Pratique

ClamAV : le modèle de la moule appliqué à la virologie informatique...

Tomasz Kojm

orientés à voler les données ou à transformer l'ordinateur de la victime en une machine à sous, sont une réalité quotidienne. La lutte contre ce type de courriel a démontré que la meilleure stratégie consiste à éliminer directement les menaces au niveau des serveurs de messageries électroniques. Les logiciels Open Source nous viennent en aide en proposant un éventail de solutions pour lutter contre les spams et au moins un programme pour lutter directement contre les logiciels malveillants ; nous le décrirons dans cet article.

Degré de difficulté

Le problème du courriel non souhaité nous concerne tous, que ce soit les administrateurs systèmes ou les simples utilisateurs Internet ; ce courriel est devenu ces dernier temps, un *outil* de marketing tentant pour certaines sociétés et toute sorte de criminels, intéressés principalement par les numéros de nos cartes bancaires. Les attaques du type *phishing* ou les vers

Le projet Clam AntiVirus (ClamAV) est un ensemble d'outils antivirales, destinés aux systèmes Unix. Il est développé par une équipe internationale de plusieurs développeurs et disponible en licence GNU GPL v2. J'attire ici votre attention sur le fait que, à contrario d'autres programmes Open Source, ce logiciel nécessite des mises à jours régulières : aussi bien le code que la base des signatures de virus. En ce qui concerne les programmes anti-virus, ces éléments, logiciels et services, jouent un rôle clef.

Le mot clam signifie en anglais une moule. Les lecteurs se rappellent peut-être que les moules se nourrissent via une méthode de filtrage : pour survivre, elles doivent filtrer l'eau pour trouver des particules de nourriture. Cette brève explication devrait répondre aux questions concernant le nom du programme.

Outils de base

Le logiciel ClamAV s'adapte à un principe Unix connu, KISS (en anglais *Keep it Simple, Single*) donc l'éventail des programmes fournis dans le paquet source contient des applications spécialisées, simples d'emploi, en ligne de commande et qui peuvent servir de base aux systèmes complexes d'analyse.

Clamscan

Clamscan est un outil pratique permettant d'analyser les fichiers et les répertoires directement en ligne de commandes. Dans le cas le plus simple, son utilisation se limite à indiquer le nom du fichier ou du répertoire à scanner. Grâce à l'option qui permet d'appeler des programmes externes, il est capable de contrôler les archives non supportées par la bibliothèque *libclamav*. *Clamscan* peut être utilisé pour effectuer des tâches de routine, par exemple, scanner les répertoires d'utilisateurs ou s'intégrer aux autres

Cet article explique...

- Comment est constitué Clam AntiVirus,
- Comment utiliser la bibliothèque libclamav,
- Comment créer votre propre signature du virus pour ClamAV.

Ce qu'il faut savoir...

- Vous devriez avoir des connaissances de base sur les menaces en provenance de Internet,
- Vous devriez connaître les systèmes Unix et les services qu'ils proposent.

programmes afin de les doter de l'option de protection antivirale. Il faut toutefois souligner que clamscan doit charger les bases complètes de signatures à chaque démarrage, ce qui prend du temps et de la mémoire vive supplémentaire. Pour cette raison, il ne doit pas être utilisé dans le cadre d'opérations nécessitant de scanner de nombreux fichiers séparément et simultanément, tels que le courriel au niveau du démon SMTP.

Clamd

Clamd est un démon conçu pour les opérations où l'efficacité est importante. Le programme est multi-thread et il est donc capable d'utiliser les processeurs supplémentaires dans les systèmes SMP. Ceci garantit également une utilisation efficace de la base de signatures : elles sont chargées au démarrage et après une mise à jours, puis ensuite, partagées par tous les threads. Après le démarrage, *clamd* s'exécute en tâche de fond et attend des commandes (comme *SCAN*, *PING*, *RELOAD*, *STREAM*) en indiquant le socket Unix et/ou TCP dans le fichier de configuration. En ce qui concerne les systèmes Linux et FreeBSD, il est capable en plus de travailler avec le module Dazuko, permettant de scanner les ressources sélectionnées lorsqu'on y accède. Cette solution permet notamment de suivre les ressources des répertoires publics FTP lorsque le démon FTP ne supporte pas le scannage anti-virus.

