre d'itérations à effectuer. Cela suppose que toutes les itérations de la boucle d'origine sont équivalentes en terme de temps de calcul. Le détail des différents scheduler ainsi que plein d'informations passionnantes peuvent être trouvés dans la norme : <http://www.openmp.org/mp-documents/OpenMP3.1.pdf>.

Une autre bibliothèque très utile pour la programmation multi-cœurs est la bibliothèque C++ « Intel Threading Building Blocks » (TBB) d'Intel [7]. Contrairement à OpenMP, aucune extension du compilateur n'est nécessaire, ce qui peut s'avérer utile si ce dernier ne supporte pas OpenMP³. Le code OpenMP ci-dessus peut être par exemple écrit avec TBB comme ceci (en C++11) :

```
tbb::parallel_for(tbb::blocked_range<size_t>(0, n),
    [&](tbb::blocked_range<size_t> const& r) {
    for (size_t i = r.begin(); i != r.end(); i++) {
        r[i] = a[i]*b[i]+c[i];
    }
});
```

La référence et les exemples de la documentation [8] permettent de voir tout le potentiel que peut offrir cette bibliothèque, comme les concepts de pipeline ou de description de graphes.

Pour illustrer plusieurs techniques utiles et problèmes qui peuvent être rencontrés lors de la parallélisation d'un algorithme, nous allons prendre l'exemple du calcul d'un histogramme de nombres flottants. Le problème est le suivant : soit une liste de flottants entre 0 et 1 quelconques en entrée, nous voulons calculer leur répartition suivant des intervalles de même taille. La taille de ces intervalles va donc définir leur nombre. Le programme peut être récupéré ici : <https://github.com/aguinet/misc-examples/tree/master/histogram>.

Plusieurs variantes sont présentées : la version séquentielle d'origine, et deux versions parallélisées avec OpenMP.

La version séquentielle est la façon la plus « naturelle » d'écrire ce calcul :

```
for (size_t i = 0; i < n; i++) {
    // Calcul de l'index dans le buffer de l'histogramme
    const size_t idx = floor(pts[i]/size_interval);
```

```
// Incrément de l'entier correspondant
hist[idx]++;
}
```

Pour une taille d'histogramme de 32Ko, sur un Xeon E3-1245v2, ce calcul est effectué en moyenne à 965Mo/s.

« Naturellement », cette boucle for peut être parallélisée de cette façon :

```
#pragma omp parallel for
for (size_t i = 0; i < n; i++) {
    [...]
    hist[idx]++;
}
```

Le soucis ici est que si deux threads vont écrire en même temps au même index du tampon **hist**, alors le résultat n'est pas prévisible. En effet, l'opération d'addition effectuée ici n'est pas atomique. Lors de cette opération, il se passe ceci :

- récupération de la valeur de **hist[idx]** en RAM (étape 1) ;
- ajout de 1 à cette valeur (étape 2) ;
- stockage de cette valeur dans **hist[idx]** RAM (étape 3).

Entre chacune de ces opérations, un autre thread peut par exemple intervenir entre l'étape 1 et 2 et stocker une nouvelle valeur qu'il avait calculée. Ainsi, la valeur que notre thread mettra en étape 3 sera erronée.

Des instructions existent au sein des CPU afin de palier à ce problème et de spécifier que ces 3 opérations doivent être effectuées de manière « atomique ». OpenMP dispose d'une directive pour spécifier cela, et l'implémentation devient tout « simple » :

```
#pragma omp parallel for
for (size_t i = 0; i < n; i++) {
    const size_t idx = floor(pts[i]/size_interval);
    #pragma omp atomic
    hist[idx]++;
}
```

Avec ce nouveau code, en utilisant les 4 cœurs du précédent Xeon, nous obtenons un débit moyen de 1650Mo/s, soit un gain d'environ 1.7x.

Les opérations atomiques permettent un gain rapide pour un coût d'ingénierie faible, mais elle force une forme de barrière mémoire, elles ont donc un coût. L'utilisation d'un tampon intermédiaire par thread peut s'avérer plus efficace. La dernière version de l'algorithme utilise cette approche, et consiste globalement à :

- allouer un tampon d'histogramme local à chaque thread ;
- laisser chaque thread faire ses calculs dans son propre tampon (plus besoin d'opérations atomiques) ;

- calculer l'histogramme final à partir des histogrammes intermédiaires.

L'implémentation complète de cette approche est disponible dans le dépôt github ci-dessus.

Avec cette approche, nous obtenons, sur le même CPU, un débit moyen d'environ 3550 Mo/s, soit un gain d'environ 3.7x.

Le lecteur curieux (et qui a suivi) se demande alors : est-il possible d'atteindre le gain de 4x théorique ? La cruelle réponse reste « non », et pour plusieurs raisons :

Premièrement, nous sommes ici face à ce qui est communément appelé une « réduction ». En effet, les données d'origine (une liste de nombres flottants dans notre cas) sont calculées sous une forme « réduite », à savoir un histogramme de leur répartition. On remarquera ainsi qu'une somme de nombres entiers, le calcul d'un minimum et/ou maximum de nombres flottants ou le rendu d'une image vectorielle sont aussi des réductions. Si cette réduction est associative (une somme de nombres entiers l'est), une approche classique lors d'un calcul parallélisé (que ce soit avec des instructions SIMD, du multi-CPU, du clustering et/ou un mélange de tous) est de calculer des résultats intermédiaires et de les « réduire » ensuite au résultat final. Cette dernière étape est celle qui ne pourra pas forcément être exécutée sur toutes les unités de calcul en parallèle. Le gain ne pourra ainsi jamais être de 4. À noter qu'OpenMP apporte le support de certains types « simples » de réduction, comme l'addition :

```
int sum = 0 ;
#pragma omp parallel for reduce(sum:+)
for (int i = 0 ; i < n ; i++) {
    sum += array[i] ;
}
```

La bibliothèque TBB apporte un support plus généralisé grâce à la fonction **tbb::parallel_reduce**.

Deuxièmement, des effets de cache peuvent être la cause d'un autre goulot d'étranglement.

Pour conclure, il est important de ne pas essayer de réinventer la roue et d'utiliser autant que possible ce genre de bibliothèques. Des erreurs limitant les performances potentielles peuvent être vite commises, comme par exemple ce code issue du programme **fastgcd** permettant de calculer rapidement des PGCD sur un grand nombre de valeurs (disponible ici : <https://factorable.net/resources.html>). Dans le fichier **fastgcd.c**, la fonction **iter_threads** parallélise le contenu d'une boucle avec ce code :

```
void iter_threads(int start, int end, void (*func)(int n)) {
    int n = start;
    pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```


Abonnez-vous !

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

Consultez l'ensemble de nos offres sur : boutique.ed-diamond.com !

6 Numéros de MISC

6 Numéros de MISC + 2 HORS-SÉRIES



42€*

au lieu de 53,40 €*
en kiosque

Économie
11,40€

Économisez
plus de 20%*

* Sur le prix de vente unitaire France Métropolitaine



51€*

au lieu de 71,40 €*
en kiosque

Économie
20,40€

Économisez
plus de 25%*

* Sur le prix de vente unitaire France Métropolitaine

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

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez MISC chaque mois chez vous ou dans votre entreprise.
- Économisez 11,40 €/an ! (soit plus de 1 magazines offerts !)

4 façons de commander facilement :

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

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

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

Voici mes coordonnées postales :

Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

- Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.
- Je souhaite recevoir les offres promotionnelles de nos partenaires.



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

Vos remarques :

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

Tournez SVP pour découvrir
toutes les offres d'abonnement



Abonnez-vous !

→ Tous les abonnements incluant MISC :

offre LM



42€*
au lieu de **53,40€****
en kiosque

6 NOS

Économie 11,40€

L'OFFRE INCLUS MISC (6nos)

offre M+



51€*
au lieu de **71,40€****
en kiosque

6 NOS
+ 2 HORS-SÉRIES

Économie 20,40€

L'OFFRE INCLUS : MISC (6nos) + ses 2 Hors-Séries

NOUVEAUTÉ 2014

TOUS LES HORS-SÉRIES DE GNU/LINUX MAGAZINE ET LINUX PRATIQUE PASSENT

EN **GUIDE !**
POUR LES DÉCOUVRIR
RENDEZ-VOUS SUR :

boutique.ed-diamond.com



offre T




198€*
au lieu de **277,10€****
en kiosque

Économie 79,10€

L'OFFRE INCLUS : MISC (6nos), GNU/Linux Magazine (11nos), Open Silicium (4nos), Linux Pratique (6nos) et Linux Essentiel (6nos)

offre T+



299€*
au lieu de **411,20€****
en kiosque

Économie 112,20€

L'OFFRE INCLUS : MISC (6nos) + ses 2 Hors-Séries, GNU/Linux Magazine (11nos) + ses 6 Guides, Open Silicium (4nos), Linux Pratique (6nos) + ses 3 Guides et Linux Essentiel (6nos)



Consultez l'ensemble de nos offres d'abonnements sur : **boutique.ed-diamond.com**

Vous pouvez également commander par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

→ Nos Tarifs <small>*entendent TTC et en euros</small>	F	OM1	OM2	E	RM
	France Métro	Outre-Mer		Europe	Reste du Monde
M Abonnement MISC	42 €	50 €	62 €	54 €	58 €
M+ Abonnement MISC + 2 Hors-Séries	51 €	62 €	77 €	68 €	73 €
T Abonnement GLMF + MISC + OS + LP + LE	198 €	253 €	325 €	276 €	300 €
T+ Abonnement GLMF + GLMF HS (6 Guides) + MISC + MISC HS + OS + LP + LP HS (3 Guides) + LE	299 €	382 €	491 €	415 €	448 €

* OM1 : Guadeloupe, Guyane française, Martinique, Réunion, St Pierre et Miquelon, Mayotte

* OM2 : Nouvelle Calédonie, Polynésie française, Wallis et Futuna, Terres Australes et Antarctiques françaises

MA FORMULE D'ABONNEMENT :

Offre	Zone	Tarif
<input type="checkbox"/> M		
<input type="checkbox"/> M+		
<input type="checkbox"/> T		
<input type="checkbox"/> T+		

Exemple :
Je souhaite m'abonner à l'ensemble des magazines + tous les Hors-séries/Guides et je vis en Belgique. Je coche donc l'offre **T+** (la totale avec tous les Hors-Série/Guides) puis ma zone (E), le montant sera donc 415 euros.

J'indique la somme due : (Total)

€

Je choisis de régler par :

Chèque bancaire ou postal à l'ordre des Éditions Diamond

Carte bancaire n° _____

Expire le : _____

Cryptogramme visuel : _____

Date et signature obligatoire





```

void *thread_body(void *ptr)
{
    double start = now();
    for (;;) {
        pthread_mutex_lock( &mutex );
        int i = (n)++;
        pthread_mutex_unlock( &mutex );
        if (i >= end)
            break;
        func(i);
    }
    [...]
}
pthread_t thread_id[NTHREADS];
for (int i=0; i<NTHREADS; i++)
    pthread_create(&thread_id[i], NULL, thread_body, NULL);
[.]
}

```

Ici, chaque thread va attendre la disponibilité d'un mutex pour incrémenter un compteur permettant de savoir quelle itération faire. Si le coût de ce mutex est équivalent ou supérieur à l'exécution de **func**, alors nos CPU vont passer plus de temps à s'attendre mutuellement qu'à exécuter du code vraiment utile. De plus, à chaque appel de **iter_threads**, les threads sont créés et détruits, ce qui peut être un coût non négligeable si cette opération est répétée au sein d'une boucle.

Une première façon de palier à ces problèmes serait d'éventuellement utiliser des opérations atomiques (voir ci-dessus) et de créer les threads une fois pour toute (ce qui serait un travail non-négligeable), ou alors de tout simplement remplacer cette fonction par :

```

#pragma omp parallel for num_threads(NTHREADS)
for (int i = start ; i < end ; i++) {
    func(i) ;
}

```

3.2.1 Hyperthreading

L'Hyperthreading est une technologie développée par Intel permettant d'avoir deux threads logiques par cœur physique. Au niveau de l'OS, deux CPU sont disponibles (bien que le type de CPU (logique/physique) soit connu de ce dernier). Imaginons ainsi deux threads s'exécutant sur un cœur physique. L'idée est que, lorsque le thread en cours d'exécution sur le cœur est en train d'attendre un I/O quelconque (accès RAM, disque, réseau...), alors l'autre thread prend la main.

Cette technologie peut avoir un avantage lorsqu'un nombre important d'opérations d'entrée/sorties est couplé avec des calculs intensifs. Cependant, pour des algorithmes utilisant beaucoup les caches

des CPU, deux threads utilisant le même cœur (et donc les mêmes caches) peuvent croire qu'ils ont chacun des caches différents et ainsi s'invalider tour à tour les données présentes. Il peut ainsi être important de soit désactiver l'hyperthreading, soit de faire en sorte de n'utiliser que la moitié des CPU logiques disponibles sur une machine pour certains algorithmes.

La librairie **hwLoc** permet par exemple de connaître facilement le nombre de cœurs physiques et logiques d'une machine (le tout de manière portable !).

Conclusion

Cette première partie a permis de découvrir une partie de la théorie derrière les applications « scalable », et une description de ce qu'il est possible de faire avec les processeurs actuels. Les concepts et problèmes abordés peuvent être appliqués à d'autres architectures et applications, ce qui fera l'objet de la deuxième partie. ■

■ RÉFÉRENCES

- [1] <http://www.webrankinfo.com/dossiers/google/chiffres-cles>
- [2] <http://linux.die.net/man/1/dd>
- [3] <http://valgrind.org/docs/manual/cl-manual.html>
- [4] <http://nt2.metascale.fr/doc/html/>
- [5] http://nt2.metascale.fr/doc/html/the_boost_simd_library.html
- [6] http://www.irisa.fr/alf/downloads/rohou/doc/Rohou_CGO11.pdf
- [7] <http://threadingbuildingblocks.org>
- [8] <https://www.threadingbuildingblocks.org/documentation>
- [9] <http://www.open-mpi.org/projects/hwloc/>

■ NOTES

- (1) **Benchmark effectué sur un Xeon E3-1245v2 cadencé à 3.8Ghz, en utilisant l'implémentation d'OpenSSL**
- (2) **Profitons de l'occasion pour faire de la publicité pour le projet Pythran <http://pythonhosted.org/pythran/> qui permet d'utiliser OpenMP pour le langage Python.**
- (3) **Comme Clang 3.4 dans sa version officielle**



HPC, « BIG DATA » : DE LA THÉORIE À LA PRATIQUE (2/2)

Adrien Guinet – adrien@guinet.me - Ingénieur R&D chez Quarkslab

Serge Guelton – serge.guelton@telecom-bretagne.eu - Ingénieur R&D chez Quarkslab,
Chercheur associé à Télécom Bretagne en Compil & HPC

mots-clés : CALCUL DISTRIBUÉ / ARCHITECTURE MATÉRIELLE / STOCKAGE

1 Introduction

La première partie de cet article nous a montré, entre autres, les avantages et inconvénients dont disposent les processeurs multi-cœurs modernes. Nous allons voir ici quels sont les autres moyens permettant de rendre une application scalable. Nous nous intéresserons aux performances de toutes ces technologies, et aux applications possibles au domaine de la sécurité informatique.

2 Les architectures parallèles

2.1 GPU

Les GPU (Graphical Processing Unit) se sont depuis quelques années transformés en GPGPU (General Purpose GPU). Ils peuvent maintenant être utilisés à des fins plus scientifiques que de jouer à GTA V en full HD.

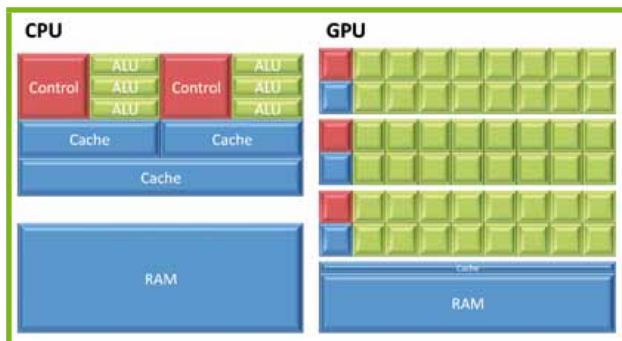


Fig. 1 : Différence d'architecture CPU/GPU. ALU = Arithmetic and Logic Unit, ou Unité Arithmétique et Logique. Image issue de <http://bioinfo-fr.net/gpgpu-le-supercalculateur-du-pauvre>, sous licence CC-by-SA 2.0

L'architecture d'un GPU est très différente de celle d'un CPU classique, comme le montre la figure 1.

Cette architecture est particulièrement adaptée pour des problèmes de type du SIMD. Chaque unité de calcul (ALU) exécute la même instruction sur des données différentes. Ce que le programme décrit est le noyau de calcul, ou kernel, exécuté par chacune des ces ALU.

Plusieurs outils et langages existent pour programmer ces GPGPU. NVIDIA a développé le framework CUDA [1], spécifiques à ces GPU. Le groupe Khronos, responsable d'OpenGL, a créé le standard OpenCL [2], qui décrit un modèle générique pour la programmation de systèmes hétérogènes (composés par exemple de plusieurs CPU et de plusieurs GPU), adopté par ex. par ATI pour ses GPU, et Intel & AMD pour leurs CPU.

Le principal problème de ce genre de périphériques est leur bus d'accès, généralement le PCI Express. Dans ses dernières versions (3.0), en 16x, ce bus peut transférer jusqu'à 15Go/s [3].

Or les données d'entrée nécessaires au calcul sont souvent localisées... en RAM. Il faut donc les transférer sur la carte graphique, calculer puis... rapatrier le résultat. De ces trois opérations, seule la deuxième, le calcul, est utile. Les autres ajoutent un surcoût. Ainsi, s'il s'avère (après benchmarks, bien sûr) que le CPU est capable de traiter le problème à une vitesse supérieure à la vitesse du bus PCI Express, alors le calcul sur GPU ne permet pas de gagner en performance. Une fois de plus, la loi d'Amdhal nous frappe de plein fouet.

Si au contraire le problème est de type SIMD, qu'il est très gourmand en calcul, et moins en mémoire, il y a de bonnes chances que l'accélération sur le GPU soit importante.

2.2 NUMA

Une architecture NUMA (Non Uniform Memory Access) est un système comportant plusieurs processeurs, chacun possédant son propre bus mémoire. Les

différents processeurs sont reliés entre eux par des bus de type Intel QPI (QuickPath Interconnect), ou AMD HyperTransport. De part cette architecture, pour un CPU donné, les temps d'accès diffèrent suivant la zone mémoire accédée. La figure 2 décrit ce type d'architecture pour quatre processeurs :

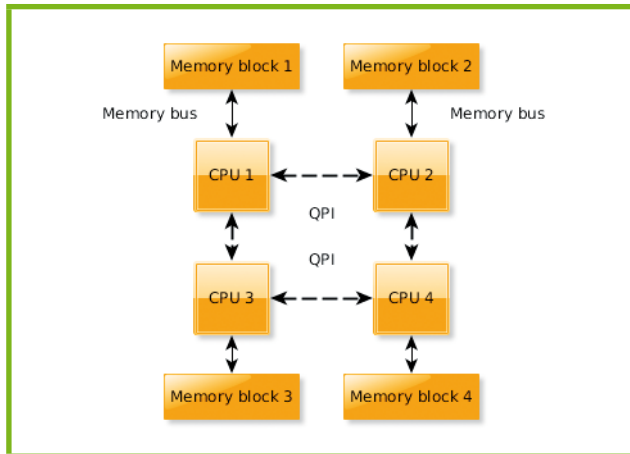


Fig. 2 : Architecture NUMA avec quatre CPU

Lorsque par exemple le CPU 4 veut accéder à des données situées dans le bloc 1, il passe donc par ce réseau d'interconnexion. Le débit de ce bus est assez important pour qu'il ne devienne pas vite un goulot d'étranglement. La bande passante de ce bus est parfois exprimé en GT/s, soit en Giga Transferts par seconde. Cela définit le nombre de transferts effectués sur le bus par seconde. Sachant que celui-ci est bi-directionnel, qu'il est d'une largeur de 20 bits (dont 16 bits de données utiles), cela donne une bande passante théorique de $(T*16/8)*2 = T*4$ (avec T le nombre de transferts par secondes). Ainsi, le processeur Intel Xeon E5-2680 (ci-dessus) possède un QPI offrant une bande passante utile de $8*4=32\text{Go/s}$.

Ces architectures présentent plusieurs intérêts :

- chaque processeur possédant son propre bus mémoire, il est possible, si les données traitées sont bien réparties, d'utiliser pleinement chaque bus, et ainsi de faire passer à l'échelle des algorithmes « memory-bound »
- cela permet d'augmenter au passage la quantité maximum de RAM possible au sein d'un système. Des systèmes se basant sur cette architecture proposés par SGI peuvent supporter jusqu'à 64To de RAM (comme l'UV 2000 [4]).

Pour des applications limités par le débit RAM-> CPU, il faut apporter quelques changements afin de pouvoir exploiter au mieux cette architecture. En effet, sous Linux, la politique d'allocation mémoire par défaut va allouer des espaces mémoires physiquement au plus proche du CPU [5], donc sur son propre bloc mémoire. L'idée est de spécifier au système que cet espace doit être entrelacé entre les différents blocs. Plusieurs façons de faire cela :

- utiliser la libnuma [6] et la fonction `numa_alloc_interleave()`, qui fait exactement cela. Cette fonction est implémentée avec deux appels systèmes : `mmap` (pour un mapping de mémoire privé) et `mbind`, qui permet de définir la politique sur l'espace fraîchement alloué :

```
void* ptr = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED, -1, -1);
mbind(ptr, size, MPOL_INTERLEAVE, NULL, 0, NULL);
```

- définir la politique d'allocation par défaut avec `set_mempolicy` :

```
set_mempolicy(MPOL_INTERLEAVE, NULL, 0);
```

- définir la politique par défaut avec `numactl` (paquet `numa-tools` sous Debian) :

```
$ numactl -interleave=all /path/to/mysoftware args
```

L'avantage de cette méthode est qu'il n'implique aucune modification du programme source.

Changer la politique d'allocation par défaut a le désavantage d'éliminer toute localité pour certains buffers, et ainsi surcharger le bus QPI inutilement (et impliquer potentiellement une perte de performance). Cela est malheureusement à voir au cas par cas suivant les performances de l'application. Par exemple, cette stratégie a été adoptée dans le paquet de 10gen de MongoDB sous Debian¹.

Pour conclure, le prix de tels systèmes peut souvent aller jusqu'à plusieurs centaines de milliers d'euros pour une machine à huit CPU. Ce prix est à comparer avec l'achat de huit machines installées au sein d'un cluster, beaucoup moins élevé.

2.3 Clusters

Traditionnellement, un cluster de machine se programme à travers une bibliothèque de passage de message, le standard de référence étant MPI pour... Message Passing Interface. Ça ne s'invente pas. Chaque machine doit alors gérer ses communications avec l'ensemble des autres machines, encapsuler des messages associés à des signaux, un vrai régal. Cette méthode a fait ses preuves pour de nombreux codes scientifiques, mais reste une approche assez bas niveau. Gérer à la main l'ordonnancement de tâches hétérogènes n'est pas forcément à la portée de tout le monde.

Dans cette partie, nous allons vous présenter deux projets qui abordent le problème à plus haut niveau. Libre à vous de descendre aussi bas qu'il vous plaira !

2.3.1 Celery

Celery (<http://www.celeryproject.org/>) est un projet sous licence BSD permettant d'implémenter une file d'attente distribuée.

Le principe est décrit figure 3 et reste assez simple : d'un côté, des clients vont produire des tâches à réaliser, et les mettre dans une file d'attente. De l'autre côté, des serveurs « écoutent » sur cette file d'attente et vont tour à tour exécuter ces tâches. Dans Celery, la fonction de la file d'attente est en « FIFO » (First In First Out).

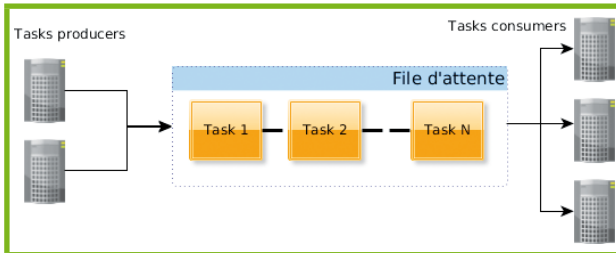


Fig.3 : Celery

Celery distingue deux composants essentiels pour gérer ces tâches : le « broker » et le « backend ». Le broker est responsable de la gestion de la file d'attente, de recevoir les tâches des « producteurs » (Celery utilise le terme de producer) et de les rediriger vers les collecteurs. Le rôle du backend est de stocker (potentiellement de manière permanente) les résultats des tâches exécutées, afin de les retourner aux producteurs à la demande. Le fonctionnement général est décrit figure 4.

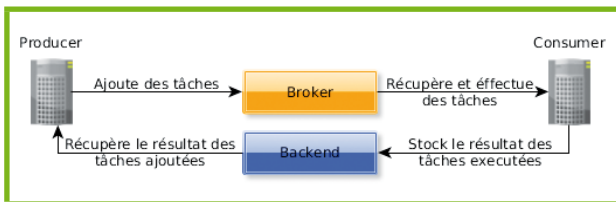


Fig. 4 : Fonctionnement simplifié de Celery

Un des intérêts principaux de Celery est qu'il permet d'utiliser indifféremment différentes technologies pour la gestion de la file d'attente (appelée « broker ») et le stockage des résultats (appelée « backend »).

Nous allons étudier un court exemple afin d'illustrer nos propos (disponible ici : <https://github.com/aguinet/misc-examples/tree/master/celery>). Premièrement, installer Redis, RabbitMQ et Celery avec la méthode de votre choix. La procédure à suivre sous Debian est la suivante :

- Installer Python 3, un serveur Redis (backend) et un serveur RabbitMQ (broker). Attention, par défaut certains de ces serveurs écoutent sur toutes les interfaces réseaux et ne possèdent aucune authentification, donc vérifier que vous êtes en environnement « sûr » avant de lancer ces commandes.