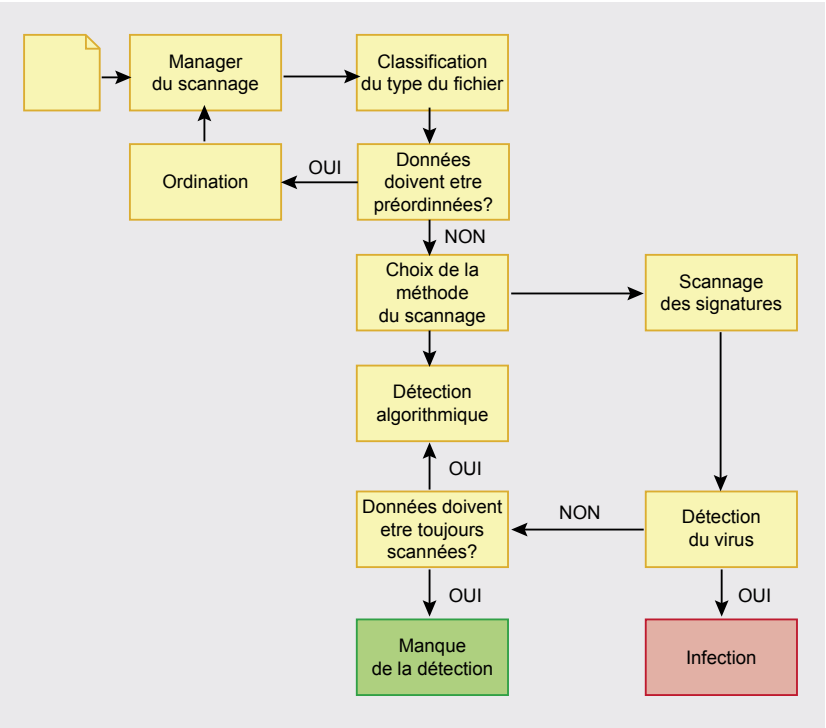

Figure 1. *Schéma simplifié du fonctionnement de la bibliothèque libclamav*

Clamdscan

Clamdscan est l'un des outils permettant d'utiliser les fonctionnalités du démon *clamd*. Le nom suggère qu'il a été créé d'après le premier outil décrit et c'est effectivement le cas. *Clamdscan* est capable de remplacer *clamscan* de manière efficace dans de nombreux cas. De manière efficace parce qu'il n'a pas besoin de charger à chaque fois la base de signatures, ce qui réduit considérablement le temps de démarrage et l'utilisation de la mémoire. *Clamdscan* ne contient aucune procédure de scannage et repose uniquement sur *clamd*, en lui envoyant les noms de fichiers et de répertoires, les flux de données et les descripteurs à scanner. Une restriction importante en résulte : lors de la vérification de fichiers ou de répertoires, *clamdscan* est capable de vérifier uniquement ceux qui sont accessibles au démon clamd.

Listing 1. *Détection algorithmique du virus utilisant une simple cryptanalyse*

```
/* W32.Parite.B */
if(SCAN_ALGO && !dll && ep == EC32(section_hdr[nsections - 1].PointerToRawData)) {
   lseek(desc, ep, SEEK_SET);
   if(cli_readn(desc, buff, 4096) == 4096) {
      const char *pt = cli_memstr(buff, 4040, "\x47\x65\x74\x50\x72\x6f\x63\x41\x64\x64\x72\x65\x73\x73\x00", 15);
      if(pt) {
         uint32_t dw1, dw2;
         pt += 15;
         if(((dw1 = cli_readint32(pt)) ^ (dw2 = cli_readint32(pt + 4))) == 0x505a4f &&
         ((dw1 = cli_readint32(pt + 8)) ^ (dw2 = cli_readint32(pt + 12))) == 0xffffb &&
         ((dw1 = cli_readint32(pt + 16)) ^ (dw2 = cli_readint32(pt + 20))) == 0xb8) {
            *ctx->virname = "W32.Parite.B";
            free(section_hdr);
            return CL_VIRUS;
         }
      }
   }
}
```