```
# aptitude install python3.2 python3-pip redis-server rabbitmq-server
```

- Si nécessaire, configurer RabbitMQ et Redis pour écouter seulement sur localhost (recommandé pour cette phase de test) :

- Installer Celery, un client AMQP et Redis avec pip :

```
# pip3 install celery redis pyamqp
```

Ensuite, après avoir récupéré le code à l'URL ci-dessus, le lancement du « consumer » se fait grâce à la commande `celery`, de cette façon :

```
$ celery worker -A tasks:app -l info
```

Ici, nous indiquons à `celery` de lancer un « worker » en utilisant l'objet Python `celery.Celery` "app" se trouvant dans le module `tasks`.

Ce module est composé du code suivant :

```
from celery import Celery

app = Celery('tasks', backend='redis://localhost/', broker='amqp://localhost/')

@app.task()
def task_sum(numbers):
    return sum(numbers)

@app.task()
def task_mul(x, y):
    return x*y
```

La première étape consiste à configurer Celery en lui indiquant le backend et broker utilisé. Ensuite, l'objet `app` créé va permettre de rajouter de nouvelles tâches. Le worker `celery` sait donc quelles tâches il peut exécuter, et comment.

Le producer tient lui aussi en quelques lignes de Python :

```
from tasks import task_mul, task_sum
import celery

# Compute 4*4
res = task_mul.delay(4,4).get()
print(res)
```

Le décorateur `app.task` a créé un objet qui permet de créer et de soumettre simplement des tâches. La fonction `delay` va envoyer la tâche au broker et renvoie un objet temporaire de résultat. Sur ce objet, la méthode `get` va attendre que le résultat soit disponible et le récupérer du broker.

Des schémas plus avancés peuvent être aussi décrit avec Celery :

```
# Compute 4*4 + 8*8
res = celery.chord([task_mul.subtask(args=(4,4)),
                   task_mul.subtask(args=(8,8))])
                   (task_sum.subtask()).get()
print(res)
```

Ici, nous lançons deux tâches `task_mul` de manière asynchrone, et indiquons à travers la méthode `chord` que, une fois que ces deux résultats sont finis, ils doivent être passés à la fonction `task_sum`. Le tout en une ligne (un peu longue) de Python.

Ceci n'est qu'une introduction à Celery. Pour aller plus loin, je vous invite à lire sa documentation



(<http://docs.celeryproject.org/en/latest/index.html>) qui est très fournie. Il est aussi facile d'obtenir de l'aide sur le channel IRC #celery sur irc.freenode.net.

2.3.2 Hadoop

Hadoop est un framework open-source qui a pris naissance en 2004 au sein du projet open-source Apache Nutch, un moteur de recherche open-source. Nutch est ainsi le nom original d'Hadoop, et ce dernier a été adopté lorsque la partie traitement des données de Nutch a été externalisée.

Hadoop se base sur l'algorithme map-reduce. Le principe est de diviser les tâches en deux groupes : le mapping et le reduce. Les tâches d'un groupe peuvent être exécutées en parallèle. Le mapping est la première étape du processus : son but est de transformer les données en entrée en couple (clé, valeur). La réduction est une fonction qui va prendre en argument, pour une clé donnée, la liste des différentes valeurs, et va les « réduire » en une seule valeur. En sortie de l'algorithme, une liste de couples (clé, valeur) est créée. Ce principe est expliqué figure 5.

Pour exemple, l'histogramme calculée précédemment peut être décrit selon ce modèle :

- la fonction de mapping correspond au calcul de l'index d'un nombre flottant dans l'histogramme. La valeur associée est 1.
- la fonction de réduire va tout simplement ajouter les différentes valeurs passées pour un index.

De plus, Hadoop se base sur son propre système de fichiers, le HDFS, adapté à la répartition des données sur un cluster de machines. Les données sont réparties par blocs, et ces blocs répartis sur les différents nœuds du cluster. Un système de réplication permet de dupliquer chacun des blocs sur différents nœuds. Ainsi, si une machine appartenant à ce système de fichier n'est plus fonctionnel, alors le système de fichier peut être reconstruit tout en étant disponible et sans aucune perte de données. Le taux de réplication c'est-à-dire le nombre de fois que chaque bloc est réparti - est configurable, afin d'être capable de supporter la perte de un ou plusieurs nœuds. Il y a ainsi un compromis stabilité/capacité de stockage à définir. De plus, lorsque les tâches définies ci-dessus sont exécutées, alors un principe de localité est mis en place : chaque nœud travaille au possible sur les

données qui sont stockées sur ses disques, afin de minimiser les transferts réseaux.

Hadoop est ainsi un très bon candidat pour les applications ayant à la fois besoin d'accéder à des données importantes et d'effectuer des calculs assez coûteux.

3 Les performances du « commodity hardware »

Il est important de connaître les performances des composants des machines actuels afin de pouvoir dimensionner correctement son infrastructure et essayer de prédire quels seront les éléments limitants.

Pour les accès aux données, nous distinguerons deux critères dont nous avons vu l'importance précédemment : les vitesses de lecture et écriture séquentielles, et le nombre d'opération d'écriture et/ou de lectures possibles par seconde (appelées IOPS).

3.1 Disques durs

Pour mesurer les performances bruts des disques durs rotatifs actuels, il existe plusieurs solutions.

En ce qui concerne le débit séquentiel de lecture séquentiel, la commande **dd** sous Linux permet de se faire une idée assez rapidement. En tant que root, lancer :

```
# dd if=/dev/sda of=/dev/null bs=16M count=20
[...]
```

Il faut ici faire attention à deux choses. Premièrement, d'après le manuel de **dd**, celui-ci divise par 1000 et non par 1024 pour le calcul du débit en Mo/s. Ainsi, en divisant par 1024, le débit est donc de 78.9 Mo/s. Deuxièmement, si vous relancer une deuxième fois cette commande, vous allez tomber sur quelque chose comme :

```
# dd if=/dev/sda of=/dev/null bs=16M count=20
20+0 records in
20+0 records out
335544320 bytes (336 MB) copied, 0.0982657 s, 3.4 GB/s
```

Votre disque dur n'est pas devenu soudain plus rapide qu'un SSD (voir section suivante), ceci est juste l'effet du cache disque Linux. En effet, les données viennent ici d'être lues depuis la RAM, et non depuis le disque, car accédées récemment.

Afin de ne pas fausser les résultats, nous devons donc dire au noyau de vider ce cache avant toute mesure. En tant que root, faire :

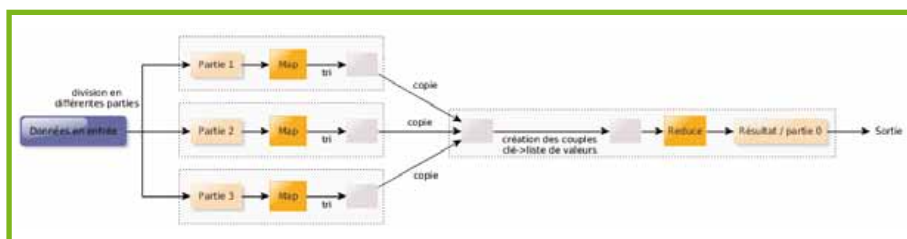


Fig. 5 : Principe du map-reduce

```
# sync ; echo 3 > /proc/sys/vm/drop_caches
```

Cela a pour effet de vider les caches disques. Plus d'informations sur drop_caches peuvent être trouvées dans la documentation du noyau Linux ici : <https://www.kernel.org/doc/Documentation/sysctl/vm.txt>.

Nous venons donc à l'écriture séquentielle. Pour cela, afin d'éviter au départ l'overhead potentiel du système de fichier, nous allons utiliser un disque vierge (ou éventuellement une partition vierge s'il vous reste un peu d'espace), et utiliser de la même façon **dd** :

```
# dd if=/dev/zero of=/dev/sdX bs=16M count=10
[...]
167772160 octets (168 MB) copiés, 2,28166 s, 73,5 MB/s
```

Le débit obtenu reste dans l'ordre de grandeur de celui de la lecture séquentielle. On pourrait se demander quel est l'overhead dû au système de fichiers de cas là.

Par exemple, dans le cas de l'ext4, nous obtenons un overhead d'environ 0.02 % en lecture et en écriture. Au final, cela peut être considéré comme négligeable, ce qui nous intéresse ici étant plus les ordres de grandeur.

Afin de mesurer les performances en accès aléatoire, nous utilisons ce programme : https://github.com/aguinet/misc-examples/tree/master/tools/mem_rand_bench. Il permet de lire de manière aléatoire par bloc de taille définie un fichier ou un périphérique, et rapporte le nombre d'opérations effectuées en moyenne par seconde.

En n'oubliant pas à chaque fois de vider les caches, nous obtenons, en lecture aléatoire de blocs de 4Ko, environ 200 IOPS, et 180 IOPS en écriture. Comme nous allons le voir par la suite, ces performances sont très basses par rapport à ce qui peut se faire avec des SSD ou en RAM, et cela est une des limites des disques durs magnétiques actuels.

Le lecteur est invité à faire ces tests sur sa propre machine afin de se rendre compte par lui-même !

3.2 SSD

Les disques SSD changent un peu la donne, et permettent d'augmenter premièrement les débits en lecture et écritures séquentielles, mais aussi et surtout en terme d'IOPS.

En effet, sur un SSD, les débits généralement atteints avoisinent les 500Mo/s en lecture et écriture séquentielle. En ce qui concerne l'écriture, les performances peuvent avoir tendance à varier avec le temps, dû aux mémoires flashs utilisées.

Outre le débit amélioré, le gros avantage des SSD est l'amélioration du nombre d'opérations par secondes possibles. En effet, tandis que les disques durs classiques plafonnent à 200/250 IOPS, les SSD changent d'ordre

de grandeur et arrivent plutôt vers 75 000 écritures aléatoires par seconde, et 35 000 lectures aléatoires par seconde [8]. Cela dépend beaucoup des constructeurs et quelques benchmarks vous permettront de vous rendre compte des performances réelles des SSD existants.

3.3 RAM

Les accès en RAM eux disposent de propriétés assez intéressantes. Le débit d'accès a augmenté considérablement ces dernières années. Un CPU Xeon de l'architecture Ivy Bridge, disposant de quatre canaux d'accès, peut atteindre un débit allant jusqu'à 10Go/s en lecture/écriture.

Cela peut être vérifié avec un programme très simple comme disponible ici (sous Linux) : <http://www.cs.virginia.edu/stream/ref.html>.

Sur un CPU Intel Ivy Bridge (à double canaux mémoires), nous obtenons, pour un seul CPU, les débits suivants :

```
$. /stream
[...]
-----
Function  Best Rate MB/s  Avg time  Min time  Max time
Copy:    12269.0        0.013058  0.013041  0.013079
Scale:   12301.1        0.013057  0.013007  0.013190
Add:     13496.1        0.017850  0.017783  0.018011
Triad:   13633.4        0.017748  0.017604  0.017938
```

3.3.1 Les canaux multiples

Aujourd'hui, les contrôleurs mémoires (souvent intégrées aux CPU) possèdent plusieurs canaux, qui permettent de multiplier en théorie la bande passante vers la RAM. Pour que cela soit possible, il faut que la machine soit occupée d'un nombre de barrettes de RAM multiple du nombre de canaux, et que celles-ci soient identiques.

L'emplacement de ces barrettes est aussi important. En effet, chaque canal est représenté par une « bank » (généralement avec une couleur différente), et il faut donc au mieux mettre un nombre égal de barrettes par « bank ».

3.3.2 Caches CPU

Les caches des CPU remplissent un rôle important : ils tentent de masquer le coût important des accès aléatoires en RAM.

En effet, comme vu précédemment, les caches sont organisés en plusieurs niveaux. Plus le cache est proche, plus les accès sont « rapides », mais plus le cache est petit. Typiquement, sur les processeurs actuels, les caches de niveau 1 ont une taille de 32Ko pour les données, et 32Ko pour les instructions CPU. Les caches de niveau 2



avoisinent les 256Ko, et les caches de niveau 3 sont souvent partagés entre les cœurs et peuvent atteindre jusqu'à 32Mo. Les caches fonctionnent par ligne de 64 octets, c'est-à-dire qu'à chaque renouvellement/écriture en RAM d'un octet en cache, c'est en fait 64 octets qui sont acheminés/réécrits en RAM.

Nous allons lancer un test simple consistant à écrire de manière aléatoire dans un buffer de taille donnée et de voir le débit moyen d'écriture, afin de voir l'effet de ces différents niveaux de caches. Le programme peut être téléchargé et compilé (sous Linux) ici : <https://github.com/aguinet/misc-examples/tree/master/caches>.

Les résultats sur un CPU Intel Xeon E5-1245v2 (32Ko de cache L1 et 256Ko de cache L2 par CPU, 8Mo de cache L3), en utilisant un seul cœur, sont affichés figure 6.

La première barre bleue correspond à 32Ko (taille du cache L1), la deuxième à 256Ko (taille du cache L2). On voit ainsi, comme pour le cas précédent de l'histogramme, que les performances chutent dramatiquement lorsque la taille du buffer dépasse 32Ko. En effet, les écritures aléatoires vont ainsi se faire en cache L2. Les données présentes en cache L1 sont souvent invalidées et ainsi des transferts mémoires inutiles se produisent. Le même phénomène se produit pour le cache de niveau 3 ensuite. Ce phénomène a été de plus très bien décrit par Ulrich Drepper dans son article « What Every Programmer Should Know About Memory » [7] (lecture conseillée ;-).

3.3.3 Exemple de l'histogramme

La figure 7 montre en vert les performances du calcul de l'histogramme vu précédemment en fonction de sa taille totale en mémoire.

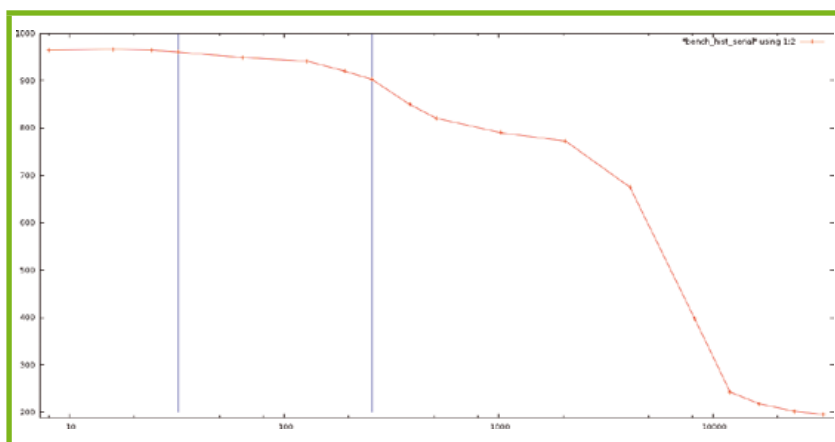


Fig. 6 : Débit d'écriture aléatoire (en Mo/s) en fonction de la taille du buffer d'écriture (en octets, échelle logarithmique)

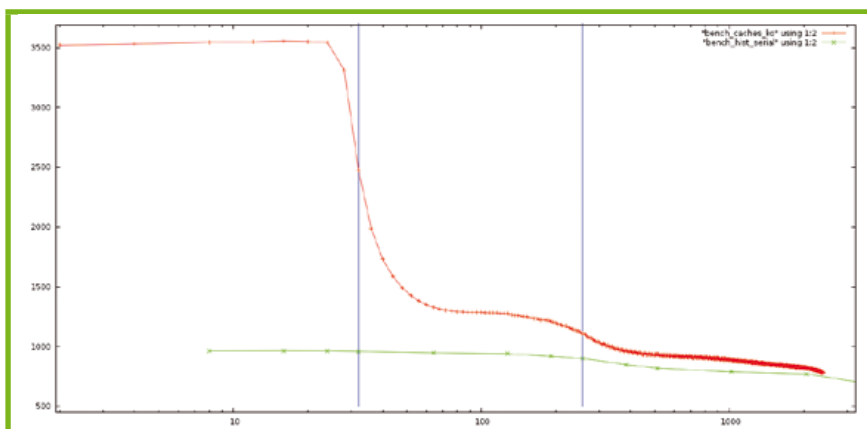


Fig. 7 : Débit moyen de calcul d'un histogramme (en Mo/s) par rapport à la taille de l'histogramme en Ko (échelle logarithmique). Le premier trait vertical correspond à 32Ko (taille de cache L1), le deuxième à 256Ko (taille de cache L2)

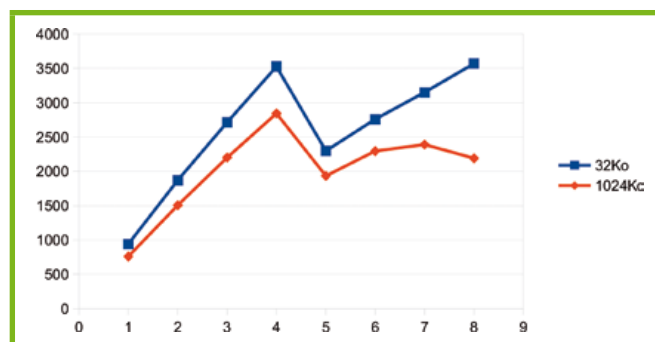


Fig. 8 : Débit moyen de calcul (en MB/s) en fonction du nombre de threads.)

La courbe rouge est la réplique de la courbe présentée figure 7 (débit des accès aléatoires en écriture en RAM). Ce qui est intéressant est que l'on voit bien que les performances de notre application deviennent de plus en plus limitées par les accès aléatoires au cache L2 (puis L3) du CPU.

La parallélisation de cet algorithme peut nous apporter des performances supplémentaires, car les caches L1 et L2 sont propres à chaque CPU (voir première partie). Cependant, l'hyperthreading peut ainsi devenir ici un vrai soucis, car deux threads partageraient les mêmes caches et passeraient ainsi leur temps à les invalider.

Ce phénomène se voit très bien figure 8, où le débit de traitement (toujours en Mo/s) est tracé en fonction du nombre de threads, pour un histogramme de 32Ko et de 1024Ko.

Ainsi, dans ce cas là, l'hyperthreading n'apporte pas de gain conséquent, et peut surtout amener à des performances réduites.

Certaines classes d'algorithmes se comportent très bien vis à vis du cache, et d'autre non. Généralement, le principe de localité est primordial : tant que l'ensemble des thread accède à des données proches en mémoire tout va bien, sinon (comme dans notre exemple), le risque que chaque thread passe son temps à invalider le cache du voisin augmente.

4 Les bases de données « NoSQL »

Les bases de données dites « NoSQL » sont un type de base de données divergeant des bases relationnelles classiques. Les données ne sont pas forcément stockées sous forme de table, et le côté relationnel est généralement mis de côté. De plus, le langage SQL n'est pas le langage natif de ces bases.

Plusieurs types de bases peuvent être distinguées. Une des premières grandes catégories de bases stockent leurs données sous forme de document (sans nécessairement de schémas prédéfinis). Une autre famille se concentre sur un format d'association clé → valeur.

4.1 Stockage de documents

Ce type de base n'est pas tout jeune. Les annuaires de type LDAP, qui ont vu le jour dans les années 90, en sont un parfait exemple. Les données sont stockées sous forme hiérarchique, et l'unité de stockage est un document contenant un ensemble d'attributs. Chaque niveau de hiérarchie peut définir un schéma différent. Ces bases sont généralement optimisées pour des lectures rapides, car beaucoup plus fréquentes que les écritures. Le contrôle d'accès est assez évolué et permet de définir les permissions de lecture et écriture au niveau de la hiérarchie et des attributs.

Comme bases de données plus récentes, MongoDB est parfois cité comme référence dans le domaine. Les documents sont stockés sous format BSON (Binary JSON), une version binaire sérialisée du JSON. Aucun schéma n'est défini, et les documents sont organisés dans des collections elles-mêmes réparties dans des bases de données différentes. MongoDB stocke le plus possible les données en RAM en utilisant le principe de fichiers « mappés » en mémoire. Des API dans plusieurs langages sont disponibles, et restent assez simples à utiliser. Voici un exemple de document MongoDB, qui pourrait être représenté des logs de connexion à un serveur :

```
{
  "_id" : ObjectId("408e291e810c19729de760fb")
  "user" : "root"
  "logins" : [
```

```
{time: , ip: 633024326, result: "Invalid password"},
{time: , ip: 3228333187, result: "Success"}
]
```

De plus, un de ses gros avantages est qu'il scale sur plusieurs serveurs, en utilisant le concept de « sharding » [9]. Techniquement, les documents au sein d'une collection sont réparties entre plusieurs serveurs, utilisant ainsi les ressources en RAM de chaque machine.

D'autres serveurs sont basés sur le même principe, tels que CouchDB.

4.2 Stockage « clé / valeur »

Le stockage clé/valeur est aussi très répandu. Le principe est globalement d'avoir un tableau associatif d'une très grande taille, pouvant être réparti potentiellement entre plusieurs serveurs.

Le serveur Redis est un bon exemple. Il stocke les données en RAM avec un système de sauvegarde sur disque par intervalle régulier. Cela peut poser problème car les données présentes sur disque auront toujours un temps de retard par rapport à la réalité présente en mémoire. Cela peut être acceptable dans certaines conditions, mais il faut en être conscient. Le client en ligne de commande est assez simple et permet de se rendre compte du fonctionnement de ce type de logiciel :

```
$ redis-cli
redis 127.0.0.1:6379> set "result-1" 16
OK
redis 127.0.0.1:6379> set "result-2" 20
OK
redis 127.0.0.1:6379> keys *
1) "result-1"
2) "result-2"
redis 127.0.0.1:6379> get result-1
"16"
```

Le code python associé reste lui aussi assez simple :

```
import redis
client = redis.StrictRedis(host='127.0.0.1')
client.set('result-1', 16)
client.set('result-2', 20)
print(client.get('result-1'))
```

D'autres serveurs de ce type existent, tels que Hypertable ou Cassandra.

5 Écrire une application « scalable »

Répétons-le, la règle d'or est de ne pas partir sur des optimisations prématurées. Faire des prototypes des traitements à effectuer et en déterminer les



performances et les différents goulots d'étranglements est donc primordial. Avoir en tête les ordres de grandeurs évoqués précédemment et les problèmes classiques rencontrés est un plus indéniable pour interpréter les résultats des benchmarks...

Il est aussi important de voir les performances de chaque composants de la machine de test face aux processus testés. Par exemple, pour **grep**, quelle bande passante peut vraiment traiter notre CPU ? Comme vu en première partie, le disque dur devient vite limitant. Nous pouvons donc créer un ramfs afin de pouvoir tester assez sereinement les performances de notre CPU (en gardant en tête l'ordre de grandeur de 10Go/s du débit de la RAM utilisé sur notre système de test) :

```
# mkdir /tmp/ramfs
# mount -t ramfs -o size=1G ramfs /tmp/ramfs
# cp /var/log/auth.log /tmp/ramfs
# time parallel --pipe --block 16M grep -E 'invalid user (\S+)
[...] ' </tmp/ramfs/auth.log >/dev/null
real    0m2.771s
```

ce qui nous donne un débit d'environ 192Mo/s (loin des 10Go/s d'accès en RAM). Ainsi, le dimensionnement d'une machine pour ce type d'application posera ce genre de questions :

- quelles seront les performances d'un CPU plus puissant ? Quel sera son coût ?
- si les performances de traitement d'un tel CPU sont par exemple de 500Mo/s, quelles implications cela aura sur le stockage ? (utilisation de SSD, de RAID0, des deux ?) Et la même question : quel sera son coût ?
- est-il plus avantageux de partir sur une infrastructure cluster ? Quel est son coût matériel ? Quelle sera son coup en terme d'ingénierie et de développement ?

Plusieurs choses sont à considérer ici. Premièrement, c'est (malheureusement ?) souvent le coût financier qui tranche dans ces situations. Pour information, voici un ordre de grandeur du coup du stockage à l'heure actuelle, suivant le support utilisé : (voir tableau ci-dessous).

De plus, on reste souvent confronté à trois types d'applications :

- celles qui demanderont peu de données face aux calculs complexes qui sont effectués, et qui sont donc généralement « CPU-bound ». Par exemple,

un bruteforce d'un hash MD5 (ou autre) demande beaucoup de calcul pour une donnée en entrée assez faible (dans l'ordre de 16 octets).

- celles qui demanderont beaucoup de données mais peu de calcul, et qui sont ainsi communément appelés « IO-bound ». Par exemple, le calcul de l'histogramme précédent est limité par les accès en mémoire.
- celles qui sont entre les deux, et qui sont aussi les plus compliquées à appréhender.

La frontière entre ces différents mondes n'est bien sûr pas claire pour un algorithme donné, et ne dépend pas forcément de sa complexité. Pour s'en sortir, bien que l'expérience ici ait aussi son rôle, obtenir des chiffres sur les performances de son application dès le début d'un projet reste une démarche importante.

N'oubliez pas non plus (pour faciliter le problème) que le code que vous aller écrire doit rester assez évolutif, car les mesures faites aujourd'hui ne seront peut être plus valides demain... et qu'écrire un code spécialisé à outrance pour une architecture, c'est s'exposer à le réécrire pour la prochaine génération de CPU / GPU / ...

6 Applications à la sécurité informatique

MISC restant un journal traitant de la sécurité informatique, il paraît opportun d'appliquer tout ce qui a été évoqué précédemment à ce domaine.

On peut trouver sur Internet plusieurs exemples de projets open-source « haute-performance » lié à la sécurité informatique.

Nous pouvons commencer par le projet truecrack [10], dont le but est d'utiliser la puissance des GPU actuels afin de bruteforcer des conteneurs truecrypt. Le site annonce un gain de performance d'environ 10.

Dans un autre registre, l'actualité récente a vu fleurir beaucoup de logiciels permettant de « scanner Internet » en moins d'une heure. Nous pouvons citer Zmap [11] ou encore masscan [12]. Ils se basent sur le fait que, au niveau d'une machine, le goulot d'étranglement d'un scanner doit être le nombre de paquets par seconde (noté PPS par la suite) que peut générer la ou les cartes réseaux associées. Plusieurs astuces

permettent d'arriver à cela, généralement un bypass de plusieurs couches du noyau afin d'écrire au plus proche du driver de la carte réseau, couplé parfois à l'utilisation de drivers

Technologie	Prix moyen au Go	Exemple
Disque dur 7200 trs/min SAS	~0.12€	WD RE SAS 1 To SATS 6Gb/s (~130€)
Disque dur SSD MLC	~2.31€ (environ x20)	Intel SSD DC S3700 400Go
RAM DDR PC1600 ECC	~20€ (environ x166)	Kingston ValueRAM 8 Go DDR3 ECC



spécifiques. Avec ces astuces, Masscan affiche un nombre de paquets émis avoisinant les deux millions par seconde (sur une carte ethernet 10gb). Cependant, ces exemples sont intéressants car ils se heurtent à un autre goulot d'étranglement (classiquement..) : la capacité qu'ont le ou les routeurs les reliant à Internet à router ces millions de paquets. À titre de comparaison, un routeur de cœur de réseau Cisco peut router dans le meilleur des cas jusqu'à 50 millions PPS [13]. Hors l'utilisateur d'un de ces logiciels n'est généralement pas seul sur son réseau d'opérateur... Envoyer une telle quantité de paquets peut de plus provoquer le remplissage des tampons des différents routeurs, et impliquer que beaucoup de paquets soient simplement éliminés. L'effet inverse de celui recherché est donc atteint. Rappelez-vous ce cher Donald Knuth. Ainsi, effectuer ce genre de scans sur un clusters de machines déployées dans des réseaux différents paraît nécessaire pour atteindre les performances annoncées. Une technologie par « passage de messages » type Celery peut être intéressante dans ce cas là.

Ces scanners introduisent aussi un autre problème intéressant : la génération de nombres aléatoires uniques, nécessaire afin de ne pas scanner linéairement tout Internet. Le problème est évoqué en détail ici [14]. Bien que ce ne soit pas le cas, il est intéressant de vérifier que ce générateur est capable de générer autant de nombres aléatoires par seconde que de paquets à envoyer. Si nécessaire, des optimisations vectoriels et/ou une parallélisation du processus peuvent apporter une solution.

Un autre exemple intéressant d'un autre genre est l'utilisation de l'API streaming d'Hadoop. Imaginons que vous ayez stockés tous vos logs réseaux sur un système HDFS (bien qu'intéressante, la configuration d'un tel cluster ne sera pas discuté ici ; de très bonnes documentations peuvent être trouvées ici [15]), et que vous ayez envie rapidement de lancer (au hasard) la commande **grep** sur ces données afin de chercher (toujours au hasard) les connexions SSH échouées, alors une commande de ce genre peut donner le résultat :

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/mapred/contrib/streaming/
hadoop-streaming.jar stream
-input /path/to/hdfs/auth.log
-mapper 'grep "invalid auth"'
-D mapred.reduce.tasks=0
-output /path/to/hdfs/invalid_auth.log
```

Pas forcément très « sexy », mais facilement scriptable.