Freshclam

Le paquet ClamAV contient un programme appelé *freshclam*, chargé d'automatiser complètement le processus d'actualisation des bases de signatures des virus. Le programme propose une série d'options : il peut être lancé à la demande ou travailler en tâche de fond en tant que démon, il est capable d'utiliser un serveur proxy, il permet d'exécuter des programmes externes en fonction d'événement (actualisation des bases, erreur, détection d'une nouvelle version du logiciel). Afin de contrôler la disponibilité d'une nouvelle version de la base de signatures, *freshclam* se sert du protocole DNSet plus précisément d'un enregistrement texte spécial :

```
$ host -t txt current.cvd.clamav.net
current.cvd.clamav.net TXT
   "0.88.4:39:1640:1155043741:1"
```

Cet enregistrement contient également l'information sur la version stable actuelle de ClamAV, ce qui permet à *freshclam* d'informer l'administrateur d'une nouvelle sortie du programme.

Sigtool

Le dernier outil décrit est destiné avant tout aux personnes travaillant sur les signatures de virus et à l'actualisation des bases dans le projet ClamAV. *Sigtool* permet de décompresser, de vérifier et de créer des bases de signatures signées de manière numérique, de générer les fichiers différentiels (*.cdiff), de récupérer le code des macros depuis les fichiers Office ou de normaliser les fichiers HTML, en facilitant la création des signatures correctes.

LibClamAV

La bibliothèque *libclamav* est l'élément responsable de détecter les virus. Elle est *thread-safe* et propose une API simple (en anglais *Application Programming Interface*) permettant aux programmes d'accéder aux fonctions de scannage universelles. La Figure 1 présente un schéma général du fonctionnement de la bibliothèque et nous décrirons toutes les étapes ci-après. L'article se concentre sur la version actuelle du logiciel dans le repository CVS, qui remplacera prochainement la version stable 0.8x.

Tableau 1. *Tableau des caractères de remplacement*

Caractère de remplacement	Sens de correspondance
*	N'importe quel nombre de n'importe quel caractère
??	N'importe quel caractère
{n}	n de n'importe quels caractères
{-n}	Au plus n de n'importe quels caractères
{n-}	Au moins n de n'importe quels caractères
(a|b|c)	a ou b ou c

Tableau 2. *Tableau des caractères de remplacement*

Code du fichier cible	Type du fichier
0	Libre
1	Portable Executable
2	Composant OLE2 (par exemple, le script VBA)
3	HTML normalisé
4	De messagerie électronique
5	graphique
6	ELF

Déterminer le type du fichier

Pour des raisons de sécurité et le travail direct sur les descripteurs, *libclamav* analyse le contenu du fichier et non son nom dans le système de fichiers pour déterminer le type du fichier. Des choses banales en apparence peuvent être parfois très difficiles à résoudre. Autant la reconnaissance d'un fichier exécutable ELF ne pose aucun problème, autant déterminer si un fichier donné est une archive tar demande plus d'effort. Puisque le succès du scannage dépend directement du résultat de ce processus, il doit être le plus exact possible. Pour ce faire, la bibliothèque *libclamav* utilise trois techniques :

- les tests standards qui vérifient les nombres magiques (en anglais *magic numbers*) : les valeurs uniques à l'intérieur du fichier, caractéristique pour un type donné,
- la détection algorithmique, basée sur l'analyse structurelle ou le contenu du fichier,
- le scannage du fichier en fonction des fragments caractéristiques de données (par exemple, les fragments du code HTML).

Gestion des fichiers spéciaux

Lorsque le type du fichier est déterminé, il faut décider comment le gérer. Les fichiers spéciaux, comme les archives ou les documents, nécessitent une préparation préalable qui consiste à extraire les données. La bibliothèque est dotée de la gestion des types suivants :

- archives : zip, RAR, CHM (en anglais *Compiled HTML*), cabinet, OLE2, tar, SIS (paquets d'installation pour le système Symbian),
- fichiers compressés : gzip, bzip2, compress,
- fichiers de la messagerie électronique : tous les formats Unix populaires, TNEF (*winmail.dat*), PST,
- documents : HTML, PDF, MS Office,
- fichiers exécutables : PE (ainsi que la gestion des compresseurs

ou protecteurs UPX, FSG, Petite, PESpin, WWPack32, Y0da Cryptor), ELF,
- d'autres : JPEG, GIF, RIFF (analyse des fichiers en fonction de présence des exploits).

Les modules de gestion des fichiers ont été créés de sorte à détecter les irrégularités qui pourraient être utilisées à induire en erreur le scanneur. À titre d'exemple, les procédures de gestion des fichiers zip sont capables de détecter une manipulation dans les en-têtes locaux des archives, les entrées fausses ou cachées. Un élément supplémentaire de protection sont des limites, liées au niveau de récursivité, au nombre et à la taille des fichiers décompressés, qui constituent une protection contre les attaques du type *Denial of Service*, utilisant les bombes d'archives (petites archives contenant un grand nombre de fichiers compressés). Les informations supplémentaires placées dans les archives (méta-données) sont également utiles ; *libclamav* les scanne à l'aide des signatures spéciales, permettant ainsi de détecter dans certains cas un logiciel malveillant au sein des archives codées.

Scannage à l'aide des signatures

Afin de scanner les fichiers de manière efficace à la recherche des dizaines de milliers de virus, *libclamav* utilise deux algorithmes : Aho-Corasick et un algorithme étendu de Boyer-Moore. Il s'agit des algorithmes de correspondance multi-modèles, permettant de parcourir les fichiers à l'aide de nombreuses signatures simultanément.

De plus, la bibliothèque supporte les accélérateurs matériels qui accélèrent donc considérablement le scannage.

L'algorithme Aho-Corasick créé un arbre spécial d'après un ensemble de modèles (dans notre cas, un ensemble de signatures de virus) ; cet arbre sera ensuite transformé en un automate fini.

La Figure 2 présente un exemple d'un arbre et d'un automate lui correspondant, créés pour un ensemble de modèles `P = {GAT, GC, TGCT}`. Les états de l'automate, marqués de couleur verte, sont des états finis – lorsque l'automate se trouve en cet état, l'un des modèles a été correctement adapté. Les flèches noires (bords de l'arbre) représentent une fonction de transfert et permettent de modifier l'état lorsque le caractère de sortie n'est pas accepté. Les flèches rouges représentent en revanche une fonction d'échec et permettent de modifier l'état lorsque ceci est impossible à faire via la fonction de transfert. L'automate commence à fonctionner dans l'état zéro (qui est une racine de l'arbre et le premier état de l'automate) et effectue les modifications nécessaires de l'état pour chaque caractère du texte d'entrée. Dans l'exemple décrit, la recherche du texte se déroule en temps linéaire – chaque caractère du texte d'entrée n'est vérifié qu'une seule fois. L'implémentation de l'algorithme en ClamAV ne garantit pas ce temps de fonctionnement de l'automate parce que des restrictions de profondeur de l'arbre A-C ont été ajoutées pour limiter l'utilisation de la mémoire. L'efficacité est toutefois satisfaisante. De plus, l'algorithme bénéficie d'une gestion des cartes génériques (en anglais *wild cards*) décrites dans le prochain point.

L'algorithme de Boyer-Moore étendu utilise, de même que la version originale, l'idée de la table de décalage (en anglais *shift table*), permettant d'omettre une partie du texte d'entrée dans lequel la correspondance ne peut pas avoir lieu. La partie essentielle de l'algorithme ne travaille pas sur les caractères individuels mais sur les hachages calculés à partir de trois caractères consécutifs. Ces caractères servent d'index du tableau de décalage et d'un tableau spécial de suffixes. Pour cette raison, l'algorithme n'est utilisé que pour les signatures statiques (qui ne contiennent pas de caractères de remplacement).

Le processus de scannage à l'aide des signatures peut être accéléré au moyen d'un accélérateur matériel. ClamAV supporte la plate-forme NodalCore de Sensory Networks qui propose les cartes PCI dont le débit s'élève à 2 Gb/s. Leur utilisation accélère le scannage à proprement parler et en plus réduit la charge CPU, ce qui est important pour les systèmes de filtrage des milliers ou des millions des fichiers quotidiens. Vous trouverez les données techniques et les prix des cartes sur les sites Web des producteurs.