Conclusion

Le domaine du High Performance Computing est vaste, et comprendre les différentes technologies qui ont été développées dans ce domaine au fil des

années est une lourde tâche. La création d'applications performantes est un challenge quotidien pour les personnes travaillant dans le domaine. Tout ici n'a pas été évoqué, mais nous avons essayé de donner un aperçu général du domaine (avec des exemples concrets). Les différents pointeurs de l'article peuvent amener les plus curieux à creuser d'avantage les technologies décrites.

N'hésitez pas à nous envoyer toutes remarques, questions et/ou erreurs de notre part aux mails ci-dessous.

Nous laisserons le mot de la fin à Frederick Brooks :

« *Nine women can't make a baby in one month* » ■

■ RÉFÉRENCES

- [1] <https://developer.nvidia.com/category/zone/cuda-zone>
- [2] <http://www.khronos.org/opencl/>
- [3] [http://www.tested.com/tech/457440-theoretical-vs-actual-bandwidth-\[pci-express-and-thunderbolt/](http://www.tested.com/tech/457440-theoretical-vs-actual-bandwidth-[pci-express-and-thunderbolt/)
- [4] <http://www.sgi.com/products/servers/uv/models.html>
- [5] http://man7.org/linux/man-pages/man2/set_mempolicy.2.html (description de MPOL_DEFAULT)
- [6] <http://linux.die.net/man/3/numa>
- [7] <http://www.akkadia.org/drepper/cpumemory.pdf>
- [8] <http://www.tomshardware.com/reviews/ssd-dc-s3700-enterprise-storage,3352-5.html>
- [9] <http://docs.mongodb.org/manual/core/sharding/>
- [10] <http://code.google.com/p/truecrack/>
- [11] <https://zmap.io/>
- [12] <https://github.com/robertdavidgraham/masscan>
- [13] <http://www.cisco.com/web/partners/downloads/765/tools/quickreference/routerperformance.pdf>
- [14] <http://blog.quarkslab.com/unique-random-number-set-computation.html>
- [15] http://hadoop.apache.org/docs/stable/cluster_setup.html

■ NOTE

(1) numactl est utilisé dans le script d'initialisation du service

DÉTECTION DE MENACES

PAR ANALYSE COMPORTEMENTALE DE L'ACTIVITÉ RÉSEAU ET DES UTILISATEURS

Christophe Briguet – christophe@hivedata.com / @cbriguet

Srinivas Doddi – srinivas@hivedata.com / srinivas_doddi@sbcglobal.net



mots-clés : BIG DATA / RÉSEAU / MENACE / DÉTECTION / COMPORTEMENT / PROBABILITÉ / GRAPHE / MODÉLISATION

Cet article présente le concept de modélisation des comportements des systèmes et utilisateurs et montre en quoi il complète les techniques à base de signatures pour détecter les menaces persistantes et avancées.

1 Introduction

Au cours des dernières années, les systèmes de préventions et de détection traditionnels reposant sur des règles et signatures ont vu leur efficacité diminuer avec la sophistication des techniques d'attaques. En effet, poussés par une économie underground en forte croissance et dorénavant comparable en terme de valeur aux marchés de la sécurité informatique, le paysage des menaces a évolué vers une forme plus ciblée et insidieuse. Les attaquants visent maintenant spécifiquement des organisations stratégiques avec pour objectif un gain financier et l'appropriation de propriété intellectuelle. Ainsi, malgré un investissement massif en solutions de sécurité, procédures et expertises, nombre d'entreprises ont été victimes de ces attaques persistantes et suffisamment élaborées pour contourner les différentes couches de protection et de détection. Dans cet article, nous aborderons les limites des solutions actuelles, verrons en quoi le profilage et l'analyse des comportements vient enrichir l'arsenal de défense déjà en place dans les entreprises. Nous nous intéresserons également aux différents modèles mathématiques applicables pour détecter les changements « d'habitudes » ou les écarts par rapport à des groupes d'utilisateurs de référence ou des systèmes similaires.

2 Limite des solutions de protection traditionnelles

La majorité des moyens de défense utilisés actuellement sont limités par le fait qu'ils se basent sur des informations construites à partir de connaissances provenant du passé

ou bien d'experts, par exemple : règles de corrélation d'événements prédéfinis, signatures d'attaques, de virus, de malwares ou de vulnérabilités connues, niveaux de réputation des adresses IP ou des noms de domaine connus, violation de protocoles standardisés, etc. Pourtant des nouvelles techniques dites « bac à sable » fonctionnant sans signatures (basées sur l'exécution de code non testé ou d'origine douteuse pour en analyser les comportements) sont utilisées dans la détection « zero-day ». Les connaissances et le travail d'experts aboutissent à l'identification et la compréhension d'un code malicieux. Cependant, rien n'empêche le concepteur de la menace de détecter la présence du bac à sable et de faire en sorte qu'aucun comportement malicieux ne s'active dans ce cas. Certains malwares sont déjà capables de contourner de tels systèmes...

Ces techniques, même si elles ont prouvées dans une certaine mesure leur efficacité, sont réactives, toujours avec un temps de retard sur les attaquants et ne permettent pas de détecter des menaces inconnues ou qui n'ont jamais été considérées auparavant. Pour preuve, ces grandes entreprises telles que Google, Adobe, Juniper, RSA, Yahoo, Symantec, Northrop Grumman, Sony, Lockheed Martin, Visa, Citigroup et bien d'autres dont on imagine aisément qu'elles aient toutes à leurs dispositions des experts hautement qualifiés et qu'elles ont déployé des solutions de pointes pour se protéger, n'ont pas été en mesure de contrer des campagnes d'attaques s'étalant sur plusieurs mois et dans la majorité des cas (63% selon Mandiant [1]) n'ont pas pu les détecter par leurs propres moyens.

Confronté à cette réalité, une majorité de responsables sécurités estime désormais possible que l'organisation dont ils ont la charge soit continuellement compromise. Afin de savoir dans quelles mesures, ils ont dû trouver de nouvelles approches.

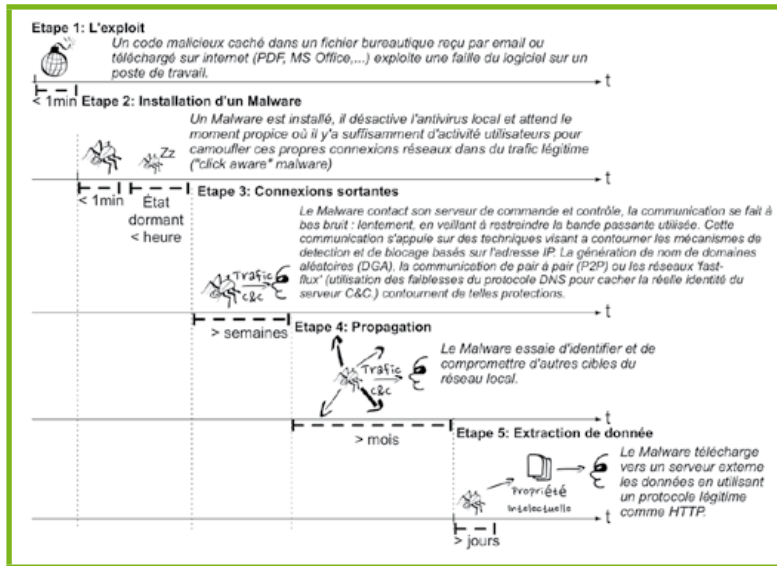


Fig. 1 : Exemple d'un scénario d'attaque persistante et avancée

3 Pourquoi modéliser les comportements?

C'est ainsi que les techniques de modélisation de comportement ont été introduites sur le marché pour compléter les systèmes de détection des menaces avancées. Elles reposent sur le principe que toute entité (individus, machine, système, etc.) peut être caractérisée par ses habitudes et son type de comportement. Ces techniques font l'objet de nombreuses études et sont mis en œuvre dans de multiples domaines tels que la détection de fraude [5] ou la lutte contre le terrorisme. Dans ces cas précis, l'interaction entre individus peut être modélisée sous forme de graphe ou de réseau, pour ensuite détecter de possibles collusions. Les réseaux sociaux ou les sites de commerce en ligne utilisent aussi intensément cette même logique pour recommander de manière très ciblée articles, publicités, offres promotionnelles, etc. Dans le domaine de la sécurité informatique, il est très difficile d'envisager de stopper une attaque que l'on a pas considéré au préalable, il est par contre possible d'identifier de telles attaques de par le caractère anormal de leurs caractéristiques. Par exemple un changement de comportement d'un système dans le temps ou bien une divergence de comportement d'un utilisateur vis-à-vis des autres utilisateurs d'un même groupe peut être considéré comme un signal d'attaque faible ou fort selon la teneur et le contexte du changement.

Les solutions NBA (Network Behaviors Analysis) ont été les premières à analyser le trafic réseau pour détecter les activités ou comportement suspicieux, néanmoins, elles sont limitées à des données agrégées du trafic (Netflow) générées par les routeurs et autres équipements réseau.

Les technologies Big Data (ex: Hadoop, HBase, Map Reduce, SolR, etc.) mis au point ces dernières années

rendent maintenant possible la modélisation de comportements avec un grand nombre de dimensions provenant d'une multitude de sources incluant le trafic réseau bien sûr, mais également entre autres les événements de sécurité (alerte), trace d'audit (log), donnée d'activité des terminaux utilisateurs (appel système, utilisation de ressources, température, etc.).

Par exemple, un log généré par un serveur proxy contiendra des informations concernant les entités concernées (identifiant utilisateur, adresse IP, nom de fichier, etc.), un type d'activité (contrôle d'accès, audit de configuration, etc.), l'action en résultant (autorisation d'accès, mise en quarantaine) et de nombreux détails potentiellement utiles à la création de modèles (nombre de bytes échangés, durée de la session, etc.). Malheureusement les logs pèchent par leur manque d'information et de contexte, tout comme le protocole Netflow, leur utilisation seule reste limitée pour détecter précisément une attaque suffisamment

élaborée. L'inspection en profondeur des paquets réseau (Deep Packet Inspection) bien que plus compliquée à mettre en œuvre pour des raisons à la fois techniques (capture du trafic) et politiques (protection de la vie privée) fournit un niveau d'information beaucoup plus riche. À condition que le trafic ne soit pas chiffré, ou tout au moins puisse être déchiffré si c'est le cas, le décodage des protocoles se fait jusqu'à la couche 7 du modèle ISO permettant l'extraction de nombreuses meta-données utiles au profilage des utilisateurs. Enfin, la combinaison de ces meta-données avec les logs fournis par les machines apporte une vision précise de l'activité d'un système et permette de créer une multitude de modèles.

Inévitablement, alors que la richesse des modèles est améliorée grâce aux données issues du trafic réseau, les ressources nécessaires pour collecter, stocker et analyser toutes ces informations augmentent significativement. La protection des Systèmes d'Information est donc en train de devenir un problème d'analyse « Big Data », où une énorme quantité de données doit être analysée pour en identifier les corrélations et en extraire des motifs complexes et signaux permettant de détecter au plus tôt les anomalies résultant d'une compromission. Ce type d'analyse est ce que nous appelons le profilage des comportements à grande échelle.

4 Profilage des comportements à grande échelle

La modélisation de comportement nécessite le profilage d'un grand nombre d'entités. Cela requière tout d'abord l'identification des entités en question puis l'extraction d'un nombre important de variables



(features) qualitatives or quantitatives associées à ces entités identifiées. Enfin vient la mise en œuvre de modèles statistiques et mathématiques pour modéliser les comportements et détecter des menaces.

4.1 Identification des entités

Cette première étape consiste à identifier les entités ou acteurs dont les comportements sont susceptibles de varier en cas d'attaques. Les vecteurs d'attaques sont de plus en plus diversifiés et avancés, par conséquent le nombre d'entités à considérer dans les modèles est particulièrement important.

Dans le scénario d'attaque suivant, après une phase initiale de reconnaissance, un attaquant envoie un email contenant un fichier PDF piégé à un utilisateur d'une organisation victime. Ce dernier en ouvrant le fichier met en œuvre une vulnérabilité locale qui établit un canal de communication caché avec un serveur hostile de commande et de contrôle (C&C) sur un port spécifique (par exemple HTTP ou IRC). L'attaquant, après avoir pris le contrôle de la machine victime, exécute un certain nombre de commandes et connexions à la recherche

d'autres cibles potentielles (terminaux, serveurs, bases de données, tables, fichiers, etc.). Dans ce scénario, les terminaux, serveurs, bases de données, tables, fichiers, etc. doivent être considérés comme des entités à part entière afin d'en détecter des changements potentiels de comportement. Prenons par exemple le cas d'un fichier, les accès tels que lecture, écriture, renommage, le type, etc. peuvent être capturés et modélisés pour identifier des comportements suspects d'utilisateurs.

De même, en modélisant l'activité du poste de travail il est possible d'identifier une activité transversale anormale : un groupe d'employés travaille souvent avec un pool de ressources communes (ex. serveurs de fichiers, de base de données, imprimantes...) en revanche les interactions entre les postes de travail sont marginales, une augmentation de celles-ci peut être la marque d'un comportement suspect.

Il est important de comprendre que cette approche diffère des approches précédentes consistant à modéliser seulement les connexions (adresse IP, port, protocole, etc.) et que les données extraites de l'inspection du trafic réseau et des logs fournissent un considérable corpus d'entités à modéliser. Dans un scénario typique d'entreprise, le nombre d'entités peut être très significatif et représenter



La CNIL, autorité administrative indépendante, est chargée de veiller au respect, par les entreprises et les administrations publiques, de la loi « Informatique et Libertés ». Elle dispose d'un pouvoir de conseil, de contrôle sur place et de sanction administrative. La CNIL anime également le réseau des correspondants Informatique et Libertés. Elle analyse les conséquences des nouveautés technologiques sur la vie privée et travaille en étroite relation avec ses homologues européens et internationaux.

RECRUTE

UN INGENIEUR D'ETUDES SECURITE DES SYSTEMES D'INFORMATION (H/F)

Activités principales :

- Au sein du service de l'expertise informatique et sous la responsabilité du chef de service, cette personne sera chargée de :
- Réaliser les analyses et études nécessaires en rapport avec les demandes de conseil, les dossiers de formalités (demandes d'autorisation, d'avis ou de labels) et les contrôles, ainsi que dans le cadre du programme annuel d'études et d'expertises informatiques ;
 - Assurer une veille technologique ;
 - Contribuer activement à la mise en forme et à la communication régulière de l'information, en interne, vers les services ou le collège des commissaires, ou en externe ;
 - Rédiger tous documents nécessaires : notes, synthèses, courriers, rapports ;

- Représenter la CNIL au sein des groupes de travail, clubs et réseaux d'experts auxquels la Commission est associée et intervenir aux colloques, congrès et salons auxquels la CNIL contribue, tant en France qu'à l'étranger ;
- Contribuer au développement du laboratoire de la CNIL.

Compétences et qualités requises :

- Ingénieur ou universitaire (Master 2) informaticien, chercheur ou consultant ;
- Connaissance approfondie et pratique des technologies et systèmes mis en œuvre : nouvelles technologies, sécurité des réseaux et systèmes d'information, bases de données, cryptographie, cartes à puces ;
- Bonne pratique de l'évaluation des stratégies et des risques informatiques ;
- Aptitude au travail en équipe ;
- Aptitude à la communication et à la prise de parole en public ;
- Très bonnes qualités d'analyse, de synthèse et d'expression écrite ;
- Capacité à organiser et animer des projets appréciée ;
- Langues : Pratique courante de l'anglais, deuxième langue appréciée ;
- Sensibilité aux problématiques « informatique et libertés ».

Statut et candidature :

CDD (agent contractuel de l'Etat) de 3 ans renouvelable 1 fois. Le poste peut être pourvu par détachement sur contrat (CDD) ou mise à disposition d'un fonctionnaire titulaire d'une des fonctions publiques. Poste à pourvoir à partir de novembre 2013. Rémunération selon profil et expérience. Adresser un CV et une lettre de motivation sous la référence « MSI » par courriel à rh@cnil.fr



plusieurs centaines de milliers d'objets. Dans certains cas, il est également intéressant d'étendre le concept d'entité à des groupes d'entités. Par exemple, un sous-réseau entier peut être traité comme une seule et même entité afin d'en extraire une vue agrégée et d'y déceler des comportements malicieux.

4.2 Sélection des variables

La seconde étape consiste pour chaque entité préalablement sélectionnée à extraire les variables dont les variations caractérisent bien l'activité de l'entité considérée. Par exemple, mesurer la fréquence des requêtes DNS émises par un terminal utilisateur est une caractéristique pouvant révéler une activité potentiellement suspicieuse telle la présence d'un malware membre d'un botnet. En effet, l'activité du logiciel malicieux peut générer un nombre élevé de requêtes par seconde.

Les méthodes utilisées par les attaquants étant en constante évolution, il n'est pas possible d'en définir exhaustivement les variables, il est donc nécessaire de regrouper ces menaces par famille et d'en identifier les variables communes. Modéliser le comportement «normal» de celles-ci permet d'en identifier les écarts qui peuvent être la marque d'actions malicieuses. Cette modélisation peut être obtenue par agrégation sur une plage prédéfinie de temps (qui peut varier de la seconde à plusieurs semaines) des données relatives à chaque variable observée.

La sélection des caractéristiques soulève deux questions:

- Quelles variables sont nécessaires pour détecter la menace?
- Quelle plage de temps doit être considérée pour agréger les variables?

Le caractère discret et insidieux des attaques sophistiquées impose des modèles allant au-delà d'une simple énumération (ex.: nombre distinct de ports, protocoles, etc.), d'une mesure de volume (ex.: nombre de paquets transmis) ou de ratio (bytes reçus / bytes émis). L'utilisation de modèles mathématiques complexes et variés est nécessaire pour les caractériser.

Considérons par exemple le scénario d'attaque suivant connu sous le nom de watering hole. Un attaquant compromet un site web populaire au sein d'une organisation. Par l'intermédiaire de celui-ci, il souhaite accéder à des terminaux utilisateurs. Les machines ciblées seront infectées par un malware lorsqu'elles visiteront le site web piégé et initieront des connexions à destination d'un système contrôlé par l'attaquant. Le changement de comportement d'une ou plusieurs machines (connexion vers un système inconnu jusqu'alors, par exemple) sera traité comme un signal important dans la détection de menace. L'utilisation des réseaux de graphes [6] et de leurs propriétés [4] (caractéristiques intrinsèques) permet de détecter ce type de scénario et de le caractériser en remontant aux propriétés des entités qui en sont parties prenantes. Ainsi des variables telles que les relations entre entités, le nombre de connexions, etc. seront extraites du graphe pour qualifier précisément l'anomalie : voir tableau ci-dessous.

Afin de créer des séries temporelles, ces variables peuvent être consolidées dans une matrice à deux dimensions dans laquelle chaque ligne correspond à une plage de temps et chaque colonne à une variable particulière.

La figures ci-dessous représente une série temporelle du nombre de connexions entre machines sur une plage de 68 heures consécutives issues du jeu de donnée ISCX [2].

Exemple de variable	Description	Influence de l'attaque sur le comportement
Degré (ou valence)	Nombre de connexions d'un noeud	L'augmentation du degré est une indication d'un possible scan réseau. En distinguant degré sortant et entrant, il est possible de savoir si la machine est la cible ou l'origine de ce scan.
Triangles	Un Triangle est un graphe possédant 3 sommets (noeuds) et 3 arêtes (Connexions entre 2 noeuds). Le nombre de triangles est aussi une caractéristique.	L'apparition de nouveaux triangles impliquant plusieurs machines internes et des machines externes est potentiellement signe d'une compromission en cours.
Sunexité (centralité d'interposition)	Pour un nœud donné, somme des chemins les plus courts reliant les autres nœuds du graphe et passant par ce nœud.	Pour les serveurs auxquels se connectent les clients, un indice de sunexité élevé est normal. Une augmentation de cet indice pour un poste de travail peut mettre en évidence une activité malicieuse comme des mouvements latéraux dans une attaque persistante avancée.
Coefficient de clustering d'un noeud	Rapport entre le nombre de connexions existant entre les différents voisins d'un nœuds et le nombre maximal de connexions si tous les voisins étaient connectés entre eux.	Un changement dans le coefficient de clustering d'un serveur de messagerie peut être l'indicateur du chargement d'un malware.

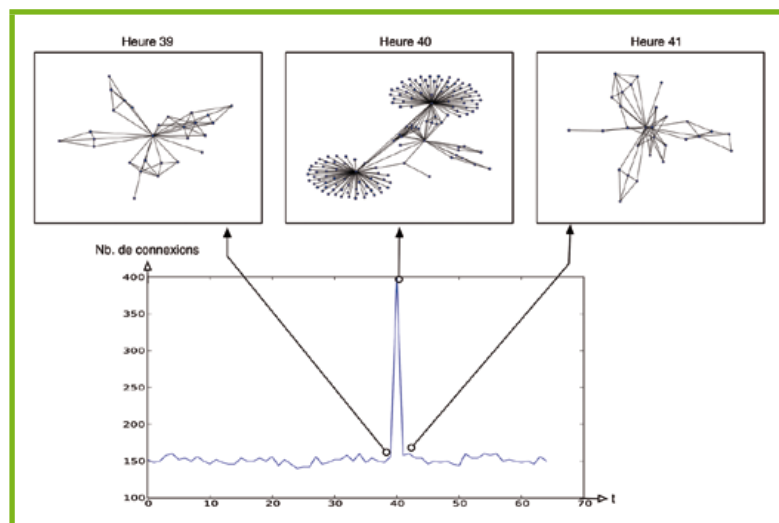


Fig. 2 : Exemple de série temporelle et de réseaux de graphes

Un réseau de graphes peut être généré pour chaque tranche horaire, chaque nœud (ou sommet) représente les machines, elles sont reliées entre elles par un lien si au moins une connexion est établie durant l'heure en question. On remarquera que les graphes pour les heures 39 et 41 sont sensiblement similaires, cependant le graphe de l'heure 40 présente des caractéristiques totalement différentes et met en évidence l'activité suspicieuse de deux machines en particulier. De plus, une analyse complète du jeu de données montre que durant cette plage horaire, une majorité des liens correspondent à une première connexion, ce qui peut être considéré comme indicateur d'une activité malicieuse. Il existe une multitude d'outils dans le monde propriétaire et côté open source pour visualiser des réseaux de graphe, on retiendra Gephi [3] disponible librement et permettant de créer de spectaculaires visualisations.

4.3 Modélisation et analyse des comportements

Maintenant que nous avons sélectionné les variables nécessaires à la modélisation, voyons les différents outils mathématiques à notre disposition pour caractériser les comportements.

Nous verrons comment les lois de probabilité permettent d'assigner une probabilité d'occurrence à chaque valeur d'une variable. Les variables sont de deux natures : discrètes et continues. Les premières, sont dites discrètes car elles prennent un ensemble de valeurs dénombrables : un dé fourni ainsi une valeur discrète. Par contre, les variables dites continues peuvent prendre une infinité de valeur : la position du soleil au court de la journée est une variable continue.

4.3.1 Lois discrètes

Considérons le trafic réseau d'une machine, choisissons un intervalle de temps (une heure par exemple). Pour chaque intervalle, choisissons quelques variables d'étude : comptabilisons, par exemple, séparément le trafic de chaque protocole réseau, le nombre application (de la couche 7 ISO ou vue par l'utilisateur (Facebook, Gmail, ...)) et représentons ces données sous forme d'histogrammes. Une série de 24 périodes consécutives décrit une journée.

Analyser l'activité réseau en continue revient à observer pour chaque machine, les séries de 24 histogrammes pour chaque variable tous les jours. Cela devient rapidement complexe à faire. Une fonction mathématique permet de mesurer la diversité des valeurs constatées pour une variable et certains

intervalles, c'est l'entropie. Pour les courageux, voici la formule :

$$H(X) = - \sum_i Pr(X_i) \log(Pr(X_i))$$

Fig. 3 : Entropie

Dans notre exemple, X_i est le nombre d'occurrences pour chaque protocole réseau i et $Pr(X_i)$ correspond à la probabilité de X_i . De cette manière, nous pouvons caractériser l'ensemble des histogrammes par une simple valeur H . Tout changement significatif de l'entropie dans une série temporelle sera un indicateur de changement du comportement de la machine en question.

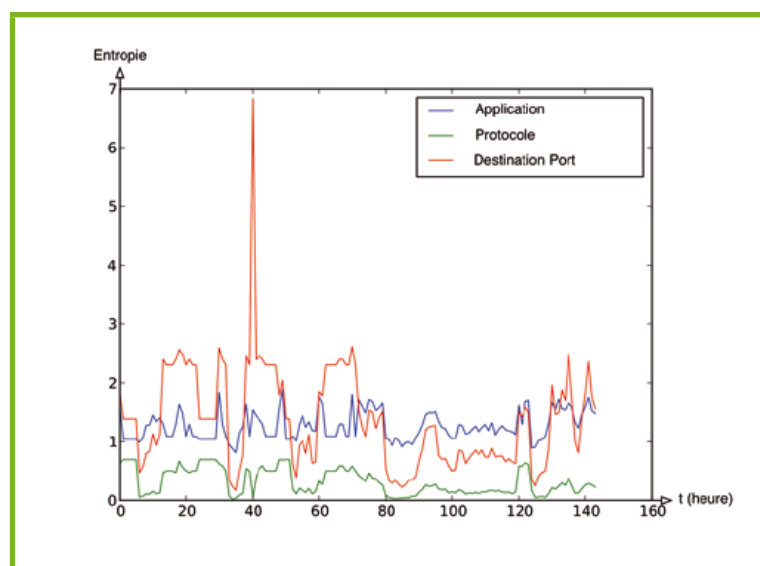


Fig. 4 : Exemple de série temporelle pour des entropies

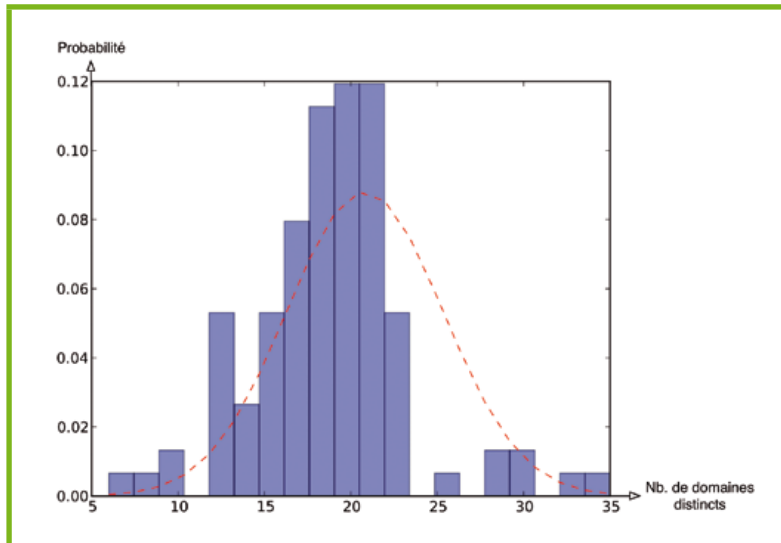


Fig. 5 : Exemple d'histogramme représentant la distribution gaussienne ($\mu = 21$ et $\sigma = 4.5$) des requêtes DNS sur un réseau (agrégées par heure)

4.3.2 Lois continues

Les variables telles que la durée d'une session, la taille moyenne des paquets réseau, la longueur des courriers de messagerie reçus et émis, l'intervalle de temps séparant des communications consécutives, la taille d'un fichier, etc. ont des ensembles de valeurs continues. Les différentes valeurs correspondant à ces variables peuvent être organisées en série temporelle et classées par entité (machine, utilisateur, etc.).

Pour illustrer ce principe, prenons la variable V représentant le nombre de requêtes DNS pour des domaines distincts par heure. L'histogramme ci-dessous décrit la valeur de V sur une période de 140 heures pour une machine donnée. La majorité des valeurs de V est centrée autour de la moyenne (21). Les motifs sous-jacents associés à ce comportement peuvent alors être identifiés à l'aide d'une variété de fonctions appelées densité de probabilité telle que Gaussien, Chi-Square, Poisson, etc. ayant chacune leur propre efficacité et approximation (voir Figure 5).

Attardons nous sur l'une de ces fonctions, la distribution gaussienne qui se caractérise par deux paramètres (moyenne) et (écart type) calculés à partir des données. Moyenne et écart type étant définis, cette fonction permet de calculer, selon la formule

$$Pr(f_i = x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Fig. 6 : Distribution gaussienne

ci-dessous, la probabilité que la valeur soit x . Les puristes diront que nous parlons de la loi normale (voir Figure 6).

Il faut remarquer qu'avec la distribution gaussienne, la valeur la plus probable est centrale : quand x est au milieu (à la valeur moyenne) de la distribution, de même on considère la probabilité faible quand $\mu - 3\sigma > x$ et $x > \mu + 3\sigma$ (équivalent à 0.4%).

Autre exemple connu sous le nom de beaconing: une machine interne compromise communique périodiquement (toutes les heures) via HTTP avec des serveurs distants, ses serveurs C&C. Considérons maintenant la variable f_i comme l'intervalle de temps entre deux requêtes consécutives. En situation normale, les requêtes HTTP GET sont émises à intervalle irrégulier en fonction de l'activité de l'utilisateur et les valeurs f_i sont plutôt aléatoires. En revanche, dans un cas de beaconing un grand nombre de requêtes HTTP GET vers le groupe de serveurs en particulier auront des intervalles réguliers

et ces intervalles apparaîtront comme des exceptions dans une distribution gaussienne.

Bien que ces techniques de calcul de probabilités de distribution permettent de créer des profils caractéristiques à certains comportements, toutes les lois de probabilité ne suivent pas le même schéma : la distribution n'est pas forcément centrée autour de la moyenne par exemple. Il faudra donc choisir la fonction adaptée au comportement de la variable et modéliser les paramètres en conséquence. Sans cette étape cruciale, la modélisation ne suivra pas précisément le comportement réel de la variable.

De plus, la taille de l'échantillon de modélisation influe aussi la pertinence du résultat. L'analyse statistique permet cette évaluation qualitative des données utilisées. Cela permet de définir un intervalle de confiance dans lequel sont décrits les comportements légitimes. En se concentrant sur les données aberrantes, qui sortent de cet intervalle, l'analyste regarde dans la bonne direction pour identifier les comportements malicieux.

4.4 Détection de la menace

Par définition, les attaques persistantes et avancées (APT) se veulent suffisamment furtives pour contourner les règles de détection traditionnelles reposant sur un nombre limité de variables isolées et basées sur des seuils fixés au préalable. Elles pourront cependant être décelées à l'aide des techniques de détection d'anomalies basées sur l'analyse des comportements. Comme vu précédemment, cela consiste à collecter

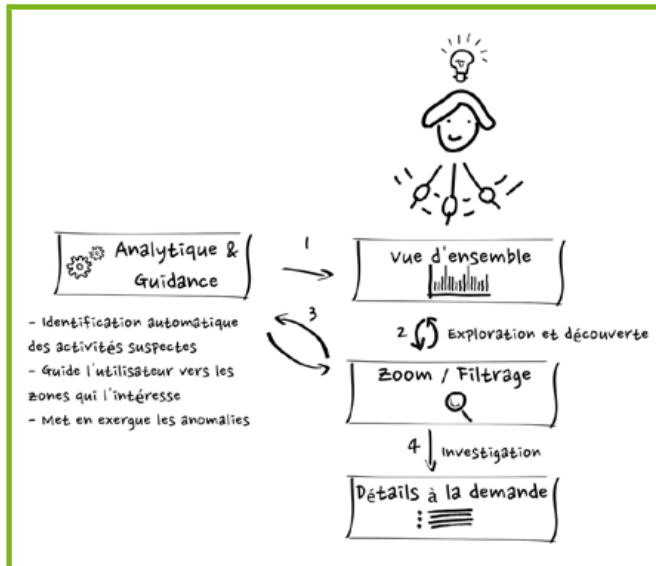


Fig. 7 : Processus d'exploration des données basé sur le « mantra » de Ben Shneiderman [7]

la multitude de signaux faibles produit lors de la brèche de sécurité et à les mettre en évidence par des traitements statistiques, ce qui revient à les transformer en signaux forts afin de les identifier. En pratique, il ne s'agit pas de considérer chaque anomalie comme un incident de sécurité, mais plutôt de donner aux équipes en charge de la supervision de la sécurité d'un SI, les moyens de découvrir les comportements suspects nécessitant une investigation approfondie. En se posant les bonnes questions les analystes pourront explorer les données, les remettre dans leur contexte, comprendre ce qu'il se passe pour finalement déceler l'intrus et réduire la fenêtre de compromission (voir Figure 7).

Conclusion

La modélisation des comportements, à l'aide d'un grand nombre de variables, de l'ensemble des acteurs (humain ou machine) du Système d'Information permet de déceler des propriétés ou changements d'états invisibles jusqu'alors, mais potentiellement indicateurs d'activité malicieuse. Bien que ce type d'analyse nécessitant un grand nombre de données soit permis grâce aux importantes avancées technologiques réalisées en matière d'analyse de données (Big Data), il ne faut pas occulter le fait qu'en matière de sécurité informatique, l'humain sera toujours le meilleur des gardiens. Ainsi, l'implémentation des techniques décrites dans cet article doit permettre avant tout à l'analyste sécurité d'utiliser sa fantastique intuition, sa connaissance du contexte, sa faculté à détecter des motifs (patterns) ou encore sa capacité à deviner le futur. Les signaux faibles ou forts produits sont préanalysés, si possibles

mis en perspective par rapport à une menace connue, mais surtout présentés aux analystes chargés de la supervision de la sécurité du Système d'Information afin de les guider vers les données potentiellement anormales.

Enfin, puisque gérer la sécurité d'un Système d'Information c'est évoluer dans un monde sans règles ni limites, qu'il y a un nombre incalculable d'attaquants, qui peuvent tester les défenses en toute impunité jusqu'au moment où finalement ils réussissent, les entreprises ont tout intérêt à partager collectivement leurs informations de sécurité (Security Intelligence) liées à la menace. Par exemple, les profils de comportement et autres meta-données peuvent alors être mis en commun et analysés non seulement au sein d'une organisation donnée, mais également avec les données issues de leurs pairs. ■

■ REMERCIEMENTS

Un grand merci à nos relecteurs tout particulièrement Jean-Yves Rialhon et Laurent Pautet pour leurs conseils avisés ainsi que leurs corrections.

■ RÉFÉRENCES

- [1] Mandiant. M-Trend 2013. www.mandiant.com/library/M-Trends_2013.pdf
- [2] ISXC. Jeu de donnée. <http://www.iscx.ca/datasets>
- [3] The Open Graph Viz Platform. <https://gephi.org>
- [4] Deepayan Chakrabarti and Christos Faloutsos Graph mining: Laws, generators, and algorithms ACM Computing Surveys (CSUR) 38,1, Article No. 2 (2006)
- [5] Anurat Chapanond, Mukkai S. Krishnamoorthy and Bülent Yener. Graph Theoretic and Spectral Analysis of Enron Email Data. Journal Computational & Mathematical Organization Theory archive, Volume 11 Issue 3, Pages 265-281, October 2005.
- [6] Cliff Joslyn, Sutanay Choudhury, David Haglin, Bill Howe, Bill Nickless and Bryan Oslin. Massive scale cyber traffic analysis : a driver for graph database research. Proceeding GRADES '13 First International Workshop on Graph Data Management Experiences and Systems, Article No. 3, ACM New York, NY, USA 2013.
- [7] Shneiderman. <http://isr.uci.edu/sites/isr.uci.edu/files/speakers/slides/Shneiderman-Feb-2009.pdf>



BIG DATA : ENTRE EXPÉRIMENTATIONS ET CONFUSIONS

Tris Acatrinei - @tris_acatrinei www.hackersrepublic.org

mots-clés : VIE PRIVÉE / SANTÉ / OPEN-DATA / MARKETIN

Apparu dans la presse généraliste au moment des élections présidentielles américaines de 2012, le Big Data semble apparaître comme une solution miracle pour les entreprises, au point qu'un salon lui soit consacré¹. Qualifié de nouvel eldorado des données ou de nouveau pétrole, ce phénomène pluridisciplinaire soulève des questions tout aussi transversales.

1 Les jalons

Le Big Data est un terme qui désigne à la fois un grand volume de données et la gestion de ces mêmes données, synthétisé en une expression, « les trois V » pour volume, vitesse et variété. Proche de cette dénomination et incluse dans celle-ci, les termes de data-mining et de micro-targeting ont émergé pour qualifier la recherche et l'exploitation des données produites. Une application concrète serait la suivante : trouver dans un fichier-client, les personnes qui ont entre 18 et 25 ans, sont de sexe féminins, qui lisent des romans policiers et qui vivent en région parisienne, afin de leur adresser, de façon spécifique et donc ciblée, un contenu, une offre, une promotion. Le terme français qui recouvre ces deux notions est celui de micro-segmentation.

C'est avec la campagne présidentielle de Barack Obama en 2012 que la notion a fait son apparition dans les médias grands publics. En effet, l'équipe de Barack Obama a d'abord procédé à une collecte, en faisant créer des contenus par les citoyens, les a ensuite exploités en s'adressant à des communautés spécifiques. Pour les analystes politiques, les thèmes de campagnes du POTUS (President Of The United States) étaient suffisamment larges pour concerner un grand nombre de citoyens – économie, société – mais l'approche a été suffisamment fine et précise pour que les électeurs se sentent en phase avec le candidat.

En matière de Web, on assiste à un changement de modèle : les personnes créent des données, tant pour leurs activités sur le Web que pour leurs activités

quotidiennes. D'après Gartner, si en 2010, la data « pesait » 3,2 milliards de dollars, en 2015, il est estimé à 17 milliards et de 2012 à 2015, l'agence estime que 4,4 millions d'emplois seront liés au Big Data aux Etats-Unis.

2 Les stratégies

Si on devait schématiser, on pourrait dire qu'il y a d'un côté, les grandes stratégies commerciales et de l'autre, les lignes directrices de l'administration – au sens large du terme.

2.1 Les stratégies du secteur privé

Les stratégies commerciales sont assez simples à définir : vendre encore plus de produits à une population de plus en plus soucieuses du bien-être de son portefeuille. Ethiquement discutable mais cela reste très classique. L'enjeu étant d'arriver à des profils extrêmement précis de personnes, sans que celles-ci en soient conscientes. Comme le dit très bien Jean-Marc Manach, à l'heure actuelle, le souci n'est plus d'avoir son quart d'heure de célébrité mais son quart d'heure d'anonymat². Vivre sans apparaître dans un fichier quelconque est devenu quelque chose d'impossible et, a fortiori, dans un fichier à but commercial.

En matière de marketing, l'idée est de toucher une population ciblée à moindre coût : on ne passe plus

nécessairement pas des campagnes d'affichage grand public dans des zones géographiques déterminées mais on segmente et les annonceurs achètent des profils et des comportements de consommateurs. De façon concrète, dans le commerce, cela aboutit à des offres personnalisées, basées sur les historiques de cartes de fidélité.

« Avez-vous la carte de fidélité du magasin ? » Cette simple phrase, d'apparence anodine, est devenue une sorte de leitmotiv magique dès que l'on sort de chez soi. A l'heure actuelle, rares sont les commerces qui ne proposent pas une forme de contrat d'affiliation permettant d'obtenir réductions et/ou cadeaux. Les prémices du Big Data résident dans ces cartes de fidélité qui ont envahi nos portefeuilles et nos sacs. Nous les oublions bien souvent, pour les ressortir très spontanément dès qu'on nous la demande. Geste anodin, et pourtant qui permet à des sociétés de tracer un profil extrêmement détaillé de nos achats³. Ainsi, le PDG de Catalina explique « à partir des historiques sur les cartes de fidélité, nous déterminons les besoins du client et proposons des promotions personnalisées sur les tickets de caisse ». L'histoire ne dit pas à quel point les offres sont personnalisées.

A terme, l'idée générale est de vendre mais cela peut également servir à une forme de discrimination qui n'a pas été anticipée. En effet, un article récent expliquait qu'en fonction des amis que nous avions sur Facebook, nous pourrions nous voir refuser un prêt bancaire⁴. Il peut être opposé qu'un compte Facebook n'est pas une obligation et que l'on peut très bien vivre sans. Sauf que la majorité des recruteurs actuels commencent par faire des recherches sur les candidats notamment sur les réseaux sociaux et qu'une absence de profils sur des réseaux sociaux est devenu un facteur discriminant. En résumé, vous n'êtes pas obligé d'être présent sur les réseaux sociaux mais cela vous est vivement conseillé. On qualifie ce type de recherches de stratégie multi-canal. LinkedIn est utilisé pour identifier les candidats et les contacter mais Facebook leur permet de prendre des références et valider les entretiens. Aux Etats-Unis, il a été estimé que 93% des recruteurs étaient susceptibles de consulter les profils sur les réseaux sociaux et 42% déclarent avoir changé d'avis sur un candidat après avoir consulté les réseaux sociaux⁵.

Il ne faut absolument pas minimiser le lien entre Big Data et réseaux sociaux. Différentes études marketing et sociologiques soulignent l'essor du premier grâce à l'existence du second et la présence, désormais massive, des entreprises sur les réseaux sociaux est une autre démonstration de l'ampleur du phénomène. Ainsi que l'explique Jean Tillinac dans un article pour Institut Tendances⁶, les clients sont devenus les ouvriers des entreprises. Ces dernières ne sont plus au service du client mais ont mis les clients à contribution. De façon cynique, on pourrait se dire que c'est une version plus optimisée du bouche-à-oreille mais c'est une transposition

NE MANQUEZ PAS NOTRE NOUVEAU NUMÉRO !

ACTUELLEMENT DISPONIBLE !

OPEN SILICIUM N° 8



COLLECTE ET VISUALISATION DE CONSOMMATION ÉLECTRIQUE !

Comment facilement faire des économies sur sa consommation énergétique avec une Raspberry Pi ?



DISPONIBLE JUSQU'AU 5 DÉCEMBRE CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR NOTRE SITE :

boutique.ed-diamond.com



beaucoup plus pernicieuse dans la mesure où elle peut porter un préjudice aux internautes. Mais il n'y a pas que les entreprises du secteur privé, qui s'intéressent au Big Data. En effet, les administrations et les autorités publiques regardent avec attention, comment exploiter cette nouveauté.

2.2 Les stratégies publiques

A la base, ce n'était pas le Big Data qui intéressait les administrations mais l'Open Data. En effet, dans un objectif de transparence, le dernier Gouvernement et l'actuel font des efforts pour ouvrir les données aux citoyens, afin de leur donner ou leur redonner, une place active au sein de la vie politique. Encore une fois, le président Barack Obama aux Etats-Unis avait été précurseur puisque la Maison Blanche avait lancé un programme de ce type dans le courant de l'année 2009. Avec Etalab, on avait une adaptation française. En permettant aux citoyens de se réappropriier les données et les informations, on leur redonnait une part de pouvoir, ce qui revient à une adaptation plus moderne que l'actuelle CADA – Commission d'Accès aux Documents Administratifs – et théoriquement plus accessible. Mais les récents débats sur la transparence ont montré qu'« il y a une grande résistance de l'administration » à libérer certaines informations. L'une des raisons de cette réticence provient du coût : la plupart des accès à certains fichiers comme un extrait de K-Bis, sont payants et si certaines personnes acceptent de payer pour obtenir des données qui devraient être accessibles, pour d'autres « il n'y a pas de raison que l'on paie deux fois nos impôts⁷ ».

Sur le plan de la sécurité, l'idée est de détecter les signaux faibles afin de les anticiper avant qu'ils n'agissent. En France, d'après les informations publiques, la Direction Générale de la Sécurité Extérieure collecterait des signaux électromagnétiques et les compilerait sans que toutes les finalités ne soient connues. Si la Suède procède déjà depuis quelques années à cette collecte avec le FRA, elle possède néanmoins un garde-fou qui encadre très strictement son action, en l'espèce, le SUIN.

De ce que l'on voit, les entreprises privées et le secteur public partagent une vision relativement positive sur le Big Data. Néanmoins, il existe une Pomme de Discorde entre eux : la santé.

3 Big Data = big pharma ?

L'autre « danger » qui alimente considérablement le Big Data réside dans les recherches que nous faisons quotidiennement. A ce jour, même si certaines initiatives telles que Seekers commencent à percer, le grand public reste désespérément fidèle à Google. Mais

à partir du moment où nous disposons d'une adresse Gmail, toutes nos recherches sont associées à ce compte, et archivées dans un historique. Pire encore, ces données sont utilisées dans la nouvelle version du moteur de recherche pour nous suggérer d'autres recherches, à travers le Google Graph Knowledge qui prétend répondre aux questions que les internautes ne sont pas encore posées, incitant surtout à naviguer plus longtemps et à se promener de pages en pages. Si on prend l'hypothèse d'un internaute qui fait une recherche sur un phénomène médical, des suggestions en rapport avec sa première recherche lui seront faites de même que des publicités très ciblées. Google n'ayant aucune raison d'être plus angélique que Facebook, rien n'indique que les données de recherches des internautes ne sont pas déjà vendues à des entreprises dont des compagnies d'assurances médicales et des mutuelles, d'autant que Google spécifie dans ces conditions générales d'utilisation (les CGU) que la société se réserve le droit de revendre les données qu'elle collecte.

Pour le cabinet McKinsey⁸, le système de santé américain actuel est déficitaire et pourrait économiser jusqu'à 100 milliards de dollars en procédant à de l'analyse de données médicales, ce qui aurait pour but – avoué – d'optimiser la consommation de soins, que ce soit la consommation de médicaments ou celle d'actes médicaux. Si l'intention peut paraître louable, elle n'en est pas moins dangereuse car elle enferme l'humain dans un schéma mathématique. Voici un exemple concret : Antoine Gourevitch, consultant au Boston Consulting Group, explique que la première cause d'échec des thérapies est la non-prise du traitement par les patients et qu'un système de comptage permettrait d'assurer un suivi mais également une modulation des primes d'assurances. Dans cette hypothèse, on ne traite plus l'humain en tant qu'individu malade mais en tant que malade tout court et on inverse les paradigmes de la médecine moderne, qui consistent à traiter un patient dans son ensemble. La juriste Michèle Guillaume-Hofnung explique dans différents écrits l'importance de la dénomination entre les termes de patient, de malade et d'homme malade et donc les différences philosophiques et juridiques de ces termes.

En France, nous sommes encore loin du schéma américain. En effet, la santé reste, d'une certaine façon, un monopole d'Etat, puisque la CNAM – Caisse Nationale d'Assurance Maladie – refuse l'accès à son fichier de feuilles de soins électroniques, avec le soutien du Ministère de la Santé et de l'industrie pharmaceutique. L'argument principalement avancé pour justifier ce refus est la protection de la vie privée des patients. En ouvrant les données, on risque d'arriver à un croisement des informations et donc, à un profilage⁹. Or, le Code de la santé publique, en ces articles L.1110-2, L.1110-3 et article L.1110-5 souligne que le malade a droit au respect

À NE PAS MANQUER !

APPRENEZ À PROTÉGER VOTRE VIE PRIVÉE !

MISC Hors-Série N° 8



GARDEZ LE CONTRÔLE DE VOS DONNÉES !



DISPONIBLE CHEZ VOTRE MARCHAND DE
JOURNAUX ET SUR NOTRE BOUTIQUE EN LIGNE :
boutique.ed-diamond.com



de sa dignité, à un égal accès aux soins, les plus appropriés et les plus efficaces. En ouvrant l'accès aux fichiers des feuilles de soins tels que réclamés, notamment par des assureurs et des comparateurs de service de santé, on prend le risque de créer une médecine inique.