Tableau 3. *Tableau des décalages acceptables*

Décalage	Lieu de correspondance
*	N'importe quel endroit dans le fichier
n	n-ème octet
EOF-n	n-ème octet depuis la fin du fichier
EP+n (ce décalage et les décalages ci-après ne fonctionnent qu'avec les fichiers exécutables PE et ELF)	n-ème octet depuis Entry Point
EP-n	n-ème octet avant Entry Point
Sx+n	n-ème octet depuis le début de la section portant le numéro x
Sx-n	n-ème octet avant le début de la section portant le numéro x
SL+n, SL-n	Comme ci-dessus où L correspond à la dernière section du fichier

Types des signatures des virus

Libclamav supporte plusieurs types de signatures, permettant de décrire le logiciel malveillant de plusieurs manières.

Le fragment du code qui identifie le but est un élément essentiel des signatures standards. Afin d'éviter les fausses alertes et de permettre de stocker les entrées dans les fichiers textes, il est codé sous forme d'une entrée composée des nombres hexadécimaux ; il peut également contenir des caractères de remplacement décrits dans le Tableau 1. Ces caractères permettent d'augmenter la souplesse de la signature.

Les signatures standards peuvent avoir deux formes : simplifiée et celle qui permet de décrire le but plus en détails. Dans le premier cas, il s'agit des signatures placées dans les bases **.db* au format suivant :

```
NomDuVirus=SignatureHexadécimale
```

Les signatures sous forme développée se trouvent dans les fichiers **.ndb* et se présentent de manière suivante :

```
NomDuVirus:CodeDeLobjectif:Offset:
   SignatureHexadécimale[:
      VersionMinimaleDuMoteur:
   [WSMaximal]]
```

Elles permettent d'indiquer le type du fichier et l'endroit où la correspondance de la signature doit avoir lieu. Les Tableaux2 et 3 présentent les valeurs possibles pour le deuxième et troisième champ.

En ce qui concerne les logiciels malveillants qui ne modifient pas leurs fichiers, tels que la plupart de simples chevaux de Troie, il est possible d'opter pour les signatures utilisant des sommes MD5. Elles sont placées dans les fichiers **.hdb* et ont le format suivant :

```
MD5:TailleDuFichier:Nom
```