4 Le silence des juristes

Il existe trois sites qui peuvent servir de référence dans le monde juridique français : Dalloz, Legalis et Village Justice. Sur ces trois sites, une recherche d'analyses juridiques portant sur le Big Data donne peu de résultat. Que ce soit en matière de droit social, plus spécifiquement sur l'éventualité de poursuites d'un candidat envers un recruteur pour discrimination basé sur un profil de réseau social, en droit civil sur les questions de chaînes de contrats ou en libertés fondamentales, sur la question de la protection de la vie privée.

À l'exception d'un seul article écrit par Jean-Emmanuel Ray, prônant la création d'un droit commercial de l'exploitation des données personnelles, sur le site Dalloz, il n'y a rien.

Comme il a été évoqué précédemment, la question de la prise en compte d'une présence sur les réseaux sociaux peut avoir un caractère discriminatoire. Si aujourd'hui, cela concerne surtout les cadres dans les directions marketing et de communication, la tendance actuellement amorcée laisse présager qu'un emploi pourrait être conditionné par une présence sur le Web. A quand un salarié qui serait victime de vexations parce qu'il n'a pas « liké » la page de son employeur ? Fort heureusement, tout cela reste pour le moment de l'ordre du cas d'école. Mais dans la mesure où les tribunaux ont, d'une certaine manière, décomplexé le « stalking » ou traque furtive des salariés sur les réseaux sociaux, il n'y a qu'un pas à franchir¹⁰.

En droit civil, il existe ce que l'on appelle des chaînes de contrats, on les retrouve beaucoup dans le domaine du bâtiment et de la construction. Un particulier va contracter avec un artisan pour la rénovation de sa maison qui va lui-même faire appel à un autre artisan pour que ce dernier s'occupe spécifiquement des problèmes liés à l'électricité. Vous l'aurez compris : toute la subtilité des chaînes de contrats réside dans la responsabilité. Qui le particulier va-t-il attaqué si la maison brûle à cause d'une installation électrique, qui aurait normalement dû être mise aux normes, et qui, défaillante, a provoqué l'incendie ? En matière de données, la question reste entière.

Reprenons notre ménagère qui va faire ses courses dans sa supérette favorite. On lui propose une carte de fidélité, elle accepte, souscrit au programme et la

supérette va faire appel à un sous-traitant pour gérer les données collectées. Si le sous-traitant vend les données ou est victime d'une attaque informatique, contre qui notre brave ménagère va-t-elle se retourner ? On objectera que la question pouvait se poser avant le Big Data mais le volume des données est devenu tellement conséquent que certaines des entreprises font appel à des sous-traitants dont les clients ignorent totalement l'existence et contre lesquels, ils ne peuvent pas se retourner. En effet, un reportage passé sur France 2 avait fait la lumière sur cette question et plus récemment, le directeur général des ventes de HP France a reconnu que les entreprises ne savaient pas exploiter le Big Data, sous-entendant que le traitement des informations était externalisé¹¹.

Par ailleurs, en cas de liquidation judiciaire, la question est tout aussi problématique et l'exemple de Virgin a été très révélateur.

Lorsque Virgin France a mis la clef sous la porte, la FNAC s'est portée acquéreuse du fichier client, sans pour autant que les dits clients soient consultés. Le fichier en lui-même ne représente « que » 1,6 millions de clients et la FNAC avait fait savoir qu'elle respecterait le droit des anciens clients de Virgin à ne pas figurer sur le fichier client qu'elle avait racheté. A ce jour, je n'ai reçu aucun message de la FNAC me permettant de m'opposer à cette vente de données personnelles.

Plus vicieux encore dans cette chaîne de contrats, la FNAC a précisé que les données seraient exclusivement utilisées par elle et par les sociétés filiales du groupe. En substance, non seulement les anciens clients de Virgin vont recevoir les offres de la FNAC mais aussi toutes les offres de Kering¹² anciennement connu sous l'acronyme PPR. Autant d'acteurs dans la chaîne sur lesquels le client n'a aucun droit de regard sauf à supprimer l'adresse email qui a permis la création du compte. Mais cette action ne suffira pas à effacer les données.

C'est la persistance même de ces données qui pose problème : leur non-effacement automatique entraîne une forme de profilage qui est une violation manifeste de la vie privée. En matière de données personnelles, le droit français rend obligatoire l'intervention du titulaire des données pour procéder à l'effacement des informations le concernant. Par défaut, les informations concernant une personne continuent à s'accumuler, permettant non seulement l'établissement d'un profil commercial, social, professionnel et médical très précis mais aussi un véritable suivi dans le temps. De la même manière, les entreprises disent ouvertement qu'elles stockent des données mais ne les exploitent que si les clients ont donné leur accord alors que c'est le principe de collecte et de stockage qui posent un souci - et non l'exploitation pris isolément¹³. Même les techniciens énoncent que le traitement de la donnée en volume n'est plus un problème puisqu'il existe des solutions techniques pour traiter la donnée



en masse. Par ailleurs, si les serveurs qui hébergent toutes ces informations ne sont pas sur le territoire français, le citoyen français ne pourra pas légalement agir.

Il ne semble y avoir que l'Union Européenne pour s'inquiéter de ce « nouveau pétrole de l'économie » mais aucune directive concrète n'a émergé pour le moment. A Bruxelles, les instances communautaires travaillent actuellement sur un projet de réglementation, qui poserait un cadre juridique clair à cette problématique. Le but est de faciliter la vie des entreprises en allégeant les formalités préalables mais elles seraient beaucoup plus contrôlées, notamment sur la question du droit à l'oubli¹⁴.

En France, la loi Informatique et Libertés de 1978 était très bien écrite mais, elle n'est plus du tout adaptée aux enjeux du Big Data. On notera que la CNIL travaille sur une charte de bonnes pratiques, en partenariat avec la fédération des assurances mais une charte n'a aucune valeur juridique obligatoire et ne constitue donc pas une base juridique contraignante.

Autre scénario : la vie privée et la santé que nous avons abordé précédemment et plus précisément, la protection de la vie privée, dans laquelle est incluse, les questions de santé. En France, le système juridique est très protecteur de la vie privée et tout ce qui est relatif à la santé est protégé au titre de l'article 9 du Code Civil. Or, à la lecture de la jurisprudence très stricte en la matière, constitue une atteinte illicite à la vie privée d'une personne la divulgation de faits intéressants sa santé. Cette jurisprudence est appuyée par une disposition postérieure, datant de 2002 énonçant que la personne [malade] peut refuser à tout moment que soient communiquées des informations la concernant à un ou plusieurs professionnels de santé. La personne peut donner son consentement mais elle peut également la retirer à tout moment, même de façon dématérialisée. On pourrait égrener le code de santé publique, conjugué au code civil et aux textes communautaires sur des dizaines de pages, le constat sera le même : à l'heure actuelle, le droit français et communautaire ne permettent pas l'ouverture des fichiers de santé et inclure une logique ou un processus de Big Data à cette matière serait préjudiciable aux patients.

Enfin, si on se pique de prospective juridique, on peut également envisager un scénario à la Minority Report : un modèle de société dans lequel, les personnes ayant un « potentiel » de commission d'infraction serait détecté très en amont avant même de commettre l'infraction¹⁵.

Tout le monde semble voir dans le Big Data des opportunités économiques, financières et commerciales formidables mais peu de voix s'élèvent pour rappeler que la vie privée fait partie de nos droits les plus essentiels. Certains argueront qu'ils n'ont rien à cacher et que si cela peut leur permettre de faire

des économies, c'est un petit prix à payer. Dans la mesure où la vie privée est philosophiquement indissociable du libre-arbitre, ce type de discours est très inquiétant. ■

■ NOTES

- (1) Voir <http://www.big-data-expo.fr/>
- (2) Lire <https://www.wefightcensorship.org/fr/article/quart-dheure-danonymat-en-ligne.html>
- (3) Lire <http://data.blog.lemonde.fr/2013/08/14/big-data-vos-donnees-en-vente/>
- (4) Lire <http://www.demainlaveille.fr/2013/08/27/bientot-vous-pourrez-vous-refuser-un-credit-bancaire-a-cause-de-vos-contacts-facebook/>
- (5) Lire <http://www.viuz.com/2013/09/09/recrutement-et-reseaux-sociaux-les-usages-se-differencient/>
- (6) Article consultable à l'adresse suivante http://www.persee.fr/web/revues/home/prescript/article/quad_0987-1381_2006_num_60_1_2053
- (7) Voir http://www.lemonde.fr/technologies/article/2013/10/13/la-lente-marche-des-donnees-publiques-vers-l-open-data_3494915_651865.html
- (8) Voir http://www.lemonde.fr/technologies/article/2013/10/14/dix-secteurs-bouleverses-par-le-deluge-d-informations_3495278_651865.html
- (9) Voir http://www.lemonde.fr/technologies/article/2013/10/13/la-sante-un-tresor-trop-bien-garde_3494916_651865.html
- (10) Voir l'analyse sur le rapport entre salariés et employeurs sur les réseaux sociaux : <http://www.juritravail.com/Actualite/reseaux-sociaux/Id/13500>
- (11) Voir http://www.lemonde.fr/technologies/article/2013/10/13/bataille-geante-autour-du-magot-des-donnees_3494909_651865.html
- (12) Voir fr.wikipedia.org/wiki/Kering
- 13 Lire le point de vue d'Harper Reed sur cette thématique <http://www.zdnet.fr/actualites/le-big-data-c-est-de-la-connerie-39790807.htm>
- (14) Lire http://www.lemonde.fr/economie/article/2013/10/13/souriez-vous-etes-traques_3494910_3234.html
- (15) Lire <http://www.bloomberg.com/news/2013-08-14/how-big-data-could-help-identify-the-next-felon-or-blame-the-wrong-guy.html>

QUELQUES PISTES SUR LE RENFORCEMENT DE LA SÉCURITÉ AUTOUR DU PROTOCOLE DE ROUTAGE BGP

Cédric Llorens, Denis Valois

mots-clés : ROUTAGE / CRYPTOGRAPHIE / SYSTÈME AUTONOME / ROA / AS

Le routage fait partie des éléments critiques de la disponibilité des réseaux. Le niveau de protection du routage a été souvent discuté et différentes pistes de renforcement de sa sécurité ont été étudiées. Nous décrirons succinctement le protocole BGP utilisé pour les échanges de routes sur Internet, puis nous détaillerons les mécanismes déjà mis en œuvre ainsi que les diverses approches sécuritaires étudiées.

1 Introduction

Le déploiement de réseaux IP de grande taille a rapidement nécessité la mise au point de protocoles de routage dynamique chargés de déterminer le plus efficacement possible la meilleure route pour atteindre une destination donnée. Par ailleurs, il a aussi été nécessaire de découper le réseau en différents systèmes autonomes (ou Autonomous System (AS)) afin de réduire cette complexité en taille.

Ces considérations ont donné lieu à une classification des protocoles de routage dynamique en deux grandes familles. Les protocoles de routage dynamique IGP (Interior Gateway Protocol) qui permettent d'échanger des informations d'accessibilité et de routage au sein d'un système autonome. Les protocoles de routage dynamique EGP (Exterior Gateway Protocol) qui permettent d'échanger des informations d'accessibilité et de routage entre systèmes autonomes.

Le protocole BGP établit une connexion TCP (port 179) entre deux routeurs et échange d'une manière dynamique et incrémentale les annonces de routes [RFC4271].

Les calculs de routage effectués par BGP utilisent l'algorithme de Bellman-Ford étendu par l'annonce du chemin d'AS complet. BGP n'a pas de vision globale de la topologie de routage et envoie donc uniquement à ses voisins les annonces de routes. Pour notamment

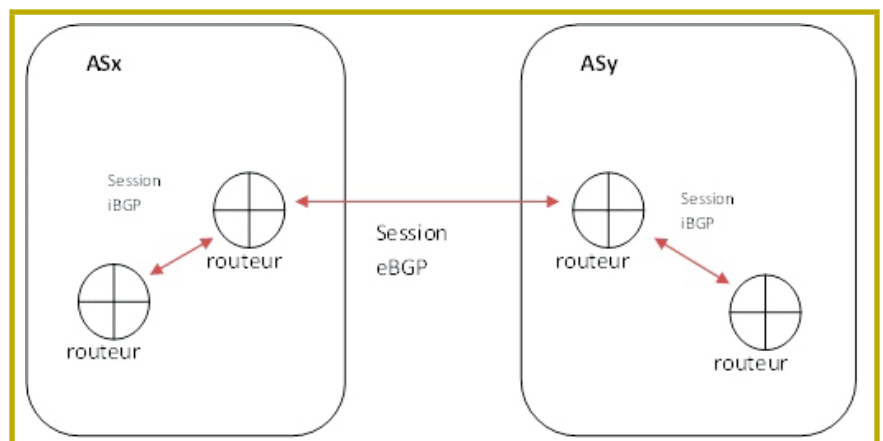


Figure 1 : Architecture BGP



éviter tout bouclage de routes, le protocole BGP gère un attribut contenant l'ensemble des AS traversés.

Les sessions BGP établies entre des routeurs appartenant à des AS distincts sont qualifiées de type eBGP (external BGP), alors que les connexions établies entre des routeurs BGP appartenant au même AS sont qualifiées de type iBGP (internal BGP) comme l'illustre la figure 1.

Sachant que toute attaque ou perturbation du routage peut impacter directement la disponibilité du réseau et de ses services, il est primordial de considérer les protocoles de routage comme des éléments-clés de la sécurité d'un réseau. Il doit être noté qu'il est possible de détourner du trafic par le routage à des fins de vol d'information.

2 Rappels sur quelques mécanismes actuels de sécurité

2.1 Le contrôle des topologies de routage

Les routes apprises par les sessions eBGP d'un système autonome doivent être propagées au sein du système autonome par le biais de sessions iBGP. Il s'agit effectivement de maintenir une vue cohérente de l'ensemble des routes externes au sein du système autonome pour l'ensemble des routeurs. Quels que soient les modèles de topologie iBGP considérés (modèle complet : chaque routeur communique avec tous les autres routeurs, ou réflecteur de routes : chaque routeur communique avec des réflecteurs de routes plutôt que de communiquer avec tous les autres [WIKI RR]), le contrôle des topologies de routage est primordial afin d'assurer la disponibilité du réseau et de ses services.

Ces contrôles de configuration sont possibles grâce à l'outil que nous avons développé HAWK qui permet une analyse des configurations des équipements réseaux [HAWK].

2.2 Le contrôle par les secrets partagés

Le contrôle d'une session de routage BGP entre deux routeurs peut être réalisé par l'option d'empreinte MD5 véhiculée dans les paquets TCP. Il s'agit alors de vérifier en point à point les annonces de routes échangées entre deux routeurs à l'aide d'un secret

partagé ou clé secrète [RFC2385]. Ce contrôle doit être mis en œuvre en priorité sur les sessions eBGP qui sont le plus à risque pour un AS. La configuration de ce contrôle s'écrit de la manière suivante en Cisco :

```
! association du routeur à l'AS
router bgp as
```

```
! partage du mot de passe avec le voisin de routage
neighbor "destination_adresse" password "mot"
```

2.3 Le contrôle par les TTLs

Une autre méthode pour contrôler une session de routage BGP consiste à mettre en place un contrôle du TTL (Time To Live) contenu dans les paquets IP échangés par la session de routage BGP. Ce contrôle doit être mis en œuvre en priorité sur les sessions eBGP qui sont le plus à risque pour un AS. En effet, partant du principe que les sessions de routage BGP entre deux routeurs sont généralement directes, les paquets IP contenant des informations de routage BGP émis par un routeur doivent arriver à l'autre routeur avec un TTL = TTL - 1. Comme une annonce de routes entre deux routeurs correspond à chaque fois à un nouveau paquet IP, le TTL du paquet IP émis sera par défaut égal à 255. Ainsi, si l'autre routeur reçoit des annonces de routes ayant un TTL qui n'est pas égal à 254, il peut en conclure que ce n'est pas le routeur avec lequel il a une session de routage qui a émis cette annonce. La configuration de ce contrôle s'écrit de la manière suivante en Cisco :

```
! association du routeur à l'AS
router bgp as
```

```
! attribution du nombre de saut ip avec le voisin de routage
neighbor "destination_adresse" ttl-security hops "number"
```

GTSM (Generalized TTL Security Mechanism) permet de généraliser cette approche à plusieurs sauts possibles et autres protocoles [RFC3682].

2.4 Le contrôle des annonces de routes eBGP

Les annonces de routes peuvent être soumises à une politique de filtrage définie par l'administrateur d'un système autonome. Cette politique peut à la fois s'appliquer aux annonces de routes reçues par un système autonome (routes transmises à l'intérieur d'un AS) ainsi qu'aux annonces de routes qu'émet le système autonome (routes émises à l'extérieur d'un AS). La configuration de ce contrôle s'écrit de la manière suivante en Cisco :


```
! association du routeur à l'AS
router bgp as

! application d'une liste de filtrage des annonces de routes
entrants et sortantes
neighbor "destination_adresse" prefix-list "nom_liste" in
neighbor "destination_adresse" prefix-list "nom_liste_2" out

! definition des listes de filtrage
ip prefix-list " nom_liste " seq " nombre " permit/deny " adresse "
etc.
ip prefix-list " nom_liste_2 " seq " nombre " permit/deny " adresse "
etc.
```

2.5 Le contrôle des attaques de type Déni de Service

Le protocole BGP ne constitue pas une contre-mesure aux attaques de type Déni de Service vers des destinations clientes, mais peut aider à anticiper et limiter leurs effets [RFC3882]. Dans le cas d'une attaque de type Déni de Service, ce ne sont généralement pas les équipements réseau contenus au sein d'un AS qui sont visés (bien qu'ils soient une cible de choix pour ce type d'attaque, ils se protègent eux-aussi par divers mécanismes), mais plutôt les équipements (serveurs Web, de mail, etc.) de ses clients. Une telle attaque peut alors générer un trafic important qui dans le meilleur des cas saturera seulement le lien d'accès du client, et dans le pire des cas saturera un ou des liens d'infrastructure de l'opérateur de télécommunications. Deux techniques permettent de mitiger de telles attaques :

- BGP et puits de routage (black hole) : Dans le cas d'un système autonome avec de multiples points d'accès vers d'autres ASs, l'attaque peut venir de différentes sources et il n'est pas envisageable d'intervenir sur tous les routeurs d'interconnexion. Il suffit donc d'annoncer dans BGP l'adresse IP destinataire de l'attaque, avec comme point de sortie un puits de routage au lieu de l'interface cliente. Ce puits de routage met alors à la poubelle systématiquement tout le trafic qu'il reçoit. Il est cependant possible de faire plus simple et d'éviter de transporter ce flux inutile. Dans BGP, on va annoncer que le chemin pour atteindre l'adresse IP visée par l'attaque est une interface poubelle du routeur lui-même (null0 pour Cisco par exemple). Une fois que BGP aura propagé cette information, dès qu'un routeur recevra un paquet à destination de l'adresse attaquée, il le détruira. Cette option est brutale car tout le trafic est détruit, et donc ne protège pas réellement le service du client contre l'attaque DOS.
- BGP et puits de filtrage (sink hole) : Les routeurs IP n'ont pas la capacité à analyser le trafic pour

séparer le trafic légitime de celui de l'attaque car cela nécessite d'inspecter le contenu du paquet IP alors que le rôle du routeur s'arrête à la couche IP. L'idée est donc de rediriger le trafic vers un équipement réseau dédié, qui lui aura cette capacité. Dans ce cas, BGP spécifie comme point de sortie l'adresse du « puits de filtrage ». Ainsi, tous les paquets à destination de l'adresse IP attaquée vont passer par cet équipement filtrant. Le système « puits de filtrage » permettra alors de déterminer exactement l'attaque à l'aide d'outils embarqués (Snort, Radware Defense Pro, etc.). Une fois les données analysées, le trafic épuré de l'attaque sera alors envoyé vers l'adresse IP destinatrice. D'un point de vue du routage BGP, on a tout d'abord routé le trafic externe vers le puits au sein de l'AS. Puis le puits route le trafic « nettoyé » vers le client, en général à travers un tunnel afin de cacher ce trafic au réseau (sinon le réseau routerait ce trafic à destination du client de nouveau vers le puits).

2.6 Conclusion

La sécurité du routage existe donc belle et bien à travers de nombreux contrôles mis en œuvre, cependant elle ne s'est pas attaquée aux deux problèmes principaux/fondamentaux du routage qui sont :

- La validité de l'origine d'une route.
- La validité d'un chemin suivi par une annonce de route.

Pour assurer la sécurité de ces deux problèmes, il est nécessaire de passer à la vitesse supérieure en terme de techniques et d'administration comme le détaille les mécanismes tels que: SBGP, soBGP, ROA, etc.

3 Les mécanismes de sécurité étudiés/futurs

3.1 Introduction

De nombreuses initiatives ont vu le jour pour authentifier l'origine d'une route et le chemin pris par une annonce de route. Cependant, aucune de ces solutions n'a été déployée à grandes échelles pour diverses raisons détaillées ci-après.

3.2 sBGP (secure BGP)

La première initiative a été le protocole sBGP (secure-BGP) qui permet à la fois de valider l'origine et le chemin pris par une annonce de route [SBGP].



Ce document est la propriété exclusive de Johann Locatelli (johann.locatelli@businessdecision.com) - 06 janvier 2016 à 09:44

La validation de l'origine d'une route repose sur de la cryptographie à clé asymétrique nécessitant la mise en œuvre d'un système à clé publique où chaque système autonome possède alors un certificat électronique. La structure de gestion de clés proposée permet de traiter les extensions pour les blocs d'adresses certifiant l'appartenance d'une adresse à un AS. Un certificat est donc délivré à chaque organisme propriétaire d'un bloc d'adresse IP par l'entité responsable de l'attribution des adresses IP. En partant du haut, on trouve l'IANA (Internet Assigned Numbers Authority) qui gère l'espace d'adressage d'Internet et l'assignation des blocs de numéros d'AS. Elle est l'autorité de certification (AC) racine. On trouve au second niveau les registres Internet régionaux (RIR, Regional Internet Registry), qui allouent les blocs d'adresses IP et des numéros d'AS dans leurs zones géographiques. Enfin, les registres Internet locaux (LIR, local Internet Registry) qui sont généralement des fournisseurs d'accès à Internet (FAI) formant un troisième niveau.

La validation du chemin pris par une annonce de route repose aussi sur de la cryptographie à clé asymétrique permettant de signer à chaque saut d'AS le chemin émis (avec sa clé privée) vers un autre système autonome comme l'illustre la figure 2 (les signatures s'empilent alors comme les couches d'un oignon).

Le protocole sBGP impose donc de modifier le protocole BGP pour intégrer ces notions d'authentification dans le format des messages. De plus, les équipements réseaux doivent embarquer au sein de leur système d'exploitation les bibliothèques cryptographiques afin de mener les calculs nécessaires. Enfin, chaque session de routage entre deux équipements réseaux est réalisé via le protocole IPSEC pour renforcer la sécurité des échanges de routes entre deux ASs.

L'impact de ces calculs cryptographiques sur le processeur et la mémoire d'un équipement, couplé à l'impact sur le temps général de convergence des routes ont dissuadé les opérateurs à déployer le protocole sBGP.

3.3 soBGP (secure origin BGP)

Une autre initiative soBGP (Secure Origin BGP) de Cisco **[SO-BGP]** veut répondre aux mêmes besoins de sécurité que la solution sBGP mais par l'ajout d'une

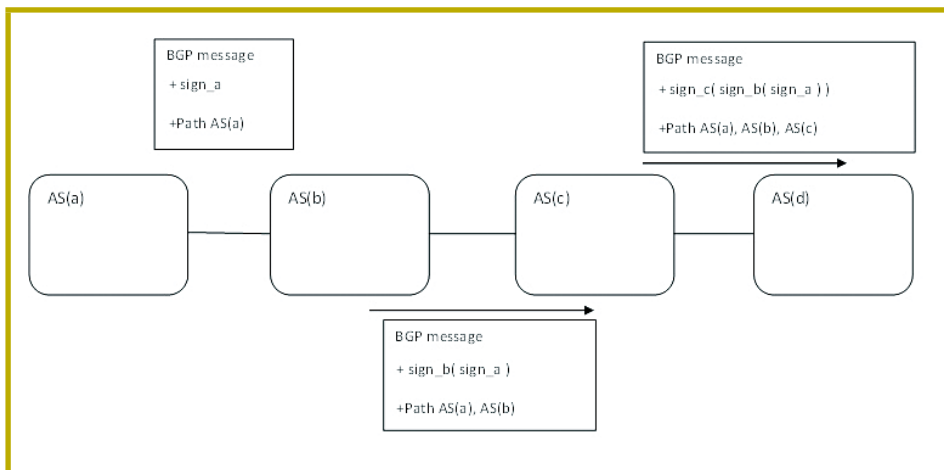


Figure 2 : sBGP et la validation de route

nouvelle primitive BGP (message de sécurité) et en ne dépendant pas d'une autorité centralisé (repose sur une notion de réseau de confiance appelé « web of trust »).

Pour assurer l'authentification, chaque AS génère sa paire de clés et la fait signer par un autre membre de confiance (l'opération de signature de « confiance » est manuelle, non automatisé par le protocole). Ainsi le réseau de confiance se construit et permet la publication de certificats associant des AS à des adresses, mais aussi décrivant une politique pour un AS donné. Ces certificats sont déployés à l'aide d'un nouveau message BGP de sécurité (type 6).

La validation de l'origine d'une route nécessite de mettre en œuvre comme pour sBGP des primitives cryptographiques sur l'ensemble des équipements réseau participant au routage. La validation du chemin nécessite pour chaque équipement de construire en amont les topologies réseaux grâce aux certificats construits préalablement comme l'illustre la figure 3. Il est aussi possible de créer un serveur par AS dont les équipements lui envoient les messages BGP de sécurité (BGP Security) contenant les certificats et c'est ce serveur qui validera les auprès des équipements réseaux les messages BGP de mise à jour (BGP Update).

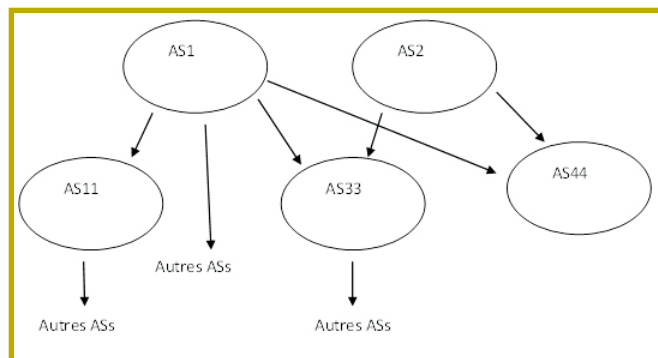


Figure 3 : Topologie des interconnexions des ASs

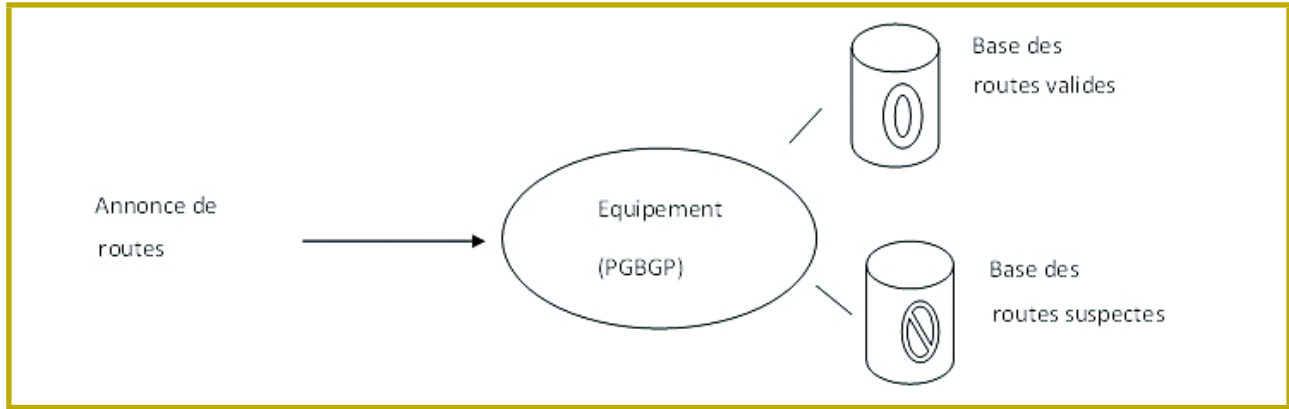


Figure 4 : Processus de PGBGP

Les impacts des primitives cryptographiques, la synchronisation parfaite nécessaire pour valider le chemin associé à une annonce de route grâce aux topologies réseaux (statique comparé à sBGP qui se base sur un calcul dynamique) ont aussi dissuadé les opérateurs à déployer le protocole soBGP.

3.4 Pretty Good BGP

Cette approche ne modifie ni le protocole BGP et ne repose pas sur des primitives cryptographiques, elle implémente seulement dans le système d'exploitation d'un équipement un algorithme assurant la fonction de validation de route (ou sur un serveur déporté de l'AS si on ne veut pas modifier les systèmes d'exploitation du réseau), mais aussi un mécanisme d'alerte informant les administrateurs d'une possible attaque détectée par ce nouvel algorithme **[PGBGP]**.

Pour initialiser notre algorithme, il est nécessaire que l'équipement réseau construise ses tables de routage avec ses différentes sessions BGP. Durant cette phase, aucune détection de route suspecte n'est possible. Une fois réalisé, on peut activer la fonction de détection de routes suspectes.

Cette nouvelle fonction agit lors de (comme illustré à la figure 4) :

- l'annonce d'une route pour un préfixe donné issu d'un AS différent de celui déjà présent dans les tables de routage du client. Un attaquant veut potentiellement voler cette route.
- l'annonce d'une route pour un sous-préfixe donné issu d'un AS différent de celui du préfixe agrégé déjà présent dans les tables de routage du client. Un attaquant veut potentiellement voler une

partie d'un préfixe déjà annoncé, rappelons que BGP privilégie toujours le préfixe le plus faible lors de sa décision de routage.

Dans de tels cas, les routes ne sont pas prises en compte et mises dans un état nécessitant une « validation » par un opérateur. Cette période de « validation » dure un temps t et si elle est dépassée, alors la route est prise en compte et propagée.

Bien que a priori efficace pour maîtriser ce type de problème, elle nécessite une validation coûteuse en terme opérationnelle sachant que la plupart des routes suspectes ne le sont pas (l'attribution de préfixes originés par un AS bougent sans être pour autant des attaques). De plus, ce processus où « l'humain » intervient dégrade fortement la convergence des routes. Ces contraintes ont donc dissuadé les opérateurs à déployer PGBGP. En effet, les clients Entreprises (ou autres) à l'accès changent régulièrement d'opérateurs (changement de contrat, etc.) modifiant en conséquence les routes annoncées dans le processus et générant de nouveaux messages BGP.

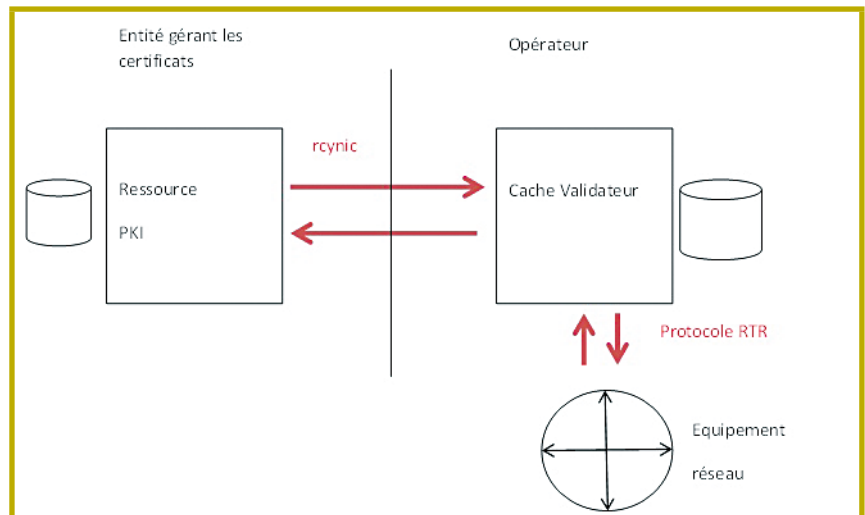


Figure 5 : ROAs et cache valideur

3.5 ROA (Route Origin Authorization)

Cette nouvelle initiative de l'IANA (groupe de travail sur la sécurité du routage inter-domaine (SIDR)) a pour objectif de limiter les impacts sur les équipements réseaux **[ROA_1]**. On ne veut aussi ni modifier le protocole BGP, ni impacter le routeur par des calculs cryptographiques. Enfin, on ne s'attaque qu'au problème de l'origine des routes.

Pour atteindre les objectifs fixés, il est nécessaire de faire tout le travail de validation en amont et d'appliquer aux équipements réseaux qu'une liste de filtrage limitant les impacts processeur et mémoire.

La solution repose sur :

- Une infrastructure de distribution de certificats numériques dont l'objectif est de prouver qu'un AS a le droit d'annoncer un préfixe. Elle s'appelle la RPKI (Resource Public Key Infrastructure) **[ROA_2]**.
- Le ROA (Route Origin Authorization) est un objet signé par une autorité indiquant que « le préfixe y peut être originé par un AS donné x ». Plus spécifiquement, il s'agit d'un fichier signé avec la clé du certificat de l'autorité donnant les AS qui peuvent être à l'origine d'un ou plusieurs préfixes.

L'autorité peut donc publier son certificat et ses ROAs dans des bases de données publiques (RIPE, etc.). A ce stade, il est donc possible de récupérer ces ROAs avec des outils de synchronisation tel que RPKI Validator du RIPE, rcynic de ISC/ARIN, etc. **[RPKI]**.

Ce sont les ROAs qui seront traduits en une liste de filtrage au niveau de l'équipement réseau. Cependant et comme l'injection de la liste de filtrage ROAs en mémoire sur un équipement semble dans bien des cas tout simplement impossible (tous les routeurs ne sont pas forcément puissants), une architecture via un cache-validateur semble donc être requise via le protocole RTR (RPKI/Router Protocol) comme l'illustre la figure 5.

On a donc :

- Un cache/validateur, typiquement un serveur de type Unix que l'on appelle le serveur RTR.
- Le routeur, contenant une instance RTR que l'on appelle le client RTR.

La configuration pour récupérer les informations entre le routeur et le cache validateur s'exprime de la manière suivante en Juniper :

```
/* association du routeur à l'as */
routing-options {
  autonomous-system 64511;
  validation {

    /* description de la session avec le cache-validateur */
    group rpki-validator {
      session 10.0.0.1 {
        refresh-time 120;
        hold-time 180;
        port 8282;
        local-address 10.0.0.2;
      }
    }
  }
}
```

Le cache/validateur ne revoit que les ROAs pour des AS donnés, la décision de routage restant à la décision du routeur via sa configuration comme l'illustre la figure 6.

Lors de la validation d'une route à l'aide des ROAs, trois cas de figures peuvent arriver :

1. la route correspond à un ROA avec une correspondance du numéro d'AS.
2. la route correspond à un ROA avec aucune correspondance du numéro d'AS.
3. aucune correspondance à un ROA.

A ce stade, c'est la politique configurée par l'opérateur sur ces équipements qui décideront de la stratégie à mettre en place. Plusieurs stratégies sont possibles comme celle de type paranoïaque « je n'accepte que les routes dites valides », etc.

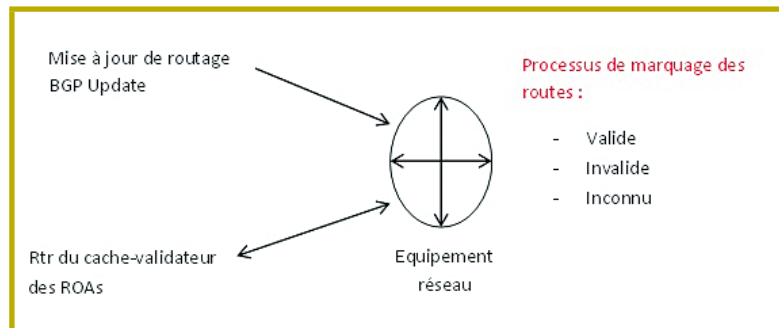


Figure 6 : Validation des routes par les ROAs

L'avantage principal de cette approche est que les impacts sur l'équipement réseau sont acceptables comparés aux calculs cryptographiques nécessaires des autres approches. L'inconvénient est que la mise à jour des ROAs doit être en temps réel pour éviter qu'une route ne soit pas prise en compte bien qu'elle soit légitime. Elle demande aussi l'adhésion de l'ensemble de la communauté, sinon chaque opérateur sera obligé d'accepter les routes non valides par défaut rendant la solution partielle.

4 Les recommandations de l'ANSSI

L'agence nationale de la sécurité des systèmes d'information vient de publier un guide des bonnes pratiques de configuration BGP. Il s'agit d'un guide pratique qui contient des exemples concrets de configuration de type Cisco, Juniper, Alcatel, etc. Après une présentation des différents types d'interconnexion BGP (peering privé entre deux AS, session établie en « multi-hop », etc.), le guide détaille un ensemble de recommandations ainsi que la matrice d'application de ces recommandations avec les différents types d'interconnexion.

Voici quelques exemples de ces recommandations :

- Authentification des sessions (référence 2.2 dans cet article).
- Filtrage des annonces de routes (référence 2.4 dans cet article). Le guide aborde de plus la suppression des AS privés, le filtrage des routes par défaut, le filtrage sur l'AS path, etc.
- La journalisation des informations relatives à BGP à des fins de supervision active (détection de problème de stabilité, etc.).
- L'utilisation de mécanisme pour lutter contre l'usurpation d'adresses IP basé sur la technique dite de l'URPF (Unicast Reverse Path Forwarding). Elle consiste à ignorer un paquet s'il ne provient pas de l'interface que le routeur aurait utilisé pour router la source du paquet.
- Etc.

Le lecteur trouvera une description détaillée de chaque mécanisme dans ce guide qui a aussi pour vocation d'évoluer régulièrement **[ANSSI BGP]**. Sur un plan international, un draft IETF est aussi en cours d'élaboration sur le sujet **[IETF BGP SECURITY]**.

Conclusion

La sécurité du routage porte à la fois sur la sécurisation des sessions et des informations de routage échangées. Bien que la sécurisation des sessions permet de mettre en œuvre de nombreux mécanismes (TTL, MD5, etc.), la sécurisation des informations de routage est plus large et n'a pas atteint sa maturité.

Une analyse transversale montre que la vitesse de convergence des routes est critique et que le futur mécanisme de sécurité ne doit pas la freiner tout en assurant l'origine et le chemin pris par une annonce de route.

Enfin, cette future solution ne peut pas échapper à une administration « lourde » de gestion des adresses,

AS, etc. impliquant aussi une distribution temps réelle avec les réseaux des opérateurs. ■

■ REMERCIEMENTS

Merci à la relecture attentive de Bruno Decraene et Benjamin Caillat.

■ RÉFÉRENCES

[ANSSI BGP] Guides de recommandations, <http://www.ssi.gouv.fr/fr/bonnes-pratiques/recommandations-et-guides/>

[Bellman-Ford] Algorithme de Bellman-Ford, Wikipédia en français (http://fr.wikipedia.org/wiki/Algorithme_de_Bellman-Ford).

[HAWK] Les sources de HAWK sont disponibles sur le site web : <https://sites.google.com/site/tableaubordsecurite/home>

C.Llorens, L.Levier, D.Valois ; B.Morin Tableaux de bord de la sécurité réseau, 3ème édition, Eyrolles, 562 pages, ISBN 2-212-12821-5, septembre 2010.

[IETF BGP SECURITY] J. Durand, I. Pepelnjak, G. Doering, BGP operations and security, IETF, draft, 2014, <http://tools.ietf.org/html/draft-ietf-opesec-bgp-security-01>

[PGBGP] Josh Karlin, Improving BGP by Cautiously Adopting Routes, 2006, <http://dl.acm.org/citation.cfm?id=1318378>

[RFC4271] Rekhter (Y.), Li (T.), A Border Gateway Protocol 4 (BGP-4), IETF, 2006, <http://www.ietf.org/rfc/rfc4271.txt>

[RFC3682] V. Gill, J. Heasley, D. Meyer, The Generalized TTL Security Mechanism (GTSM), IETF, 2004, <http://www.ietf.org/rfc/rfc3682.txt>

[RFC3882] D. Turk, Configuring BGP to Block Denial-of-Service Attacks, IETF, 2004, <http://www.ietf.org/rfc/rfc3882.txt>

[ROA_1] M. Lepinski, S. Kent, An Infrastructure to Support Secure Internet Routing, IETF, 2012, <http://tools.ietf.org/html/rfc6480>.

[ROA_2] ripe, managing ROA, <http://www.ripe.net/lir-services/resource-management/certification/resource-certification-roa-management>

[RPKI] rpki.net project site, <https://rpki.net/>

[SBGP] Charles Lynn, Joanne Mikkelsen, Karen Seo, IETF, 2003, <http://tools.ietf.org/html/draft-lynn-s-bgp-protocol-01>

[SO-BGP] R. White, IETF, 2006, <http://tools.ietf.org/html/draft-white-sobgp-architecture-02>

[WIKI RR] Réflecteurs de routes, http://fr.wikipedia.org/wiki/Route_reflector

DÉCOUVREZ NOTRE NOUVEAU GUIDE !

*DONNEZ UNE SECONDE VIE
À VOTRE ANCIEN ORDINATEUR !*



ACTUELLEMENT DISPONIBLE !



**LINUX
PRATIQUE
HORS-SÉRIE N°28**

CHEZ VOTRE MARCHAND DE JOURNAUX
ET SUR NOTRE BOUTIQUE EN LIGNE :
boutique.ed-diamond.com

MISE EN PLACE DU SIEM PRELUDE EN ENTREPRISE – RETOUR D'EXPÉRIENCE

Guillaume Lopes - Consultant Sécurité, Intrinsec – Guillaume.Lopes@Intrinsec.com - @Guillaume_lopes

Mathieu Mauger - Consultant Sécurité, Intrinsec – Mathieu.Mauger@Intrinsec.com

mots-clés : SIEM / PRELUDE / RETOUR D'EXPÉRIENCE

Prelude se présente comme une solution universelle de « Security Information and Event Management » (SIEM). L'objectif de cet article est de vous présenter la solution Prelude, ainsi que sa mise en place en entreprise. Les avantages et les inconvénients de Prelude seront également discutés.

1 Introduction

1.1 Qu'est-ce qu'un SIEM ?

Avant l'apparition du terme SIEM (Security Information and Event Management) en 2005, il faut bien comprendre qu'il y avait deux méthodes distinctes de gérer les événements de sécurité d'un système d'information.

D'une part, nous avons, notamment, l'analyse en temps réel de fichiers de journalisation (logs) d'équipements réseau. Par exemple, l'analyse des logs d'un pare-feu en temps réel afin de détecter une attaque et mettre en place une règle de filtrage adéquate. D'autre part, l'externalisation des logs sur un serveur central afin de conserver les événements de sécurité et les analyser en cas de problème.

Un SIEM permet de faire converger ces deux fonctionnalités et de fournir une solution complète permettant de collecter, de normaliser, d'agréger, d'archiver, d'analyser, d'alerter, de corréler et de fournir des tableaux de bords des événements de sécurité du système d'information.

1.2 Présentation de Prelude

Prelude a été créé en 1998 par Yoann Vandoorselaere et était à l'origine une solution de détection d'intrusion (IDS). Depuis, le projet a évolué jusqu'à rejoindre le mode de fonctionnement d'un SIEM.

En janvier 2012, la solution Prelude a été rachetée par la société C-S possédant déjà plusieurs solutions de surveillance de zones. Il existe à ce jour trois versions de Prelude :

- Une version communautaire nommée OSS publiée sous licence GPLv2 : Elle fournit les fonctionnalités de base de la solution, mais est limitée en terme de performances ;
- Une version professionnelle : Cette version intègre de nouvelles fonctionnalités (gestion de tickets, génération de rapports, gestion d'une authentification LDAP, etc.) et fournit des performances supérieures

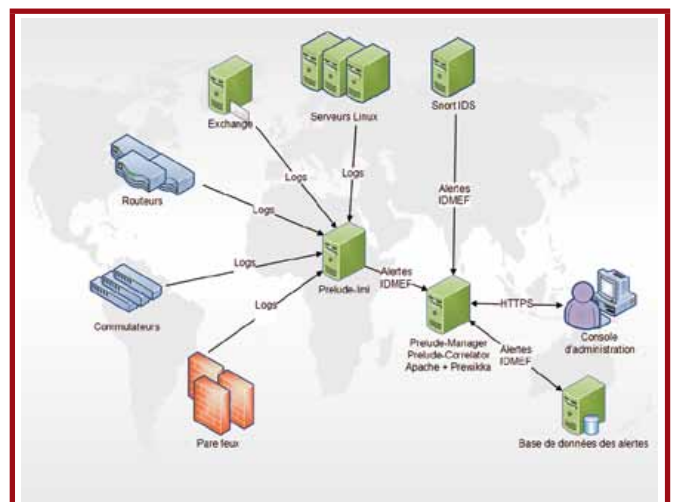


Fig. 1



à la version communautaire en utilisant des bibliothèques spécifiques ;

- Une version entreprise : Cette version est assez similaire à la version professionnelle et se différencie, notamment, par l'intégration de quelques fonctionnalités avancées (cartographie et inventaire du réseau entre autres). L'éditeur considère que cette solution permet de mettre en place un SOC (Security Operation Center) au sein d'une organisation.

Un comparatif détaillé des trois versions est présent sur le site de l'éditeur **[SITE]**.

L'architecture de la solution Prelude repose sur un modèle distribué composé d'un manager et de différentes sondes. Les sondes sont chargées d'envoyer les informations relatives aux événements de sécurité au manager, qui se charge de l'analyse (Fig. 1).

Il est à noter que toutes les communications effectuées entre les différents éléments de l'architecture sont chiffrées à l'aide d'une clé RSA de 2048 bits par défaut. Lors de l'enregistrement d'une sonde auprès d'un manager, cette dernière récupère un certificat unique de type x509 afin de protéger les futures communications. L'échange de clé est détaillé dans cet article **[CLES]**.

Prelude reçoit, analyse, corrèle et normalise les informations des différents équipements du système d'information sous un format universel. De plus, la solution est en mesure d'ajouter des informations spécifiques, sous la forme de méta-données, en fonction de la criticité de l'équipement émetteur. On peut ainsi modifier le type (admin, dos, fichier, utilisateur, etc.) ou la pertinence d'une alerte. Par exemple, les alertes émises par le pare-feu en frontal peuvent être considérées moins pertinentes.

Pour communiquer, la solution utilise le format standardisé IDMEF **[RFC]** afin de décrire une alerte de façon exhaustive en suivant un modèle dit objet. Ce format permet aux solutions commerciales et celles sous différents types de licences de communiquer dans un langage commun au sujet des événements de sécurité.

Le standard IDMEF offre un vocabulaire précis dans le domaine de la détection d'intrusion et permet, notamment, de savoir si une alerte a déjà été traitée, son état, l'émetteur de l'alerte, etc. L'implémentation des messages est simplifiée afin de limiter l'ajout d'informations inutiles lors des différents transferts entre les services de la solution.

Voici un extrait d'une alerte sous le format IDMEF :

```

alert.target(0).node.id=d1c2b3a4-001
alert.target(0).node.category=dns
alert.target(0).node.address(0).category=ipv4-addr-hex
alert.target(0).node.address(0).address=0xde796f70
alert.classification.text=Teardrop detected
alert.classification.reference(0).origin=bugtraqid
alert.classification.reference(0).name=124
alert.classification.reference(0).url=http://www.securityfocus.com/bid/124
    
```

Dans notre contexte, la version communautaire était suffisante pour répondre à nos besoins décrits dans la section suivante. Nous parlerons uniquement de cette version dans la suite de cet article.

1.3 Infrastructure cible

L'objectif de notre projet se concentrait sur la possibilité de détecter des attaques informatiques sur notre système d'information. La solution utilisée ne devait pas avoir d'impacts importants sur les performances réseau et systèmes (perte de performance engendrée par l'installation d'un agent, ouverture de flux supplémentaires, etc.). En outre, la solution devait être en mesure de détecter les événements de sécurité au travers de journaux d'événements et des messages spécifiques de certaines solutions (notamment Snort).

De plus, la mise en place d'une solution de type SIEM devait nous donner la possibilité de sélectionner de manière précise le périmètre à surveiller et permettre l'analyse des journaux d'événements sur un équipement de centralisation. L'idée était de sélectionner uniquement les équipements qui nous semblaient être les plus critiques de l'entreprise (manipulation d'informations sensibles, contraintes de disponibilités, etc.). Aucun poste utilisateur n'a été sélectionné car peu d'informations sensibles sont stockées en local et il s'agit principalement de postes nomades (l'inconsistance des logs dans le temps pouvant représenté une difficulté supplémentaire). Il est tout de même à noter que les postes nomades sont particulièrement sensibles et que leur supervision ne doit pas être négligé, bien que nous ayons fait le choix de ne pas les intégrer à notre périmètre.

Nous avons donc décidé de surveiller les équipements suivants de notre infrastructure :

- 3 routeurs Linux kernel 2.6 ;
- 2 pare-feu Linux avec IPTables ;
- 100 serveurs Linux ;
- 2 serveurs Windows 2008 : un serveur mail Exchange et un serveur de fichier ;
- Une sonde Snort IDS chargée de collecter les intrusions sur le réseau ;
- Un commutateur CISCO.

Voici un schéma simplifié de l'infrastructure cible : (Fig. 2 page suivante).

Enfin, la solution devait aussi proposer une interface de management regroupant toutes les alertes, être libre de droits et être déployable sur des équipements utilisant un noyau Linux. Ces contraintes ont écarté de facto des solutions telles que Splunk ou OSSIM dans leurs versions gratuites. On notera également

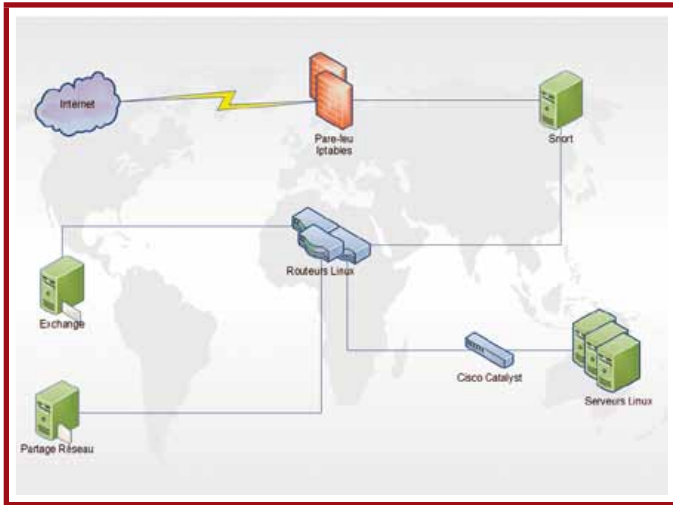


Fig. 2

que la version libre de Prelude n'intègre pas toutes les fonctionnalités disponibles en version Enterprise (par exemple : fonctionnalités de gestion de tickets et graphiques d'évolution).