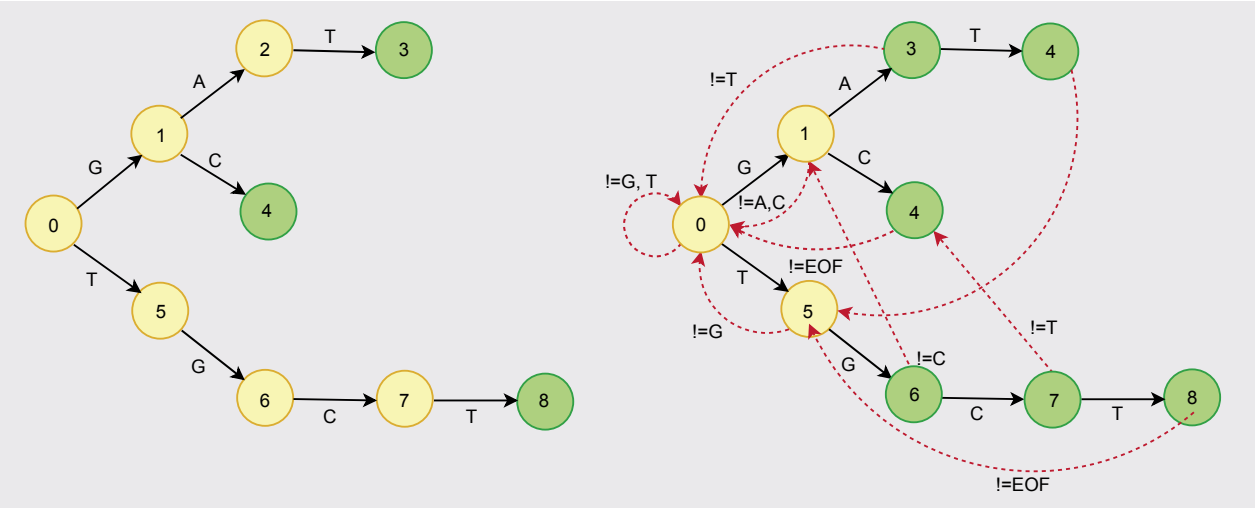

Figure 2. *Exemple d'arbre et d'automate Aho-Corasick*

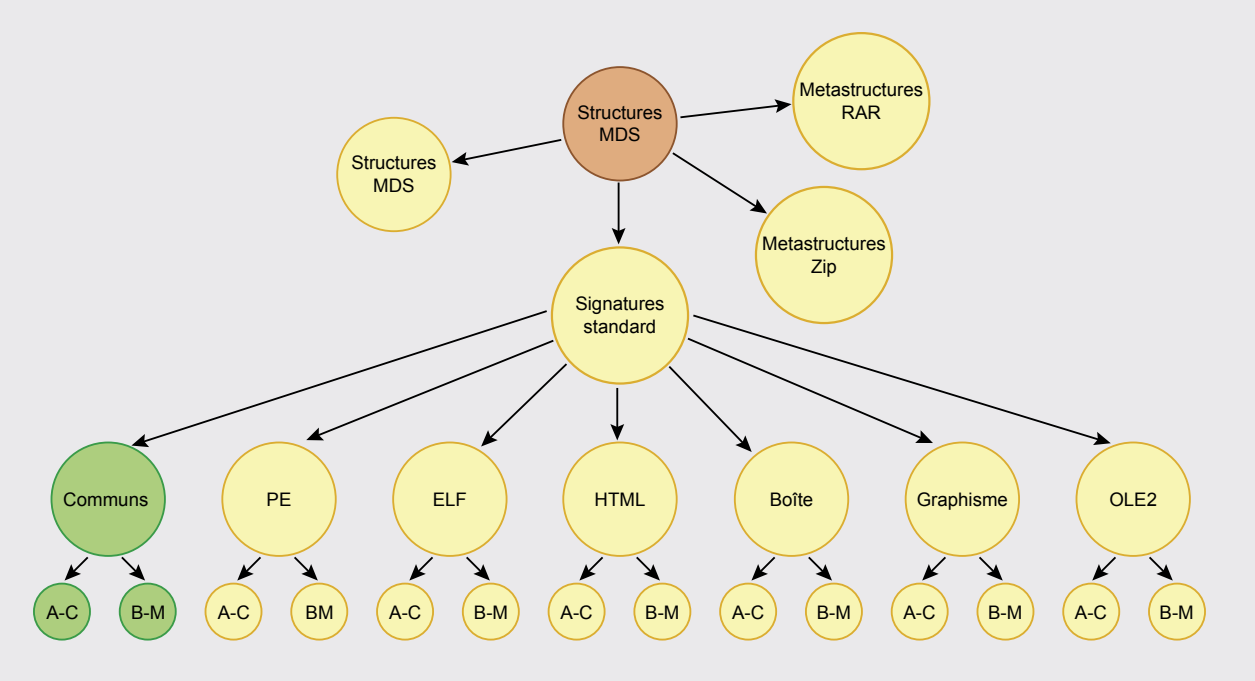

Figure 3. *Arbre de signatures en mémoire*

Listing 2. *Exemple d'utilisation de la bibliothèque libclamav*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <clamav.h> /* fichier en-tête de la bibliothèque libclamav */
int main(int argc, char **argv) {
   int fd, ret;
   unsigned long int size = 0;
   unsigned int sigs = 0;
   long double mb;
   const char *virname;
   struct cl_engine *engine = NULL;
   struct cl_limits limits;
   if(argc != 2) {
      printf("Usage: %s file\n", argv[0]);
      exit(2);

   }
   if((fd = open(argv[1], O_RDONLY)) == -1) {
      printf("Can't open file %s\n", argv[1]);
      exit(2);
   }
   /* charger toutes les bases depuis le répertoire par défaut */
   if((ret = cl_load(cl_retdbdir(), &engine, &sigs, CL_DB_STDOPT))) {
      printf("cl_load: %s\n", cl_strerror(ret));
      close(fd);
      exit(2);