Afin de répondre à cette problématique, nous nous sommes tournés vers la solution Prelude et sa console de management Prewikka.

Pour notre plate-forme SIEM Prelude, nous avons mis en place deux serveurs :

- Un serveur de centralisation des logs (8GB de RAM, Processeur intel XEON E7 et 500GB d'espace disque) : Ce serveur permet de centraliser et d'archiver les logs des différents équipements de l'infrastructure. Il disposait du composant prelude-lml ;
- Un serveur maître de management (16GB de RAM, Processeur intel XEON E7 et 250GB d'espace disque) avec les composants prelude-manager, prelude-correlator, MySQL et Apache pour l'accès à l'interface de management Prewikka.

Il est à noter que dans le cadre de la solution Prelude, il est très important d'avoir des machines performantes (RAM, CPU et espace disque) afin de traiter rapidement les données reçues par les différentes sondes.

Au niveau des alertes nous avons constaté un volume d'environ 80 GB par trimestre en période de tests et de configuration. La volumétrie a ensuite diminuée aux alentours de 35 GB tous les trois mois.

La période de rétention des données fut décidé de manière arbitraire à trois mois. Cette période nous a permis ainsi de garder des traces des actions passées mais aussi de limiter l'utilisation de l'espace disque des équipements.

2 Étude de la solution

2.1 Composants

Comme expliqué précédemment, la solution utilise un modèle distribué et est composée des éléments suivants :

- Prelude-Manager ;
- Prelude-LML ;
- Prelude-Correlator ;
- Une interface de gestion ;
- La bibliothèque transverse libprelude ;
- Des sondes tierces (par exemple la solution Snort).

2.1.1 Bibliothèque Prelude

Libprelude est le composant principal de la solution. Cette bibliothèque contient tous les modules de Prelude et permet aux développeurs d'intégrer les fonctionnalités de la solution aux autres produits présents dans le Système d'Information de l'entreprise. Par exemple, il est possible d'intégrer cette bibliothèque, moyennant un développement spécifique, à divers produits de détection d'intrusion, (par exemple la solution Bro) afin que les messages soient directement émis sous le format de détection utilisé par la Libprelude. La bibliothèque fournit aussi une interface unique et standard de communication entre les différents éléments du système de détection. Afin de respecter les standards et de permettre une interopérabilité de la solution, le format des messages utilisé par Libprelude est IDMEF.

Outre une standardisation des procédures, la bibliothèque incorpore aussi tous les outils nécessaires à l'authentification mutuelle et au chiffrement des communications entre les différents composants de la solution. Il est nécessaire d'installer cette bibliothèque sur chaque sonde de l'infrastructure Prelude.

La configuration de la bibliothèque est simple et rapide. Il est, cependant, nécessaire de configurer la fréquence d'envoi des heartbeats dans le fichier `/etc/prelude/default/global.conf` en accord avec l'intervalle TCP dans le fichier `/etc/prelude/default/client.conf` afin de ne pas voir ses sondes apparaître régulièrement comme « mortes » dans le manager.

2.1.2 Prelude-Manager

Prelude-manager a pour but de recevoir les alertes générées par les sondes dont il a la charge et est en mesure de les stocker sous différents formats :



- Fichiers plats : stockage des données en mode texte ;
- Fichiers XML-IDMEF : format de stockage en accord avec le standard RFC permettant l'export et l'import des données vers n'importe quel service Prelude ;
- Base de données SQL : utilisation d'une base de données pour le traitement des informations par des logiciels tiers.

Il est aussi possible de transférer les alertes selon deux mode :

Avant chaque insertion d'une alerte, il est possible de configurer un ensemble de filtres pour rejeter certaines alertes. Cela permet d'alléger la base de données et d'effectuer différents types d'actions à la réception d'un événement de sécurité. La configuration de ces filtres est présentée dans la section 2.3.

Il est à noter qu'il est possible de configurer prelude-manager pour qu'il soit esclave d'un autre manager sur un site distant. Dans ce cas, le manager esclave concentre les informations de la zone dont il a la charge avant d'envoyer les alertes pertinentes au manager maître.

Afin de stocker les événements de sécurité et les différentes alertes, nous avons décidé d'utiliser le système de gestion de bases de données MySQL afin que la console Prewikka puisse avoir accès aux alertes. Nous n'avons pas décidé d'utiliser les autres formes de stockages afin de ne pas surcharger l'espace disque du manager Prelude.

2.1.3 Prelude-lml

Prelude-lml est la sonde chargée de collecter et formater les logs pour les envoyer au manager pour interprétation.

La collecte des logs peut être effectuée selon deux modes :

- Analyse des fichiers de logs envoyés par les équipements sur le serveur hébergeant la sonde ;

Note

Par défaut, Prelude-lml est compatible nativement avec un certain nombre de logiciels de sécurité qui sont à même d'utiliser le Framework Prelude afin d'optimiser la collecte de données. Dans ce cas, les sondes sont en mesure d'envoyer des alertes directement au manager. Prelude supporte également le format de journalisation d'un grand nombre d'équipements et de solutions informatiques du marché. Une liste complète de ces compatibilités est disponible sur le Wiki de la communauté [COMPATIBILITES].

- Réception des logs des équipements directement par le protocole UDP.

Dans notre situation, nous avons mis en place un serveur de centralisation de log sur lequel était installé la sonde.

Afin d'effectuer cette tâche, la sonde va analyser l'ensemble des fichiers de logs mis à sa disposition à l'aide des expressions régulières Perl PCRE. Les événements de sécurité contenus dans les logs sont ensuite transformés dans le format IDMEF et envoyés au manager.

Un ensemble de règles prédéfinies est disponible dans les fichiers `/etc/prelude-lml/ruleset/pcr.rules` et `/etc/prelude-lml/ruleset/single.rules`.

L'utilisation du format PCRE pour la reconnaissance des fichiers de log permet de modifier et d'ajouter rapidement des règles supplémentaires.

Le format par défaut défini par la sonde est celui du gestionnaire syslog. Il est néanmoins possible de créer ses propres règles dans le cas où les fichiers d'événements produits par les équipements du réseau ne seraient pas compatibles nativement avec la solution Prelude.

Le temps nécessaire à la mise en place de nouvelles règles de lecture pour des logs spécifiques est très rapide via l'utilisation d'expressions régulières.

Voici un exemple de configuration pour l'utilisation du format rsyslog :

```
[format=rsyslog]
time-format = "%b %d %H:%M:%S"
prefix-regex = "^(?P<timestamp>.{15}) (?P<hostname>\S+)
(?: (?P<process>\S+)?(?: \[(?P<pid>[0-9]+)\])?)?"
file = /var/log/rsyslog/*.log
```

Dans le cas de certains types de machine, Windows NT par exemple, un logiciel supplémentaire est requis pour convertir les journaux d'événements au format syslog.

2.1.4 Prelude-Correlator

Prelude-correlator est le composant de l'architecture qui va regrouper les différentes alertes afin de générer des groupements d'alertes et des schémas d'attaques. De plus, Prelude-correlator est en mesure de détecter les faux positifs selon des règles prédéfinies.

Pour effectuer cette tâche, le service Prelude-correlator va régulièrement consulter les alertes stockées dans la base de données, en plus de celles reçues en temps réel, et rechercher des liens entre ces dernières afin de faire des groupements d'alertes.

En effet, chaque alerte remontée peut être la partie émergée d'un problème bien plus grand et bien plus grave en conséquences.



La corrélation d'une alerte par cette sonde va suivre le cheminement logique suivant :

1. Récupération des données de l'alerte (IP source et destination, processus, interfaces réseaux, ports, message, etc.) ;
2. Recherche de correspondance avec les alertes, corrélées ou non, dans la base de données du manager ;
3. Analyse heuristique du comportement par rapport aux schémas d'attaques les plus courants ;
4. Vérification des déplacements réseau entre les alertes pour détecter un éventuel pivot (par exemple compromission d'un équipement par brute force puis rebond sur les équipements du sous réseau) ;
5. Remontée de l'alerte de corrélation au manager en adaptant le niveau de sévérité en fonction des alertes corrélées. Si la nouvelle alerte rentre dans les paramètres des règles de filtrages, l'action correspondant au filtre utilisé est déclenchée (par exemple l'envoi d'un mail d'alerte) (Fig. 3).

Afin de faciliter ce raisonnement, toutes les alertes remontées sont triées et regroupées en fonction de critères prédéfinis (criticité, processus, etc.).

La corrélation étant basée sur un système d'analyse des attaques informatiques, certaines actions déclencheront une mise en marche automatique du diagramme de recherche :

- Collecte d'informations sur le réseau (prise d'empreinte, scan de ports, etc.) ;
- Recherche de vulnérabilités ;
- Exploitation des vulnérabilités identifiées en vue d'obtenir un accès à un équipement ;
- Tentative d'élévation des privilèges sur l'équipement compromis ;
- Maintien de l'accès (installation de portes dérobées, effacement des traces, etc.).

Chacune de ces actions peut être un signe d'une attaque imminente ou en cours de réalisation. C'est pourquoi dès qu'une alerte répondant à un ou plusieurs de ces critères est remontée, le scénario de corrélation s'activera.

Le service Prelude-correlator va aussi tenter, à intervalles réguliers, d'effectuer de nouveaux liens entre plusieurs alertes corrélées précédemment et des alertes isolées afin de compléter les corrélations passées.

Ce système permet ainsi de détecter des attaques s'étalant sur une période de temps importante.

Par défaut, Prelude-correlator intègre un certain nombre de plugins disponibles dans le fichier `/etc/prelude-correlator/prelude-correlator.conf`

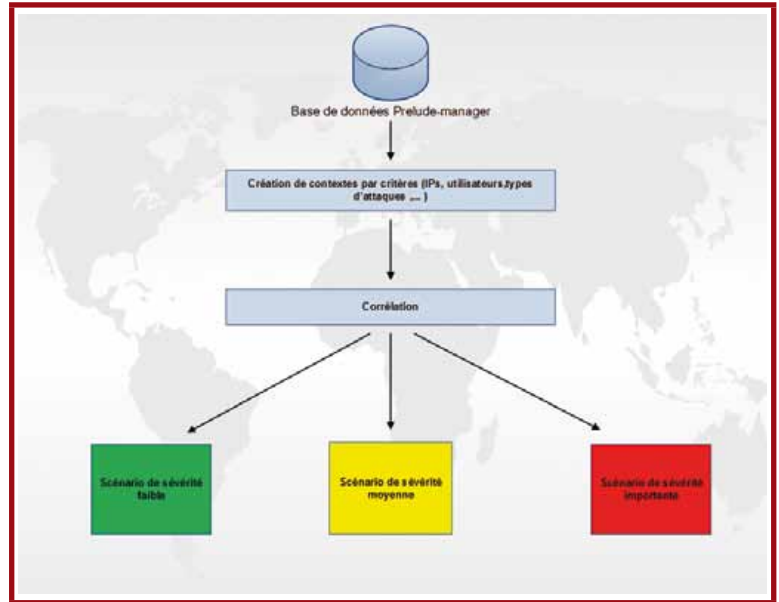


Fig. 3

permettant l'analyse des alertes dans leurs domaines de compétences respectifs.

En voici une liste non exhaustive :

- EventStorm : permet de regrouper un grand nombre d'alertes possédant un ou plusieurs points communs mais sans corrélation particulière. Par exemple, une attaque de Déni de service créera un grand nombre d'alertes qui seront regroupées sous la dénomination EventStorm ;
- OpenSSHAUTH : permet de gérer les corrélations d'alertes impliquant les services OpenSSH ;
- Worm : intègre les divers comportements des vers informatiques permettant de déterminer la présence éventuelle d'une telle menace en corrélant diverses alertes ;
- Firewall : permet la corrélation d'alertes remontées par les pare-feux surveillés.

Le mécanisme de corrélation repose sur l'utilisation du langage Python permettant une grande flexibilité dans l'écriture de nouvelles règles. Afin de créer un nouveau plugin, il est nécessaire de déclarer une nouvelle classe Python où le scénario sera défini à l'aide du langage IDMEF.

Pour faciliter l'écriture des scénarios, les classes suivantes fournissent un ensemble de fonction destinées à manipuler les informations :

- `PreludeCorrelator.context` ;
- `PreludeCorrelator.pluginmanager`.

Les nouveaux plugins doivent ensuite être déposés dans le répertoire `/usr/lib/python2.6/PreludeCorrelator/plugins`.



VENEZ DÉCOUVRIR NOS GUIDES !

DÉJÀ PARUS !



À PARAÎTRE !



DISPONIBLES CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR : boutique.ed-diamond.com

ent est la propriété exclusive de Johann Locatelli (johann.locatelli@businessdecision.com) - 06 janvier 2016

2.1.5 Prewikka

Prewikka est la solution officielle du projet Prelude fournissant une console de visualisation et de gestion des alertes stockées en base de données remontées par Prelude-manager.

On notera l'existence de précédentes solutions, non maintenues, permettant la visualisation d'alertes Prelude :

- Prelude-PHP-Frontend : solution développée en PHP et destinée à être installée sur un serveur séparé des composants de la solution.
- Prelude-Perl-Frontend : interface du projet « Le Routier » développé en Perl offrant les mêmes possibilités que son homonyme PHP.

Nous nous concentrerons sur les atouts et le fonctionnement de Prewikka puisque cette solution est devenue l'interface officielle du projet Prelude.

Prewikka intègre les caractéristiques suivantes nécessaires à l'architecture SIEM de Prelude :

- Gestion des heartbeats (Equivalent des TTL réseaux) : permet de définir quand chaque sonde a communiqué pour la dernière fois avec le contrôleur.
- Exportation et visualisation des tableaux de bord graphiques ;
- Règles de filtrages avancées permettant des recherches logiques ou dichotomiques.

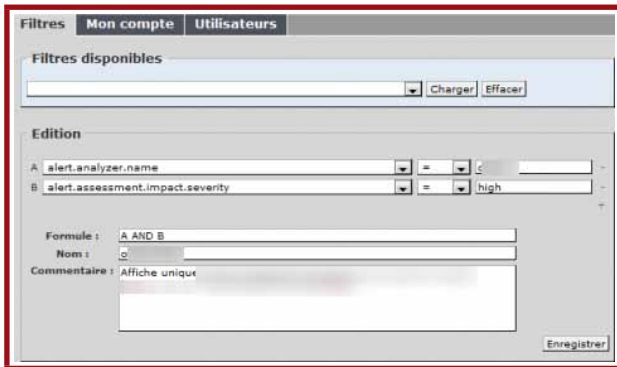


Fig. 4

Il est à noter que la version professionnelle de Prelude permet la gestion de tickets et la génération de rapports et de statistiques au travers de l'interface Prewikka.

2.2 Filtres d'alertes

Le système de filtres utilisé par le manager repose sur la définition de critères d'identification en accord avec le standard RFC IDMEF. Les filtres sont ensuite traités séquentiellement.

En d'autres termes, les règles sont exécutées dans l'ordre de définition (comme pour les règles d'un pare-feu).

La création d'un filtre est effectuée selon trois paramètres :

- Nom du filtre entre les symboles '[' et ']' ;
- Le type d'événement à traiter à l'aide du format IDMEF au sein de la variable **rule** ;
- L'action à effectuer quand une alerte correspond au filtre avec la variable **hook**.

Il est possible d'affiner l'action d'un filtre avec l'ajout de l'instruction thresholding. Cette instruction est définie par un nom de la même manière que le filtre ci-dessus mais possède en plus les attributs **count**, **path** et **limit**.

Ainsi en empilant ces deux systèmes de filtres, on est en mesure d'effectuer plusieurs actions précises sur un même type d'alerte.

Prenons par exemple le cas où un éventuel attaquant exécute une attaque par brute force sur un service SSH. En l'absence du filtre thresholding, chaque connexion fera l'objet d'une alerte « Tentative de connexion SSH avec l'utilisateur root » qui sera suivie d'un envoi de mail à l'administration. Sous le coup d'une attaque de force brute, un nombre important de mails peut être envoyé à l'administrateur, noyant ainsi les autres informations pertinentes du réseau.

Il est donc possible de limiter l'envoi de mail par l'empilement de filtres suivant :

```
[idmef-criteria=BruteForce]
rule = alert.assessment.impact.severity == 'Brute Force attack'
hook = thresholding[BruteForce]

[thresholding=BruteForce]
path = alert.classification.text, alert.target(0).node.address(0).address
count = 1
limit = 3600
hook = smtp[test]
```

Ce filtre peut être traduit ainsi : « Les alertes de BruteForce entrantes (option 0) ne sont remontées qu'une seule fois (**count=1**) toutes les heures (**limit=3600**) si la connexion provient de la même adresse IP (**path**) ».

2.3 Remontée des alertes

La solution Prelude intègre un système de remontée des alertes. Ce mécanisme repose sur l'utilisation de plugins intégrés à prelude-manager permettant d'enregistrer les alertes sous trois formats.



2.3.1 Mail

Pour mettre en place ce dispositif, il suffit d'éditer le fichier `/etc/prelude-manager/prelude-manager.conf` et recherchez le bloc `[smtp]`. Voici un exemple de configuration :

```
[smtp=test]
sender = prelude@test.com
recipients = rssi@test.com
smtp-server = 192.168.10.240
template = /etc/prelude-manager/smtp-template/template1
```

Voici un exemple d'un template mail défini dans le fichier `/etc/prelude-manager/smtp-template/template1` :

```
Sig Alert Name: $alert.classification.text:
Sig Alert ID: $alert.classification.ident
Severity: $alert.assessment.impact.severity
Source IP: $alert.source(0).node.address(0).address
Source Port: $alert.source(0).service.port
Target IP: $alert.target(0).node.address(0).address
Target Port: $alert.target(0).service.port
Create Time: $alert.create_time
http://192.168.10.45/prewikka?
```

2.3.2 Fichiers texte

Le plugin Textmod permet de stocker les alertes remontées dans un fichier texte.

Pour activer ce plugin, il faut retirer les instructions de commentaires sur les lignes suivantes dans le fichier `/etc/prelude-manager/prelude-manager.conf` en renseignant le fichier de destination :

```
[TextMod]
logfile = /var/log/prelude.log (Ce chemin peut être modifié)
```

2.3.3 XML

Le plugin XML permet de stocker les différentes alertes dans un fichier XML. Ce formatage permet d'exporter facilement les données vers une autre infrastructure et d'assurer ainsi l'interopérabilité de la solution.

Pour activer ce plugin, il suffit de décommenter les lignes suivantes du fichier `/etc/prelude-manager/prelude-manager.conf` :

```
[XmlMod]
logfile = /var/log/prelude-xml.log
```

3 Points forts

Le grand avantage de cette infrastructure est de permettre l'analyse à distance d'informations sur un serveur de centralisation. Cette concentration

d'information est alors accessible à l'ensemble des services Prelude sans avoir à installer d'agent (exception faite de certaines solution nécessitant une conversion des journaux d'événements au format syslog) sur les machines supervisées.

La personne en charge de surveiller l'activité du Système d'Information se voit ainsi attribuer une console de gestion où sont concentrées les informations de sécurité de tous les équipements.

Il est à noter que la possibilité d'empiler les filtres permet d'affiner les recherches et les actions à effectuer sur des scénarios ou des alertes complexes.

De plus, la solution Prelude possède une compatibilité native avec de nombreux équipements du marché et divers systèmes d'exploitation. Il est donc possible de configurer rapidement, comme nous l'avons vu précédemment, les différentes sondes et de déployer l'infrastructure en production. Ceci permet de mutualiser des ressources humaines expertes, rares et coûteuses, nécessaires à la mise en place des solutions de sécurité.

Grâce à cette architecture, nous avons été en mesure d'identifier rapidement les comportements suspects contraires à la politique de sécurité de l'entreprise. Par exemple, nous avons été en mesure d'identifier rapidement le contournement de proxy ou encore le raccordement de deux sous-réseau isolés par défaut.

Cet outil permet de dresser des statistiques sur le nombre d'attaques subies par l'entreprise qui permettront, par exemple, de justifier des demandes d'augmentation de budgets pour améliorer la sécurité du Système d'Information, ou de suivre leurs évolutions dans le temps.

L'utilisation de cette solution donne la possibilité, grâce à l'étude des alertes remontées, de mettre en avant les points du réseau où un durcissement de la sécurité est nécessaire.

4 Points faibles

Les alertes générées par Prelude dépendent totalement des logs fournis par les équipements du réseau.

En effet, seuls les fichiers de logs ou les équipements intégrant la bibliothèque Prelude permettent la remontée d'alertes auprès du manager.

Ainsi, dans le cas où une machine n'envoie plus ces fichiers d'activités, par exemple suite à une compromission, un lien défectueux ou encore une personne interne réalisant une écoute active en filtrant les flux, il n'est plus possible d'obtenir les événements de sécurité.

Il en va de même lorsqu'un éventuel attaquant utilise la fragmentation IP afin de passer inaperçu auprès des



sondes réseaux telles que Snort. Si la sonde ne relève pas de trafic suspect alors la solution Prelude non plus.

Un SIEM n'est donc pas une solution miracle permettant de voir toutes les attaques et tous les comportements suspects sur un réseau. Cette infrastructure fonctionne sous une forme pyramidale et repose avant tout sur le bon fonctionnement des sondes qui la composent ainsi que sur les fichiers d'événements propres à chaque équipement.

De plus, au sein de cette solution, les faux positifs peuvent être nombreux. Il est donc nécessaire de posséder un œil critique sur les différentes alertes remontées et de bien configurer les règles afin de ne pas surcharger la base de données ainsi que la personne en charge de l'analyse des événements de sécurité.

Ce genre de situation nécessite un temps de travail quotidien important afin de surveiller les actions journalières, généralement nombreuses, et d'affiner les filtres pour écarter les faux positifs. Il est aussi à noter que chaque alerte nécessite une vérification manuelle par l'administrateur qui est elle aussi consommatrice de ressources humaines et de temps.

En ce qui concerne la mise en place d'une solution SIEM, il faut être conscient que cela prend du temps sur plusieurs niveaux. D'une part, il est nécessaire de définir rigoureusement le périmètre à surveiller ainsi que les ressources qui seront attribuées pour supporter l'infrastructure. Par la suite, il est nécessaire de configurer tous les équipements du périmètre afin de centraliser les logs, dans notre cas, sur une machine.

Dans le cadre d'un réseau entre plusieurs datacenters, le temps de mise en place est multiplié par la nécessité d'installer une infrastructure par sous-réseau.

Conclusion

La solution communautaire Prelude permet, à moindre coût, de mettre en place une solution SIEM au sein de son organisation afin d'observer, de quantifier et d'analyser les différents événements de sécurité d'un système d'information.

Après configuration des filtres, il est possible d'identifier les parties du réseau, les systèmes d'exploitation ainsi que les services nécessitant un durcissement de la configuration et de la sécurité.

A moyen terme, les rapports du SIEM permettent de concentrer les efforts sur les points névralgiques du réseau et aussi d'augmenter de manière significative la sécurité du Système d'Information.

Il apparaît donc clairement que l'utilisation d'un SIEM ne permet pas de se prémunir de tous les types

d'attaques et que l'application des bonnes pratiques en matière de sécurité reste indispensable. En effet, une architecture SIEM typique ne propose pas de sécurité active et les actions de protections doivent par la suite être apportées manuellement.

Il est important de comprendre que ce genre de solution ne peut pas non plus être implémenté dans de très petites structures et être rentables. En effet, le temps passé à s'occuper des différentes alertes est non négligeable et ne peut être utilisé pour effectuer d'autres actions de maintenance. Il a été nécessaire dans notre cadre d'affecter une personne à temps partiel sur l'analyse des alertes.

Pour information, la solution fût mise en production après trois mois d'étude, de configuration et d'analyse. Une fois en production, un mois et demi supplémentaire a été nécessaire afin d'affiner les résultats et la remonté pertinente d'alerte.

Il est à noter que cette solution, dans sa version communautaire, n'est pas non plus adapté à de grosses structures. En effet, les performances de l'outil diminue proportionnellement au nombre d'événements de sécurité traités et la version communautaire ne profite pas des améliorations apportées par les solutions payantes (Archivage, gestion et indexation des journaux d'événements bruts).

Ainsi, un SIEM peut être utilisé afin de mettre en avant les problèmes de sécurité d'un Système d'Information, d'appuyer les demandes de moyens en vue d'élever la sécurité mais aussi pour observer l'évolution de cette dernière au fil du temps.

Pour finir, la solution Prelude est bien pensée, facile à prendre en main (indépendamment de l'implémentation générale d'un SIEM) et dispose d'une architecture modulaire permettant de l'adapter à la plupart des infrastructures. On regrettera seulement que certaines fonctionnalités soient devenues payantes après le rachat par C-S. ■

■ REMERCIEMENTS

Nous tenons à remercier Cédric pour sa relecture attentive et minutieuse.

■ RÉFÉRENCES

[SITE] <http://www.prelude-ids.com/en/products/compare-products/index.html>

[RFC] <http://www.ietf.org/rfc/rfc4765.txt>

[COMPATIBILITES] <https://www.prelude-ids.org/wiki/prelude/PreludeCompatibility>

[CLES] <https://www.prelude-ids.org/wiki/prelude/InstallingAgentRegistration>

Complétez votre collection d'anciens numéros !