   }
   printf("Loaded %d signatures.\n", sigs);
   if((ret = cl_build(engine))) {    /* Compiler le moteur */
      printf("Engine compilation error: %s\n", cl_strerror(ret));;
      cl_free(engine);
      close(fd);
      exit(2);
   }
   memset(&limits, 0, sizeof(struct cl_limits));   /* paramétrer les limites des archives*/
   limits.maxfiles = 1000; /* nombre maximal des fichiers à décompresser */
   limits.maxfilesize = 10 * 1048576; /* taille maximale du fichier archivé */
   limits.maxreclevel = 5; /* niveau maximal des récursivités */
   limits.maxratio = 200; /* rapport maximal entre la taille du fichier compressé et l'original */
   /* scanner le descripteur indiqué */
   if((ret = cl_scandesc(fd, &virname, &size, engine, &limits, CL_SCAN_STDOPT)) == CL_VIRUS) {
      printf("Virus detected: %s\n", virname);
   } else {
      if(ret == CL_CLEAN) {
         printf("No virus detected.\n");
      } else {
         printf("Error: %s\n", cl_strerror(ret));
         cl_free(engine);
         close(fd);
         exit(2);
      }
   }
   close(fd);
   /* calculer une taille approximative des données scannées */
   mb = size * (CL_COUNT_PRECISION / 1024) / 1024.0;
   printf("Size of scanned data: %2.2Lf MB\n", mb);
   cl_free(engine);   /* libérer la mémoire occupée par le moteur */
   exit(ret == CL_VIRUS ? 1 : 0);
}
```

Grâce à cette solution, les utilisateurs peuvent créer facilement leurs propres signatures, il suffit d'utiliser l'option *--md5* du programme *sigtool*

```
:sigtool --md5 evil.exe > evil.hdb
```

et déplacer ensuite le fichier *evil.hdb* dans le répertoire des bases de signature de ClamAV.

Le dernier type de signatures est destiné à décrire les fichiers contenus dans des archives et pour ce faire, il se sert des méta-données susmentionnées contenues dans les archives. Il est utilisé avant tout pour les vers Internet qui se servent des archives codées en tant que support et dont les fichiers gardent une propriété caractéristique, telle que le nom, la taille ou la somme de contrôle. Ces signatures sont stockées dans les fichiers *.zmd* et *.rmd*, respectivement pour les archives zip et RAR, au format suivant :

```
NomDuVer : codé(0,1) :
   nom du fichier :
   taille originale :
   taille après la compression :
   crc32 : méthode de compression :
   numéro de série du fichier dans
      l'archive:nombre maximal
      des archives intégrées
```

Si certains paramètres changent, il est possible d'utiliser un astérisque pour ignorer leurs valeurs. Worm.Padowor.A, Worm.Kimazo.A et les variantes du ver Bagle sont des exemples des vers détectés avec le succès à l'aide de cette méthode spécifique.

Afin de les utiliser de manière optimale lors du démarrage, les bases de signatures se divisent en groupes appropriés, en créant un arbre présenté sur la Figure 3. Dans un premier temps, les données sont scannées à l'aide des signatures correspondant à leur type et ensuite, à l'aide des signatures communes et MD5. Les signatures utilisant les méta-données sont contrôlées au moment de décompresser les archives.

En ce qui concerne les fichiers, dans un premier temps, on les scanne à l'aide des signatures correspondant à leur type et ensuite, à l'aide des signatures communes et MD5.

Fichiers CVD

Les fichiers CVD constituent un support de base pour les signatures de ClamAV ; ces fichiers sont en réalité des archives tar compressées et signées de manière numérique. Deux fichiers sont distribués : *daily.cvd* et *main.cvd*. Le premier d'entre eux est une base qui sert à la mise à jour quotidienne, le second est une archive principale de signatures dans laquelle une partie d'entrées est déplacée tous les 45 jours en moyenne depuis le fichier *daily.cvd*. Les bases contiennent les fichiers de texte avec les signatures aux formats décrits ci-dessus. Afin de réduire la charge des miroirs, un système d'actualisations différentielles a été élaboré (grâce auquel il n'est plus nécessaire de charger les fichiers complets de la base de signatures). À la place, *freshclam* charge un script spécial avec les dernières modifications et effectue les mises à jour appropriées dans les fichiers locaux.

Détection algorithmique des virus

Certains virus avancés nécessitent une approche spéciale. Ce point concerne notamment les virus polymorphiques qui rendent impossible la détection directe à l'aide des signatures. Une partie des scanneurs antivirus essayent d'utiliser une approche universelle basée sur l'émulation du code pour les détecter mais avec les virus modernes utilisant des astuces avancées ou les nouvelles extensions de processeurs, cette solution peut s'avérer inefficace. ClamAV propose une détection précise basée sur les algorithmes dédiés. Cette méthode prend plus de temps mais elle permet de détecter précisément et rapidement un virus. Actuellement, tous les algorithmes doivent être placés directement dans les sources du programme mais il sera bientôt possible de les distribuer sous forme de code d'octets (en anglais bytecode) avec les signatures standards. Le Listing 1 présente un court fragment du code chargé de détecter le virus *W32.Parite.B*, Je vous invite à lire davantage les sources si vous vous intéressez aux exemples plus complexes.

Exemple d'utilisation de la bibliothèque

La bibliothèque *libclamav* permet d'ajouter facilement un scannage antivirus au logiciel existant. Le Listing 2 présente un exemple complet d'utilisation de la bibliothèque – un simple outil permettant de scanner des fichiers individuels depuis la ligne de commandes. Les fonctions élémentaires y ont été utilisées : `cl_load()` charge une base ou toutes les bases depuis le répertoire indiqué, `cl_retdbdir()` retourne le chemin du répertoire par défaut avec les bases, `cl_build()` compile le moteur, `cl_scandesc()` scanne le descripteur indiqué, `cl_strerror()` traduit le code d'erreur en message en anglais et enfin, `cl_free()` libère la mémoire occupée par le moteur. Le programme retourne le code 1 s'il détecte une infection, 0 s'il ne la détecte pas et 2 s'il y a une erreur.

Infrastructure réseau

Le projet ClamAV est doté d'une infrastructure avancée de miroirs, développée depuis plusieurs années. Elle est financée par des entreprises, des organisations et des organismes éducatifs qui rendent disponibles leurs connexions Internet pour les besoins de distribution de bases

À propos de l'auteur

Tomasz Kojm est concepteur et leader du projet Clam AntiVirus. Il est diplômé de la faculté d'informatique de l'Université Mikołaj Kopernik à Toruń ; c'est un enthousiaste des logiciels Open Source et des tortues.

Sur Internet

- *http://www.clamav.net/* – site officiel du projet Clam AntiVirus,
- *http://www.dazuko.org/* – site officiel du projet Dazuko,
- *http://www.sensorynetworks.com/* – site du producteur des accélérateurs matériels.

avec les signatures de virus. Plus de cent vingt serveurs dans une quarantaine de pays garantissent une actualisation rapide et sans problèmes des bases. Pour un accès optimal aux mises à jour, il faut se référer aux adresses suivantes :

- *db.XY.clamav.net* – indique les miroirs disponibles dans le pays dont le code est XY,
- *db.local.clamav.net* – essaie de rediriger le client à l'ensemble de miroirs le plus proche en vérifiant son adresse dans la base GeoIP,
- *database.clamav.net* – l'enregistrement round-robin indique l'ensemble des miroirs les plus rapides et les plus disponibles.

En ce qui concerne la France, la configuration recommandée consiste à placer deux entrées suivantes dans le fichier *freshclam.conf* :

```
DatabaseMirror db.fr.clamav.net
DatabaseMirror database.clamav.net
```

Si la connexion avec les miroirs nationaux échoue, *freshclam* utilisera automatiquement la deuxième entrée.

Applications possibles

Bien que ClamAV ait été conçu avant tout pour scanner les courriels, il se prête bien aux autres applications. Le site officiel du projet, dans la section *Third-party software*, vous propose une liste d'une centaine de programmes qui utilisent ClamAV, permettant de filtrer les courriels au sein des protocoles SMTP, POP3 et les lecteurs de messages électroniques, de scanner le système de fichiers, les flux HTTP, les ressources FTP et d'autres.

Conclusion

Un scanneur anti-virus constitue un des outils élémentaires que se doit de disposer les administrateurs réseau. Clam AntiVirus est un logiciel souple, offrant de larges fonctionnalités et applications ; grâce à sa licence et à ses mises à jours régulières, les administrateurs peuvent découvrir que la protection anti-virus ne doit pas forcément être onéreuse. ●