Ce document est la propriété exclusive de Jehanne Locatelli - Locatelli.fr - jehanne.locatelli@business-science.com - 01 47 34 11 11 - 2016 à 09:44



VERSION PAPIER

Rendez-vous sur :
boutique.ed-diamond.com
et (re)découvrez nos magazines
et nos offres spéciales !



boutique.ed-diamond.com

MISC
Multi-System & Internet Security Content

106 - SÉCURITÉ INFORMATIQUE

SEPTEMBRE/OCTOBRE 2016

Faux certificats forgés : peut-on encore faire confiance à SSL ?

MIGRATION
Automatisez vos tests de migration grâce à Selenium !

SECURITE
Bloquez les attaques contre votre réseau avec l'IPS Suricata

MIGRATION
Migration vers IPv6 : quel impact sur votre vie privée ?

COMMERCE ÉLECTRONIQUE
SUPERVISEZ LA SÉCURITÉ de votre système d'information !

- 1 - Et vous, vous avez quoi dans vos logs ?
- 2 - Corrélation des événements systèmes et réseau
- 3 - Surfez sur les réseaux avec NetFlow
- 4 - Les TLD aux rayons X

MISC
Multi-System & Internet Security Content

107 - SÉCURITÉ INFORMATIQUE

NOVEMBRE/DECEMBRE 2016

LA SÉCURITÉ & TÉLÉPHONIE : LA SÉCURITÉ ULTRA-CONNECTÉE

- 1 - Confidentialité de l'information
- 2 - Diffusion d'informations via les réseaux sociaux
- 3 - Architecture des réseaux (IT)
- 4 - Exploitation d'outils de sécurité des réseaux sociaux



VERSION PDF

Rendez-vous sur :
numerique.ed-diamond.com
et (re)découvrez nos magazines
et nos offres spéciales !



numerique.ed-diamond.com

DÉTECTION DES SHELLCODES POLYMORPHES

Mehdi Talbi – mtalbi@arkoon.net - Ingénieur sécurité chez Arkoon Network Security

mots-clés : SHELLCODE / POLYMORPHISME / BUFFER OVERFLOW / DÉTECTION D'INTRUSIONS

Le polymorphisme est une technique d'obfuscation de shellcodes rendant obsolètes les techniques de détection basées sur de la simple recherche de motif (pattern matching). Cet article décrit un certain nombre de solutions de contournement issues du monde de la recherche.

1 Les shellcodes polymorphes

1.1 Evolutions des techniques d'obfuscation de shellcodes

Une attaque par débordement de tampon (*buffer overflow*) au niveau de la pile fait généralement intervenir la charge utile (payload) présentée au niveau de la figure 1. À la vue de ce schéma, on comprend qu'il est trivial pour un NIDS (*Network Intrusion Detection System*) de détecter un tel code malveillant. En effet, une longue séquence d'instructions **NOP** (*No Operation*, opcode **0x90**) est souvent révélatrice d'une attaque de type *buffer overflow* ou d'une exploitation d'un bug de chaîne de format (format string bug). Une partie du shellcode (ex. : la chaîne `/bin/sh`) ou bien la répétition successive de l'adresse de retour (RET) sont également candidats à faire partie d'une signature d'attaque. Afin d'échapper à la détection, plusieurs techniques d'évasion ont vu le jour. Elles peuvent s'appliquer à chacune des parties composant la charge malveillante : la section des NOPs, l'adresse de retour et le shellcode. Un exemple typique de camouflage du shellcode serait de remplacer certaines instructions par d'autres sémantiquement équivalentes. Par exemple, l'instruction `sub $0xffffffe, %eax` peut être substituée par deux instructions `inc %eax`. Cette technique d'obfuscation de code est souvent appelée « métamorphisme ».



Fig. 1 : Payload classique

Certaines protections s'appuient sur le fait que la présence d'instructions dans un flux à destination d'un service ne traitant que des caractères alphanumériques est souvent symptomatique d'un code malveillant. Cependant, des générateurs de shellcodes alphanumériques ont vu le jour [7]. Ces derniers, utilisant uniquement des instructions ayant une correspondance ASCII imprimable, permettent de contourner les protections se basant sur la nature du trafic (alphanumérique ou non). La véritable révolution dans les techniques d'obfuscation de codes provient sans doute de la scène virale : le polymorphisme. Popularisé par « Dark Avenger » (codeur de virus bulgare) au début des années quatre-vingt dix avec son outil MtE (Mutation Engine), la technique consiste à chiffrer le virus et à y inclure une routine de déchiffrement différente à chaque fois. Lorsque le virus est exécuté, la routine de déchiffrement est lancée en premier lieu afin de récupérer sa forme initiale. Ceci a eu pour conséquence de rendre obsolètes les techniques de détection par reconnaissance de motifs étant donné que le code du virus a une forme différente à chaque fois. Il s'est avéré par la suite que le polymorphisme peut être porté aux shellcodes. Un premier moteur de shellcodes polymorphes a été développé par l'équipe ADM [12].

1.2 Anatomie d'un shellcode polymorphe

Depuis l'apparition du premier générateur de shellcodes polymorphes (ADMmutate), d'autres plus évolués ont vu le jour. On peut citer à titre d'exemple, les moteurs Clet [2], JempiScore [13], et les divers encodeurs offerts par Metasploit. Ces

moteurs génèrent des charges ayant la structure illustrée par la figure 2.



Fig. 2 : Payload polymorphe

1.2.1 FAKE_NOP

Dans le cas d'un shellcode classique, il était question de le faire débiter par une séquence de NOPs permettant de compenser l'erreur d'estimation de l'adresse de retour. Cependant, il est possible de remplacer le NOP par toute autre instruction « non dangereuse » (ne faisant pas échouer l'exécution du shellcode), et tenant sur un seul octet. La liste utilisée par le moteur ADMmutate propose une cinquantaine d'instructions de remplacements possibles (**inc %eax**, **dec %eax**, **push %edx**, etc.). Étant donné que cette liste ne présente pas une palette assez large de choix, les FAKE_NOP peuvent être facilement détectés par n'importe quel IDS maintenant une liste similaire. C'est le cas par exemple de Fnord [8] qui est implémenté sous forme d'un *plugin* dans Snort.

À noter qu'il est possible également d'avoir recours à des instructions tenant sur plusieurs octets si on tient compte de la contrainte suivante : les arguments des instructions doivent être également des instructions. Ainsi, il est possible de lire des instructions valides et d'atteindre la routine de déchiffrement, et ce, quelle que soit la position vers laquelle pointerait l'adresse de retour. Par exemple, la séquence **2c 40** se traduit par **sub \$40, %al** ou par **inc %eax** selon que l'on commence la lecture à la première ou à la seconde position. L'exemple de la figure 3, extrait de [1], illustre ce principe. Les moteurs Clet et ADMmutate ne disposent pas d'une telle fonctionnalité. Il est toutefois possible de faire appel à un outil tel que ECL-Polynop [14] pour générer les FAKE_NOP et d'utiliser un des générateurs cités ci-dessus pour construire le reste de la charge.

1.2.2 Routine de déchiffrement

Le chiffrement du shellcode se base sur des techniques assez simples faisant appel à des instructions réversibles **add/sub**, **rol/ror** ou bien **xor**. Pour que le code soit totalement polymorphe, il est impératif que le code de la routine de déchiffrement soit différent à chaque fois, car dans le cas contraire, on se retrouverait certes avec un shellcode chiffré au final, mais d'un autre côté, le code de la routine étant statique, il pourrait être utilisé dans des signatures. Plusieurs techniques sont utilisées et combinées pour camoufler la routine de déchiffrement :

- chiffrement du shellcode ;
- substitution d'instructions ;

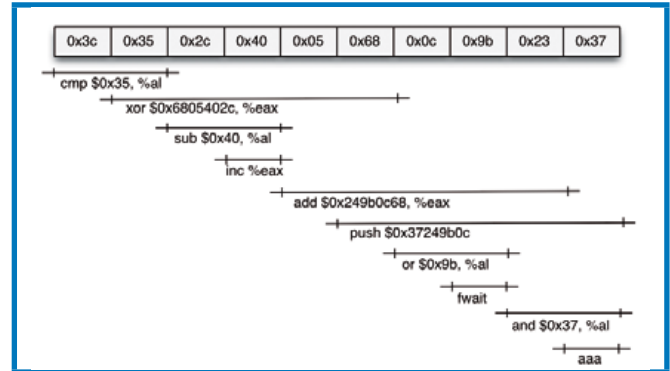


Fig. 3 : FAKE_NOP tenant sur plusieurs octets

- permutation de certains blocs du code malveillant ;
- insertion d'instructions fonctionnellement inutiles (*junk instructions*) entre différentes parties du code utile, ces instructions n'ayant pour seule utilité que de leurrer les systèmes basés sur l'analyse de flux sous forme de séquences de bits ;
- renommage (variation de l'usage) de registres, ceci ayant pour effet de produire des *opcodes* différents .

1.2.3 Padding

La zone de *padding* est utilisée généralement pour combler l'espace vide entre le shellcode et l'adresse de retour. Elle a été qualifiée par l'équipe Clet comme étant une zone en *or*, étant donné que cette composante de la charge n'est soumise à aucune contrainte. Mieux encore, cette zone a été exploitée intelligemment par la même équipe afin de défier les futures générations d'IDS se basant sur des techniques dites de Data Mining [5, 6]. L'idée est assez simple : tenter de la remplir afin que la distribution de probabilité du résultat soit conforme à un trafic normal.

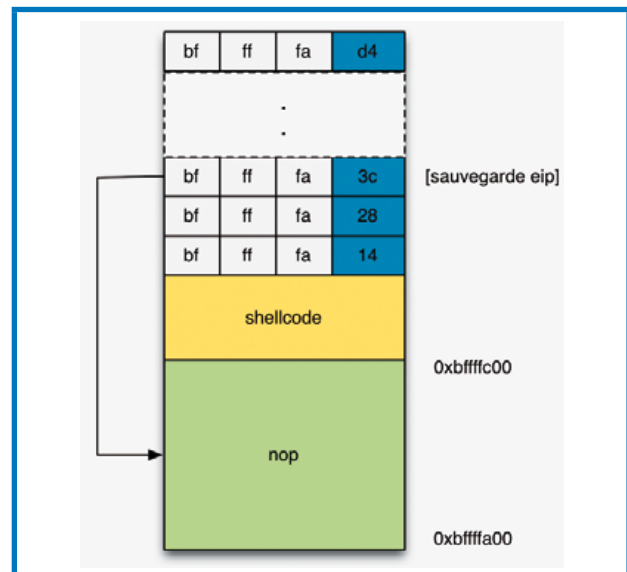


Fig. 4 : Obfuscation de l'adresse de retour

1.2.4 Adresse de retour

Actuellement, il n'existe qu'une seule méthode permettant de camoufler l'adresse de retour. Cette dernière consiste à varier les bits de poids faible (1 octet) entre les différentes occurrences. Cette légère modification influe de manière négligeable sur les chances de succès de l'attaque. En témoigne l'exemple de la figure 4 (voir page précédente) où la section des NOPs, localisée à `[0xbfffa00 - 0xbfffc00]`, permet d'atteindre le shellcode et ce peu importe l'octet de poids faible de l'adresse de retour (`0xbfffa??`).

2 Face au polymorphisme

2.1 Détection des FAKE_NOP

2.1.1 APE

La présence d'une longue suite d'instructions est assez révélatrice d'une charge malveillante. APE (*Abstract Payload Execution*)^[10] se base sur cette propriété, et plus précisément sur le calcul de la plus grande séquence d'instructions valides : MEL (*Maximum Execution Length*). Une alerte est levée si cette valeur dépasse un certain seuil (positionné par défaut à 35). Pour calculer la valeur MEL, APE utilise un algorithme récursif qui prend en paramètres deux arguments : la séquence à analyser et la position à partir de laquelle débutera l'analyse. L'algorithme désassemble les instructions itérativement en incrémentant à chaque fois la valeur MEL. Si une instruction non valide est rencontrée, le programme s'arrête et la valeur MEL est retournée. Dans le cas de sauts inconditionnels (`jmp`, `call`), le désassemblage se poursuit à l'adresse de destination qui leur est fournie comme argument. Dans le cas de sauts conditionnels (`jz`, `jne`), le désassemblage se poursuit à partir de deux nouvelles positions : 1- l'adresse de destination du saut ; 2- celle de l'instruction qui suit immédiatement l'instruction en cours. La valeur maximale MEL calculée selon ces deux chemins est retenue. Pour des raisons de performance, due au décodage d'instructions, cet algorithme n'est pas exécuté pour chaque position de la séquence mais uniquement pour quelques positions choisies aléatoirement. Les résultats d'évaluation élaborés sur un prototype d'implémentation intégré à Apache démontrent un faible impact sur les performances du serveur web.

APE se base sur la plus longue séquence d'instructions valides pour décider de la présence ou non de codes

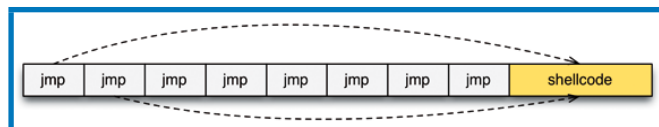


Fig. 5 : NOP Trampolines

malveillants. Or, ce paramètre ne permet pas de caractériser de façon systématique les shellcodes polymorphes. Plus précisément, APE est incapable de détecter les NOP trampolines illustrés par la figure 5.

2.1.2 STRIDE

STRIDE ^[1] détecte une séquence de FAKE_NOP (de taille n), débutant à la $i^{\text{ème}}$ position dans la chaîne à analyser, si toutes les séquences d'instructions commençant à la position $i + j$ sont valides pour tout $j \in \{0, 1, \dots, n - 1\}$. STRIDE est ainsi capable de détecter bon nombre de modèles selon lesquels sont structurés les NOP : NOP tenant sur un ou plusieurs octets, NOP trampolines, etc. Cependant, il est possible de contourner STRIDE en injectant quelques instructions non valides parmi les FAKE_NOP même si ceci aura pour conséquence de diminuer les chances d'exécution du shellcode.

2.2 Détection de la routine de déchiffrement

La représentation sous forme d'un graphe de contrôle de flux (CFG, *Control Flow Graph*) de plusieurs codes polymorphes générés par un même moteur nous permet de constater que la structure de la routine de déchiffrement est quasiment la même : les nœuds sont interconnectés entre eux de la même manière. De plus, les classes d'instructions (instructions arithmétiques, instructions logiques, instructions de transfert de données, etc.) composant un nœud sont presque toujours les mêmes. À titre d'exemple, 97 % des codes générés par Clet partagent la structure de la figure 6 (pour une même entrée : shellcode `/bin/sh`). La routine de déchiffrement débute toujours par un `jmp` (nœud 1) suivi d'un `call` (nœud 2) pour déterminer l'adresse du shellcode chiffré. Le bloc qui suit permet de récupérer cette adresse via une instruction de pile `pop`, puis initialise certains registres avant d'entrer dans la boucle de déchiffrement (nœud 4). Celle-ci, consiste en des opérations mathématiques et logiques moyennant quelques opérations de transfert de données. Un saut conditionnel vers la fin de ce bloc permet de réitérer la boucle ou bien de poursuivre l'exécution vers le début du shellcode ainsi déchiffré (nœud 6).

Les travaux ^[3, 9] s'appuient sur cette observation pour générer des signatures caractérisant l'invariance de la routine de déchiffrement. Ces signatures reflètent à la fois l'interconnexion entre les nœuds et les classes d'instructions composant chaque nœud. Cette dernière caractéristique permet de lutter contre le métamorphisme (jusqu'à une certaine limite). Le simple fait de substituer l'instruction `inc %eax` par `sub $0xffffffff, %eax` n'aura aucune incidence sur la signature puisque `inc` et `sub` appartiennent à la même classe, celle des instructions arithmétiques. Il est possible de contourner cette technique en ayant recours à des substitutions

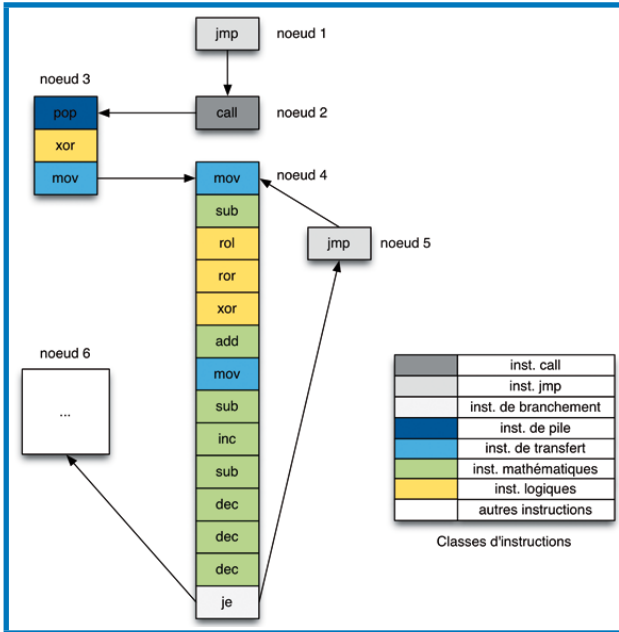


Fig. 6 : Graphe CFG de la routine de déchiffrement du moteur Clet

faisant intervenir des instructions n'appartenant pas à la même classe. Par exemple, `mov %eax, %ebx` (classe d'instructions de transfert de données) peut être substituée par `push %eax ; pop %ebx`. Cependant, le nombre de possibilités n'est pas aussi important que celui faisant intervenir des instructions de même classe.

2.3 Détection de l'adresse de retour

Buttercup [15] repose sur le fait que dans chaque code d'exploitation, l'adresse de retour est délimitée par une borne inférieure et une borne supérieure, respectivement les adresses des premier et dernier **NOP** (ou équivalents). Cet intervalle est ensuite utilisé dans une signature par l'intermédiaire d'un nouveau champ introduit à Snort : **range**. Afin de réduire le nombre de faux positifs engendrés par un large intervalle, Buttercup suggère de compléter la signature par deux nouveaux champs : **rangeoffset** (position à partir de laquelle est effectuée la recherche) et **rangedepth** (nombre de positions maximum à analyser).

2.4 Détection des motifs invariants

Bien que le code soit polymorphe, il subsiste encore quelques motifs invariants éparpillés le long du code généré. Selon des statistiques élaborées par [4], les séquences ombragées dans la figure 7 sont présents dans 100% des routines de déchiffrement générées par Clet pour une même entrée (shellcode `/bin/sh`). Les octets encadrés en jaune sont présents avec un pourcentage de plus de 20%.

NOUVEAUTÉ À DÉCOUVRIR !

GNU/Linux Magazine N° 165



INSTALLEZ VOTRE SERVEUR LDAP



ACTUELLEMENT DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX ET EN LIGNE SUR :

boutique.ed-diamond.com

EB	2D	59	31	D2	B2	20	8B	19	C1	C3	0E	81
F3	81	68	44	B3	C1	C3	0A	C1	C3	19	C1	C3
11	89	19	81	E9	FF	FF	FF	FF	41	41	41	80
E4	02	4A	4A	74	07	EB	D8	E8	CE	FF	FF	FF

Fig. 7 : Motifs invariants de la routine de déchiffrement du moteur Clet

Polygraph [4] repose sur cette observation pour générer automatiquement des signatures consistant à rechercher ces motifs dans un ordre quelconque ou particulier. Une troisième catégorie de signature permet de pondérer les motifs en les complétant par d'autres motifs propres au code d'exploitation : octets de poids fort de l'adresse de retour, vecteur d'attaque, etc.

2.5 Data Tainting

Taintcheck [11] permet de suivre la propagation des données provenant d'une source non fiable (réseau) au sein d'un programme afin de déterminer si celles-ci sont utilisées dans des situations illégitimes : utilisation de données marquées (teintées) comme adresse de retour ou bien comme spécificateur de format. L'intérêt étant d'identifier les composantes de la charge spécifiques à l'attaque et de les utiliser par la suite dans des signatures générées automatiquement. TaintCheck ne requiert pas la recompilation du binaire. Il repose sur trois modules pour analyser la charge à destination du processus qu'il instrumente :

- *TaintSeed* qui permet de marquer tous les octets provenant d'une source non sûre ;
- *TainTracker* pour suivre la propagation des données teintées en marquant si nécessaire d'autres octets (ex. : dans le cas d'une instruction **mov**, les données de l'adresse de destination seront marquées si un des octets de l'adresse source est marqué) ;
- *TaintAssert* : permet de vérifier si un octet marqué est utilisé dans une situation illégitime (adresse de retour, chaîne de format, etc).

2.6 Data Mining

Nous concluons cette section par un aperçu sur des approches alternatives pour la détection des shellcodes polymorphes. Dans les travaux [5, 6], la détection s'opère par un apprentissage au préalable sur des échantillons de codes malveillants. Dans le premier article, l'apprentissage est assuré par le biais de réseaux de neurones tandis que dans le second, les auteurs suggèrent l'usage des chaînes de Markov. L'apprentissage a pour objectif de déterminer le jeu d'instructions utilisé pour construire le shellcode polymorphe. Les résultats expérimentaux obtenus montrent un taux de détection satisfaisant, et ce malgré la fonctionnalité anti *data mining* présente dans Clet.

Conclusion

Bien que polymorphes, les shellcodes peuvent être détectés efficacement si on se réfère aux résultats annoncés par les diverses solutions décrites tout au long de cet article. Dans certains cas, ceci est facilité par la faiblesse du générateur de shellcodes polymorphes, dans d'autres cas, il est nécessaire de recourir à l'artillerie lourde pour les détecter. En effet, recourir au désassemblage d'instructions à chaque paquet de données reçu risque d'avoir un impact non négligeable sur les performances réseaux. ■

RÉFÉRENCES

- [1] P. Akritidis, Evangelos P. Markatos, Michalis Polychronakis, and Kostas G. Anagnostakis. STRIDE : Polymorphic sled detection through instruction sequence analysis. In SEC, pages 375–392, 2005.
- [2] Theo Detristan, Tyll Ulenspiegel, Yann Malcom, and Mynheer Superbus Von Underduk. Polymorphic shellcode engine using spectrum analysis. Phrack Magazine, 61(9), 2003.
- [3] Christopher Krügel, Engin Kirda, Darren Mutz, William K. Robertson, and Giovanni Vigna. Polymorphic worm detection using structural information of executables. In RAID, pages 207–226, 2005.
- [4] James Newsome, Brad Karp, and Dawn Xiaodong Song. Polygraph : Automatically generating signatures for polymorphic worms. In IEEE Symposium on Security and Privacy, pages 226–241, 2005.
- [5] Udo Payer and Stefan Kraxberger. Polymorphic code detection with GA optimized markov models. In Communications and Multimedia Security, pages 210–219, 2005.
- [6] Udo Payer, Peter Teuffl, and Mario Lamberger. Hybrid engine for polymorphic shellcode detection. In Klaus Julisch and Christopher Krügel, editors, DIMVA, volume 3548 of Lecture Notes in Computer Science, pages 19–31. Springer, 2005.
- [7] Rix. Writing IA32 alphanumeric shellcodes. Phrack Magazine, 57(15), 2001.
- [8] Dragos Ruiu. Snort preprocessor - multi-architecture mutated NOP sled detector.
- [9] Mehdi Talbi, Mohamed Mejri, and Adel Bouhoula. Specification and evaluation of polymorphic shellcode properties using a new temporal logic. Journal in Computer Virology (JCV), 5(3) :171–186, August 2009.
- [10] Thomas Toth and Christopher Krügel. Accurate buffer overflow detection via abstract payload execution. In RAID, pages 274–291, 2002.
- [11] James Newsome and Dawn Xiaodong Song. Dynamic Taint Analysis for Automatic Detection, Analysis and Signature Generation of Exploits on Commodity Software
- [12] K2. ADMmutate. <http://www.ktwo.ca/>
- [13] Matias Sedalo. JempiScode. <http://www.dsis.com.ar/goodfellas/st0xff/>
- [14] Yuri Gushin. NIDS polymorphic evasion - The End. <http://www.ecl-labs.org/papers/ecl-poly.txt>
- [15] Archana Pasupulati, Jason Coit, Karl N. Levitt et al. Buttercup: on network-based detection of polymorphic buffer overflow vulnerabilities

creation agence Comelink - cr@leptoc.fr - La photo est une œuvre exclusive de Johann Locatelli (johann.locatelli@businessdecision.com) - 06 janvier 2016 à 09:44

DEVENEZ QUELQU'UN
DE RECHERCHÉ
POUR CE QUE
VOUS SAVEZ TROUVER.

FORMATIONS FORENSIQUES

Cours SANS Institute
Certifications GIAC



FOR 408

Investigation Infoforensique Windows

FOR 508

Analyse Infoforensique et réponses
aux incidents clients

FOR 558

Network Forensic

FOR 563

Investigations infoforensiques
sur équipements mobiles

Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr



www.hsc-formation.fr



PROPOSE 2 BADGES,
FORMATIONS SUR 7 MOIS SUR

REVERSE ENGINEERING SÉCURITÉ OFFENSIVE

DIPLÔMES ACCRÉDITÉS PAR LA CONFÉRENCE DES GRANDES ÉCOLES

Volume horaire total : 220 heures de cours - projets - ateliers - conférences | Ouverture : janvier 2014 | Durée totale : 7 mois | Lieu : Paris
Clôture des inscriptions : 13 décembre 2013

BADGE REVERSE ENGINEERING

Un BADGE pour être capable d'étudier tous les programmes,

- Analyse de codes malveillants
- Reverse et reconstruction de protocoles
- Protections logiciels et unpacking
- Analyse d'implémentations de cryptographie

BADGE SÉCURITÉ OFFENSIVE

Un BADGE pour trouver, exploiter, corriger les vulnérabilités dans un système :

- Détournement des protocoles réseaux non sécurisés
- Exploitation des corruptions mémoires et vulnérabilités web
- Escalade de privilèges sur un système compromis
- Intrusion, progression et prise de contrôle d'un réseau

